2015 IEEE 56th Annual Symposium on Foundations of Computer Science

Competitive Flow Time Algorithms for Polyhedral Scheduling

Sungjin Im
EECS Department,
University of California at Merced
Merced, CA, USA 95344.
Email: sim3@ucmerced.edu

Janardhan Kulkarni Kamesh Munagala
Department of Computer Science,
Duke University
Durham, NC, USA 27708.
Email: {kulkarni,kamesh}@cs.duke.edu

Abstract

Many scheduling problems can be viewed as allocating rates to jobs, subject to convex packing constraints on the rates. In this paper, we consider the problem of rate allocation when jobs of unknown size arrive online (*non-clairvoyant* setting), with the goal of minimizing weighted delay or flow time. Though this problem has strong lower bounds on competitive ratio in its full generality, we show positive results for natural and fairly broad sub-classes. More specifically, the subclasses we consider not only generalize several well-studied models such as scheduling with speedup curves and related machine scheduling, but also capture as special cases hitherto unstudied scheduling problems such as *routing multi-commodity flows*, *routing multicast (video-on-demand) trees*, and *multi-dimensional resource allocation*.

We establish several first positive results by making connections with two disparate disciplines: Economics and Queueing theory. First, we view the instantaneous allocation of rates as a resource allocation problem. We analyze the natural *proportional fairness* algorithm from economics. To do this, we extend results from market clearing literature, particularly the Eisenberg-Gale markets and the notions of Walrasian equilibria and Gross Substitutes. This yields the first constant competitive algorithm with constant speed augmentation for single-sink flow routing, routing multicast trees, and multidimensional resource allocation with substitutes resources.

Next, we consider the general scheduling problem with packing constraints on rates, but with the restriction that the number of different *job types* is fixed. We model this problem as a non-stochastic queueing problem. We generalize a natural algorithm from queueing literature and analyze it by extending queueing theoretic ideas. We show that the competitive ratio, for any constant speed, depends polynomially only on the number of job types. Further, such a dependence on the number of job types is unavoidable for non-clairvoyant algorithms. This yields the first algorithm for scheduling multicommodity flows whose competitive ratio depends polynomially on the size of the underlying graph, and not on the number of jobs.

I. Introduction

The Packing Scheduling Problem or PSP was first defined in [24] as a natural generalization of many scheduling problems considered in literature. In this general problem, a scheduling instance consists of n jobs, and each job j has weight w_j, size p_j, and arrives at time r_j. At any time instant t, the scheduler must assign rates $\{y_j\}$ to the current jobs in the system. We assume the set of feasible rates $\mathbf{y}$ is constrained by a downward-closed convex region $\mathcal{P}$. Here, downward-closed means that if $\mathbf{y} \in \mathcal{P}$, then any $\mathbf{z} \in \mathcal{P}$, where $\mathbf{z}$ is coordinate-wise at most $\mathbf{y}$. Let $y_j^{\mathcal{A}}(t)$ denote the rate at which job j is processed at time t by a scheduler/algorithm $\mathcal{A}$. 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 $\int_{t=r_j}^{t'} y_j^{\mathcal{A}}(t)\mathrm{d}t \geq p_j$. Job j's flow time is the length of time job j waits to be completed since its arrival and is defined as $F_j^{\mathcal{A}} = C_j^{\mathcal{A}} - r_j$. Similalry, 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.

In this paper, we will focus on the *flow time objective*, specifically the objective of minimizing total weighted flow time, $\sum_j w_j F_j$. The class of scheduling algorithms we consider are constrained by several properties. It is *online* and learns about job j only when it arrives. Before this point, $y_j = 0$. The scheduler is required to be *non-clairvoyant*, *i.e.*, it does not know a job's size p_j until completing the job. Furthermore, the algorithm is allowed to re-compute $\mathbf{y}(t)$ at any real time t. Our algorithms will however perform recomputations only when jobs either arrive or complete.

As discussed in [24], the PSP problem is very general and captures several widely studied scheduling problems, such as broadcast scheduling [19], [7], [8], [15], [26], [6], unrelated machine scheduling [4], [23], [24], [25], and single machine scheduling with speedup curves [38], [13], [15], [16], [18]. Furthermore, the PSP problem becomes relevant in the context of modern data centers. Consider a typical data center setting, where there is a cluster of machines with a distributed file system implementation (such as HDFS [39]) layered on top of the cluster. Users submit executables (or jobs) to this cluster. In a typical MapReduce implementation such as Hadoop [2], each

0272-5428/15 $31.00 © 2015 IEEE
DOI 10.1109/FOCS.2015.38

job is a collection of parallel map and reduce tasks requiring certain CPU, disk space, and memory to execute. The job therefore comes with a request for different types of resources (or dimensions); these can either be explicitly specified, or can be estimated by the task scheduler from a high-level description of the job. The rate at which jobs execute is a pre-specified concave *utility function* of the resource allocation. This *multi-dimensional resource allocation* scenario has gained a lot of attention recently; see [20] and followup work [14], [41], [21], [3], [1], [37], [32]. For many commonly arising resource allocation functions, this problem is indeed a special case of PSP.

We will use the standard notion of *competitive ratio* for analyzing our algorithms. An online algorithm is said to be α-competitive if for every finite input instance, the cost incurred by the algorithm is at most α times the cost of an optimal offline solution to the instance.

A. High-level Overview of Results

The generality of the PSP problem leads to strong lower bounds: It is shown in [24] that there is an instance of PSP for which no deterministic algorithm is $O(n^{1-\epsilon})$-competitive on the flow time objective for any constant $0 < \epsilon < 1$, unless the algorithm runs with $\Omega(\sqrt{\log n})$ speed compared to the optimal algorithm. The above negative result is in sharp contrast with positive results for special cases such as broadcast scheduling and unrelated machine scheduling. This motivates the following question:

> Are there natural sub-classes of PSP for which it is possible to design competitive algorithms for flow time? Can these algorithms be designed using a few unifying algorithmic techniques?

In this paper, we answer both of these questions in the affirmative. We consider three natural and fairly broad classes of scheduling problems that not only generalize several well-studied models such as scheduling with speedup curves and related machine scheduling [23], [24], [25], but also capture as special cases hitherto unstudied problems such as *routing multi-commodity flows*, *routing multicast trees*, and *the multi-dimensional resource allocation* problem discussed above. We use two different viewpoints to make progress on these problems. In each case, we build on results in an entirely different field, by combining them with new technical ideas.

- Resource Allocation View: In the PSP problem, the instantaneous allocation of rates to jobs can be viewed as a resource allocation problem. A natural algorithm for performing resource allocation is the proportional fairness (PF) algorithm from economics [36], [11], [31]. In [24], we used KKT conditions for this program to analyze its completion time. In this paper, we consider the harder question of flow time, and this requires us to view the PF algorithm explicitly as a market clearing mechanism. We build on the notions of Gross Substitutes [22] and Eisenberg Gale markets [28] to show properties of the PF algorithm, and use it to show $O(1)$ competitiveness for a well-characterized sub-class of the PSP problem that we term MONOTONE PSP. This captures resource allocation with substitutes, flow routing to a single sink, related machine scheduling, and routing multicast trees (video-on-demand).
- Queueing View: A practically interesting version of PSP is when there are a limited number of job types. This can be viewed as a (non-stochastic) queueing problem that we term PSP-Q, and we adapt the well-known MAX-WEIGHT algorithm [40], [34], [33] from queueing theory. In contrast with stability analysis from queueing literature, our arrival process is adversarial, which necessitates a normalization step in the algorithm design, as well as a more careful analysis than standard stability arguments – roughly speaking, some quantities, that are assumed to remain constant in queueing analysis due to the fixed underlying stochastic nature of arrivals, may change dynamically in the adversarial setting. Our final competitive ratio depends polynomially on K, the number of queues (or job types).

Analysis Techniques.: Our algorithms involve solving a convex optimization problem each time instant on the set of jobs in the system, in order to perform rate allocation. As observed in [24], in contrast with single-machine scheduling settings, there is no closed form for the rates. In each case, we start with the primal optimality conditions (Propositions II.1 and III.1) connecting the rates found by our algorithm against any other rate vector. We then use the framework of amortized local competitiveness [27], where we define a potential function on the difference between the algorithm's job set and the optimal solution's set. Our potential naturally generalizes potentials used for single machine scheduling [18]; however, our analysis becomes different in that it uses the optimality conditions, sometimes iteratively, to show competitive ratio.

We note that our optimality conditions are on the convex program solved by the algorithm to perform instantaneous rate allocation, and this approach is different from previous work [4], [24] that uses the dual of the optimal offline relaxation, via dual fitting or a primal-dual approach. For the PSP problem, our previous work [24] showed a $O(1)$ competitive ratio for the completion time objective by a dual fitting analysis. However, we do not know how

to extend it for analyzing the harder flow time objective, since this requires highly structured dual variables for the instantaneous convex program, which may not exist. In contrast, our approach directly works with the primal optimality conditions of the instantaneous convex program, which yields a new analysis framework to the best of our knowledge.

Before we present more details, we make a brief digression to describe *speed augmentation analysis*. In our model, job arrivals are adversarial in nature. This implies strong lower bounds on the competitive ratio for all problems we consider; in fact, strong lower bounds arise in even simpler single-dimensional settings. We therefore perform a now standard speed augmentation analysis [29], where we assume the online algorithm can perform $c > 1$ allocations per time step, while OPT is restricted to performing allocations at rate 1. We will design algorithm that achieve bounded competitive ratio on flow time, using some speed c that will be an absolute constant. In fact, we will attempt to make c as close to 1 as possible. If the algorithm is competitive for speed $(1+\epsilon)$ where $\epsilon > 0$ can be made arbitrarily small, then it is termed *scalable*. We note that though we use speed augmentation for analysis, our algorithms are simple and natural. Further most of our algorithms are oblivious to this augmentation.

B. Resource Allocation View and MONOTONE PSP

We first consider the economics viewpoint where the scheduler is performing resource allocation among the active jobs at each time instant. The proportional fairness (PF) algorithm assigns rates y_{jt} as follows:

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

Note that $\mathcal{P}$ is a convex set in the definition of PSP. The PF algorithm generalizes the weighted round robin (WRR) algorithm for single machine scheduling. Though this algorithm is constant competitive for completion time objective [24], the same is not true for flow time; the best previous result requires the speed to depend on the number of jobs. We therefore analyze the PF algorithm under a natural restriction on the utility functions. Suppose the current set of jobs is S. Let $y_j(S)$ denote the rate allocated by PF to job $j \in S$.

Definition I.1 (Monotonicity of PF)**.** *The PF algorithm is said to be* monotone *if for any S and $\ell \notin S$, we have the following condition. For all $j \in S$, $y_j(S) \geq y_j(S \cup \{\ell\})$. The class* MONOTONE PSP *is the sub-class of PSP for which the PF algorithm is monotone.*

Our main result is the following theorem, which we prove in Section II.

Theorem I.2. *For the* MONOTONE PSP *problem, for any constant $\epsilon \in (0, 1/2)$, PF is $(e+\epsilon)$-speed, $O\left(1/\epsilon^2\right)$ competitive for minimizing weighted flow time.*

As mentioned before, we show the above theorem by amortized local competitiveness [27]. The potential function we use is a natural generalization of the potential for single-machine weighted round robin (WRR) considered in [18]; nevertheless, our analysis is very different since we don't have a closed form for the allocations or rates. We crucially use the optimality condition for the PF convex program (Proposition II.1), which we apply iteratively considering the first k jobs in arrival order versus the first $k+1$ jobs. This yields a simple analysis that works for any MONOTONE PSP instance.

The next question is to identify problem classes that belong to MONOTONE PSP. [1] We present two broad sub-classes of PSP that are monotone. Showing monotonicity requires making connections to market clearing literature, particularly Walrasian equilibria and Gross Substitutes [22], and the *Submodular Utility Allocation* markets defined in [28].

1) Resource Allocation with Substitutes (RA-S): We revisit the multi-dimensional resource allocation problem considered above. Formally, there are D divisible resources (or dimensions), numbered $1, 2, \ldots, D$. We assume w.l.o.g. (by scaling and splitting resources) that each resource is available in unit supply. If job j is assigned a non-negative vector of resources $\mathbf{x} = \{x_1, x_2, \ldots, x_D\}$, then the rate at which the job executes is given by $y_j = u_j(\mathbf{x})$, 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, so that $\sum_j x_{jd} \leq 1$ for all $d \in \{1, 2, \ldots, D\}$. A well-studied special

[1] We note that no applications mentioned in Section 1.2 of [24] are captured by monotone PSP in their full generality. However, some special cases such as the related machines scheduling do fall into the category of monotone PSP.

class of utilities in the resource allocation literature are the Constant Elasticity of Scale (CES) utilities, given by:

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

A parameter range of special interest is when $\rho \in (0, 1]$ – these utility functions are widely studied in economics, and capture resources that are *imperfect substitutes* of each other, where the parameter ρ captures the extent of substitutability. A special case as $\rho \to 0$ is termed *Cobb-Douglas* utilities: $u_j(\mathbf{x}_j) = \prod_{d=1}^{D} x_{jd}^{\alpha_{jd}}$, 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 [42]. When $\rho = 1$, CES utilities reduce to *linear utilities*.

In this paper, we generalize CES functions to a broader class that we term *resource allocation with substitutes* or RA-S.

$$u_j(\mathbf{x}_j) = \left(\sum_{d=1}^{D} (f_{jd}(x_{jd}))^{\rho_j} \right)^{1/\rho'_j} \qquad \text{where} \quad \rho_j \in (0, 1] \text{ and } \rho'_j \geq \rho_j \tag{2}$$

Here, the $\{f_{jd}\}$ are increasing, smooth, strictly concave functions, with $f_{jd}(0) = 0$. As before, the constraints $\mathcal{P}$ simply capture that each resource can be allocated to unit amount, so that $\sum_j x_{jd} \leq 1$ for all $d \in \{1, 2, \ldots, D\}$. The special case as $\rho \to 0$ corresponds to $u_j(\mathbf{x}_j) = \prod_{d=1}^{D} (f_{jd}(x_{jd}))^{\alpha_{jd}}$, where $\sum_d \alpha_{jd} \leq 1$ and $\alpha_{jd} \geq 0$ for all j, d, which can be viewed as *Generalized Cobb-Douglas* utilities. The single-dimensional case ($D = 1$) corresponds to *scheduling with concave speedup curves*, which has been extensively studied in literature [15], [16], [18]. Though we do not present details in this paper, our algorithmic results also extend to a slightly different class of utilities of the form: $u_j(\mathbf{x}_j) = g_j \left(\sum_{d=1}^{D} f_{jd}(x_{jd}) \right)$, where g_j is increasing, smooth, and strictly concave, with $g_j(0) = 0$.

Monotonicity of RA-S.: We prove that RA-S is a special case of MONOTONE PSP in Section II-B. We present the intuition and new technical ideas here. First consider CES utilities given by Eq (1), when $\rho \in [0, 1]$. Recall that for these utilities, the feasibility constraints P simply encode that each resource is allocated to at most the supply of one unit. CES utilities are homogeneous of degree one and quasi-concave. Therefore, the PF algorithm computes the market equilibrium to the equivalent *Fisher market*. Each job j is an agent with budget w_j. Suppose resource d has price p_d. Each agent buys resources to solve the following maximization problem:

$$\text{Maximize} \quad u_j(\mathbf{x}_j) \qquad \text{s.t.} \qquad \sum_{d=1}^{D} p_d x_{jd} \leq w_j$$

A Fisher equilibrium (or market clearing solution) is a set of prices $\{p_d\}$ such that the per-agent utility maximizing allocations clear the market: No resource is over allocated; for each resource with non-zero price, supply equals demand; and each agent spends its entire budget. This follows from the KKT conditions. These utilities satisfy a property termed *Gross Substitutability* (GS) [22]. The GS property means that when the price of a resource increases, the demand for resources whose prices did not increase only goes up. For utilities satisfying GS, it is easy to show that a market clearing solution will be monotone. Since the PF algorithm computes this solution, it satisfies monotonicity (Def. I.1).

For the RA-S utilities (Eq. (2)), the PF algorithm no longer coincides with a Fisher equilibrium. We therefore prove monotonicity of the PF algorithm from first principles. Our proof proceeds by considering $\log u_j(\mathbf{x})$ as a utility function, and viewing the PF algorithm as computing a Walrasian equilibrium [22] of this utility function. The GS property would imply that the equilibrium can be computed by a monotone tatonnement process, and when a new agent arrives, the tatonnement only increases prices, therefore lowering utility. The key technical hurdle in our case is that the utility function $\log u_j(\mathbf{x})$ is not zero when $\mathbf{x} = 0$; in fact it can be unbounded. We therefore need to show a stronger condition than the usual GS property in order to establish monotonicity. The end result is the following corollary to Theorem I.2.

Corollary I.3. *For RA-S utilities, the PF algorithm is $(e+\epsilon)$-speed, $O\left(1/\epsilon^2\right)$ competitive for minimizing weighted flow time, where ϵ is any constant in $(0, 1]$.*

2) Polymatroidal Utilities: This sub-class of PSP is given by the following polyhedron:

$$\mathcal{P} = \left\{ \sum_{j \in S} y_j \leq v(S) \qquad \forall \text{ subsets of jobs } S \right\}$$

where the function $v(S)$ is a non-decreasing submodular function with $v(\phi) = 0$. The feasible region $\mathcal{P}$ is therefore a *polymatroid*. Many natural resource allocation problems define polymatroids:

Single-sink Flow Routing.: We are given a directed capacitated graph $G(V, E)$, with capacities $c(e)$ on edge $e \in E$. Each job j is characterized by a pair of source-sink vertices, (s_j, t_j), as well as a total flow value p_j and weight w_j. If we allocate flow value y_{jt} for job j at time t, then y_{jt} should be a feasible flow from s_j to t_j. The $\{y_{jt}\}$ values should satisfy the capacity constraints on the edges. In the case where all jobs need to route to the same sink node t, the rate region $\mathcal{P}$ is a polymatroid: For a subset of jobs S, let $v(S)$ denote the maximum total rate that can be allocated to jobs in S, then $v(S)$ is a submodular function [35]. A classical result of Kelly *et al.* [30] shows that the TCP congestion control algorithm can be viewed as an implementation of proportional fairness in a distributed fashion. Our result shows that such an implementation is competitive on delays of the flows, assuming they are routed to a single sink.

Video-on-Demand (Multicast).: Consider a video-on-demand setting [10], where different sources of video streams on a network need to stream content to all network vertices via spanning trees. Formally, there is a capacitated undirected graph $G(V, E)$ with a sink node $s \in V$. Job (video stream) j arrives at node v_j. If job j is assigned x_T units of spanning tree T, the rate it gets is x_T; this rate is additive across trees. Any feasible allocation is therefore a fractional assignment of spanning trees to jobs, so that along any edge, the total amount of trees that use that edge is at most the capacity of the edge. This rate polytope $\mathcal{P}$ is a polymatroid [10].

Related Machine Scheduling.: There are M machines, where machine m has speed s_m. The machines are fractionally allocated to jobs; let job j be assigned x_{jm} units of machine m. The feasibility constraints $\mathcal{P}$ require that each machine can be fractionally allocated by at most one unit, so that $\sum_j x_{jm} \leq 1$ for all m; and each job is allocated at most one unit of machines, so that $\sum_m x_{jm} \leq 1$ for all j. The rate of job j is $u_j(\mathbf{x}) = \sum_m s_m x_{jm}$. It is known [17] that the space $\mathcal{P}$ of feasible rates define a polymatroid. While there already exists an $O(1)$-speed $O(1)$-competitive algorithm for this problem, we find this result interesting since the algorithm is very different from [25].

Monotonicity of Polymatroidal Utilities.: Jain and Vazirani [28] generalize Fisher markets to *polymatroidal utilities*, which they term Submodular Utility Allocation (SUA) markets. They show that the PF algorithm computes the market clearing solution. For such markets, they define the notion of *competition monotonicity*: A new agent entering the market leads to greater competition, and hence to lower utilities for existing agents. They show that this market is competition monotone, which directly implies the PF algorithm is monotone for polymatroidal utilities, leading to the following corollary of Theorem I.2.

Corollary I.4. *For polymatroidal utilities, the PF algorithm is $(e + \epsilon)$-speed $O\left(1/\epsilon^2\right)$ competitive for minimizing weighted flow time, where ϵ is any constant in $(0, 1]$.*

C. Queueing View and the PSP-Q Problem

We now take a queueing viewpoint of the PSP problem. One special case of the PSP problem that arises in practice is the case when the jobs can be grouped into a small number of *types* or *queues*. Jobs within a queue have different sizes and weights, but are interchangeable for the purpose of scheduling. Our goal will be to derive an algorithm whose competitive ratio depends on the number of queues instead of the number of jobs. (See Section I-C.) We define PSP-Q as follows: There are K queues. Job j has processing length p_j, weight w_j, and arrives at queue q_j. At each step t, let y_{jt} denote the rate assigned to job j. Let S_{qt} denote the set of jobs in queue q at time t. A feasible allocation at time t is given by the following, where we drop subscript t:

$$\left\{ z_q = \sum_{j \in S_q} y_j \ \forall q; \text{ and } \mathbf{z} \in \mathcal{P}_q \right\}$$

where $\mathcal{P}_q$ is a downward closed convex space. Note jobs in the same queue are completely interchangeable, in that feasibility is determined by the total rate at which jobs are processed in each queue.

Normalized MAX-WEIGHT *Algorithm.:* The MAX-WEIGHT algorithm was first presented in the seminal work of Tassiulas and Ephremides [40]. As with most queueing literature, their work focuses on stability, and assumes jobs are unit-sized and unweighted. To generalize this algorithm for the PSP-Q problem, we need a normalization step that we describe below. While we believe our extension is flexible enough to be combined any single machine scheduling algorithm for each queue, for simplicity, we will only consider two algorithms: Highest Density First (HDF) and an extension of Weighted Round Robin (WRR).

The algorithm MAX-WEIGHT+WRR is defined as follows. Let $g_q = \max\{z_q | \mathbf{z} \in \mathcal{P}_q\}$ be the maximum possible rate that can be assigned to queue q. At time t, let S_{qt} denote the set of jobs in queue q. Let $W_{qt} = \sum_{j \in S_{qt}} w_j$. The MAX-WEIGHT algorithm assigns rates z_{qt} as follows:

$$\max \sum_q W_q \frac{z_{qt}}{g_q} \qquad \text{s.t.} \qquad \mathbf{z}_t \in \mathcal{P}_q$$

Given the rate z_{qt} for queue q, the algorithm sets $y_{jt} = \frac{w_j}{W_q} z_{qt}$ for $j \in S_{qt}$. In contrast with queueing literature, the above algorithm maximizes the weight of the queue times the *normalized rate* $\frac{z_{qt}}{g_q}$.

We show the following theorem in Section III-A. The $(1+\epsilon)$ speed algorithm is obtained by replacing WRR within a queue with a scalable single machine scheduling algorithm; see Section III-A for details.

Theorem I.5. *For the PSP-Q problem with K queues, for any constant $\epsilon > 0$, there is a non-clairvoyant algorithm that is $(1+\epsilon)$-speed $O(K^3/\epsilon^3)$-competitive for weighted flow time. In addition, the* MAX-WEIGHT+WRR *algorithm with speed $(2+\epsilon)$ is $O(K^3/\epsilon^3)$ competitive for PSP-Q.*

We can replace the WRR algorithm for individual queues with the HDF algorithm, thereby obtaining a *clairvoyant* algorithm, MAX-WEIGHT+HDF which is also $(1+\epsilon)$-speed, $O(K/\epsilon^2)$-competitive; see Section III-B for details.

The above results are quite surprising – it is easy to obtain algorithms whose speed depends on K; we instead find an algorithm whose speed is an arbitrarily small constant. At a high level, this is similar to stability analysis in queueing, and indeed, we show the above result by using ideas from that field. A queueing system with an ergodic arrival process is said to be stable if the expected queue sizes are finite. The proof that the MAX-WEIGHT algorithm is stable [40], [34], [33] uses a Lyapunov function argument: The Lyapunov function is the sum of the squares of the queue weights. The rate of change of this function is simply the sum over the queues of the weight of the queue times the rate of change of this weight. Since the arrival process is admissible, it is easy to show that the expected rate of change is larger for the MAX-WEIGHT algorithm than for the arrival process, hence showing that the drift in queue size is always negative.

Though the arrival process is adversarial in our case, we show that Lyapunov functions of a certain form can be converted to a potential function that is defined on the difference in weights between the algorithm's and optimal solution's queues, leading to an amortized local competitiveness analysis [27]. This connection is not straightforward. The key challenge is that while in stability, we need to analyze the change of the Lyapunov function due to arrivals which are fixed and stochastic, in our setting, we have to analyze the change in potential due to the optimal solution's processing. Therefore, though the Lyapunov function method shows stability not just for MAX-WEIGHT, but also for any scheduling policy that maximizes some weighted function of rates (such as proportional fairness, square of weights), this *does not* imply we can convert all these Lyapunov functions into potential functions. In hindsight, our result is the first to use queueing theoretic ideas (stability) to perform adversarial (competitive) analysis for a natural class of scheduling problems.

Since the PSP-Q problem reduces to the PSP problem when each job defines its own queue, the polynomial dependence on K is unavoidable for all non-clairvoyant algorithms if we insist on constant speed [24].

1) Applications of PSP-Q: Though any PSP instance has an equivalent PSP-Q instance, we present some problems that can be naturally modeled by PSP-Q framework.

Flow Routing [30].: Consider the FLOW ROUTING problem from Section I-B2. When the (s_i, t_i) pairs for the jobs can be arbitrary, we can group jobs that require the same source-sink pair into one queue, so that there are at most $K = |V|^2$ queues. This yields a competitive ratio of $O(|V|^6)$ with $(1+\epsilon)$ speed. Note that $|V|$ is the number of vertices in the underlying graph $G(V, E)$, which is fixed and independent of n, the number of flows that arrive and depart over time.

Resource Allocation with Complementarities (RA-C): Consider the *multi-dimensional resource allocation* problem presented in Section I-B1. The class of Leontief utilities is given by $u_j(\mathbf{x}_j) = \min_{d=1}^{D} (c_{jd} x_{jd})$, where $\mathbf{c_j}$ is the *resource vector* of the job. Such utilities capture resources that are *complements* of each other. We term this problem the *Resource Allocation with Complementarities* or RA-C problem. These utilities are widely used to model job rates in data centers [20], [41], [3], [1], [37]. In the RA-C problem, a job is characterized by its resource requirement vector $\mathbf{c_j}$. In practice [33], the number of distinct resource vectors is a small number K, and it can be checked that this is a special case of PSP-Q with K queues. We show in Section III-C that even in the full generality of RA-C, a simple preprocessing shows that $O((\log D)^D)$ queues suffice, albeit with the knowledge of job sizes, which makes the overall algorithm clairvoyant.

D. Summary of Our Results

We summarize our contributions in Table I. We note that even with clairvoyance, better previous results were not known for any of these problems. All these problems require speed augmentation to achieve $O(1)$-competitiveness since even minimizing the weighted flow time on a single machine does [5].

Problem Class	Problem	Best Previous	Our Result	Speed Augmentation
MONOTONE PSP	RA-S	$O(1)$ for $D = 1$ [16]	$O(1)$ for any D	$e + \epsilon$
	Single-source Routing	—	$O(1)$	$e + \epsilon$
	Video-on-demand	—	$O(1)$	$e + \epsilon$
	Related Machines	$O(1)$ [25]	$O(1)$	$e + \epsilon$
PSP-Q	PSP-Q	$O(\log n)$ with $O(\log n)$ speed [24]	$O(K^3)$ Lower Bound: K^α for $\alpha > 0$ [24]	$1 + \epsilon$
	Multicommodity Flow	—	$O(\|V\|^6)$	$1 + \epsilon$

Table I
SUMMARY OF OUR MAIN RESULTS.

E. Outline of Paper

In Section II, we consider the MONOTONE PSP problem. We prove Theorem I.2 in Section II-A and show the monotonicity of RA-S in Section II-B. In Section III, we shift to the PSP-Q problem. We prove Theorem I.5 in Section III-A, and analyze a similar algorithm but using HDF as the single-queue policy in Section III-B. Finally, we present our result on the RA-C problem in Section III-C, and discuss some open questions in Section IV.

II. THE PROPORTIONAL FAIRNESS (PF) ALGORITHM AND MONOTONE PSP

Let A_t denote the set of jobs that are alive at time t; we will often drop t when the time t in consideration is clear from the context. These include jobs j for which $t \in [r_j, C_j]$. The proportional fairness algorithm computes a rate vector $\mathbf{y}_t$ that optimizes:

$$\text{Maximize} \quad \sum_{j \in A_t} w_j \log y_j \qquad \text{s.t.} \qquad \mathbf{y} \in \mathcal{P}$$

Let $y_j^*(S)$ denote the optimal rate the PF algorithm allocates to job $j \in S$ when working on a set of jobs, S. We will use the following well-known proposition repeatedly in our analysis.

Proposition II.1 (Optimality Condition)**.** *Let $\mathbf{y} \in \mathcal{P}$ 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 $f(\mathbf{y}) = \sum_{j \in S} w_j \log y_j$. We have $\frac{\partial f(\mathbf{y}^*)}{\partial y_j} = \frac{w_j}{y_j^*}$. The optimality of $\mathbf{y}^*$ implies $\nabla f(\mathbf{y}^*) \cdot (\mathbf{y} - \mathbf{y}^*) \leq 0$ for all $\mathbf{y} \in \mathcal{P}$. The proposition now follows by elementary algebra. ■

Recall that we analyze the PF algorithm under a natural restriction on the utility functions. Recall from Definition I.1 that the PF algorithm is said to be *monotone* if for any S and $\ell \notin S$, we have the following condition: for all $j \in S$, $y_j^*(S) \geq y_j^*(S \cup \{\ell\})$. We term this class of PSP problems as MONOTONE PSP.

A. Competitive Analysis: Proof of Theorem I.2

We use amortized local competitiveness to show this theorem. The potential function we use is the same as that for one-dimensional concave speedup curves [18]; however, our analysis is different and repeatedly uses Proposition II.1 to bound how the potential function changes when the algorithm processes jobs.

Focus on some time instant t, and define the following quantities. Let A_t denote the subset of jobs alive in the PF schedule, and let O_t denote those alive in OPT's schedule. For job j, let p_{jt} denote the remaining size of the job in the PF's schedule, and let p_{jt}^O 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 $A_t \setminus L_t \subseteq O_t$.

Consider the jobs in increasing order of arrival times, and number them $1, 2, \ldots$ in this order. Let $A_t^{\leq j} = A_t \cap \{1, 2, \ldots, j\}$. Recall that $y^*_j(S)$ denote the optimal rate the PF algorithm allocates to job $j \in S$ when working on a set of jobs, S. We define the following potential function:

$$\Phi(t) = \frac{1}{\epsilon} \sum_{j \in A_t} w_j \frac{\tilde{p}_{jt}}{y^*_j(A_t^{\leq j})}$$

We first show the following simple claim, similar to the one in [18]. This crucially needs the monotonicity of the PF algorithm, and we present the proof for completeness.

Claim II.2. *If $\Phi(t)$ changes discontinuously, this change is negative.*

Proof: If no jobs arrive or is completed by PF or OPT, the $\tilde{p}$ values change continuously, and the $y^*_j(A_t^{\leq j})$ values do not change. Hence, the potential changes continuously. Suppose a job j' arrives; for notational convenience, we assume that the current alive jobs are A_t plus the job j' that just arrived, and $j' \notin A_t$. For this job, $\tilde{p}_{j't} = 0$. Furthermore, this job does not affect $y^*_j(A_t^{\leq j})$ for any $j \in A_t$, since $j' \notin A_t^{\leq j}$. Therefore, the potential does not change when a job arrives. Similarly, suppose a job j' is completed by OPT but A_t remains unchanged. Then, none of the terms in the potential change, and hence $\Phi(t)$ does not change. Finally, consider the case where j' departs from A_t. We have $\tilde{p}_{j't} = 0$. This departure can change $y^*_j(A_t^{\leq j})$ for $j \in A_t$ s.t. $j' \leq j$. By the monotonicity of the PF algorithm, these rates cannot decrease. Therefore, all terms in the potential are weakly decreasing, completing the proof. ∎

Assuming that PF uses a speed of $(e + \epsilon)$ compared to OPT, we will show the following at each time instant t where no job arrives or is completed either by PF or OPT. Here, $W(S) = \sum_{j \in S} w_j$.

$$W(A_t) + \frac{d}{dt}\Phi(t) \leq \frac{2}{\epsilon^2} W(O_t) \tag{3}$$

Suppose all jobs are completed by PF and OPT by time T. Then integrating the above inequality over time, and using the fact that the discontinuous changes to Φ are all negative, we have:

$$\int_{t=0}^{T} W(A_t)dt + (\Phi(T) - \Phi(0)) \leq \frac{2}{\epsilon^2} \int_{t=0}^{T} W(O_t)dt$$

Note that the first term above is the weighted flow time of PF, the second term is zero, and the RHS is the weighted flow time of OPT. This will complete the proof of Theorem I.2.

1) Proving Inequality (3): Consider a time instant t when no job arrives or completes. To simplify notation, we omit the subscript t from the proof. Let $\frac{d}{dt}\Phi|_O$ and $\frac{d}{dt}\Phi|_A$ denote the potential changes due to OPT's processing and PF's processing respectively. Note that $\frac{d}{dt}\Phi = \frac{d}{dt}\Phi|_A + \frac{d}{dt}\Phi|_O$.

Lemma II.3. $\frac{d}{dt}\Phi|_O \leq \frac{1}{\epsilon} W(A)$.

Proof: For job $j \in A$, suppose OPT assigns rate y^O_j. Then, $\frac{d}{dt}\tilde{p}_j \leq y^O_j$ for $j \in A$, due to OPT's processing. Therefore, the change in potential is upper bounded by:

$$\frac{d}{dt}\Phi|_O \leq \frac{1}{\epsilon} \sum_{j \in A} w_j \frac{y^O_j}{y^*_j(A_t^{\leq j})} \leq \frac{1}{\epsilon} \sum_{j \in A} w_j \frac{y^O_j}{y^*_j(A)}$$

The inequality above follows from the monotonicity of the PF algorithm, since $A^{\leq j} \subseteq A$. Using Proposition II.1, the RHS is at most $W(A)$. This completes the proof. ∎

We now bound $\frac{d}{dt}\Phi|_A$, the change in potential due to PF. We first assume PF runs at speed 1, and we will scale this up later. We consider two cases:

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

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

where the second inequality follows from Lemma II.3.

Case 2.: The more interesting case is when $W(L) \geq (1-\epsilon)W(A)$. For $j \in L$, we have $\frac{d}{dt}\tilde{p}_j = y_j^*(A)$ due to PF's processing, by the definition of $y_j^*(A)$. Therefore,

$$\epsilon \cdot \frac{d}{dt}\Phi|_A \leq -\sum_{j \in L} w_j \frac{y_j^*(A)}{y_j^*(A^{\leq j})}$$

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

We now apply Proposition II.1 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 $y_j^*(A^{\leq k})$, and this goes into the denominator in Proposition II.1. We consider $y_j^*(A^{\leq k-1})$ for $j < k$, and $y_k^*(A^{\leq k-1}) = 0$ as a different set of rates that go into the numerator in Proposition II.1. This yields:

$$\sum_{j=1}^{k-1} w_j \frac{y_j^*(A^{\leq k-1})}{y_j^*(A^{\leq k})} \leq \sum_{j=1}^{k} w_j$$

Observing that $\frac{y_j^*(A^{\leq k-1})}{y_j^*(A^{\leq k})} = 1 + \delta_{jk}$, we obtain $\sum_{j=1}^{k-1} w_j \delta_{jk} \leq w_k$ for $k = 1, 2, \ldots, \kappa$. Adding these inequalities for $k = 1, 2, \ldots, \kappa$ and changing the order of summations, we obtain:

$$\sum_{k=1}^{\kappa} \sum_{j=1}^{k-1} w_j \delta_{jk} = \sum_{j=1}^{\kappa} w_j \left(\sum_{k=j+1}^{\kappa} \delta_{jk} \right) \leq W(A) \qquad \Longrightarrow \qquad \sum_{j \in L} w_j \left(\sum_{k=j+1}^{\kappa} \delta_{jk} \right) \leq W(A)$$

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

$$\frac{y_j^*(A)}{y_j^*(A^{\leq j})} = \prod_{k=j+1}^{\kappa} \frac{1}{\alpha_{jk}} = \prod_{k=j+1}^{\kappa} \frac{1}{1+\delta_{jk}} \geq \exp\left(-\sum_{k=j+1}^{\kappa} \delta_{jk} \right) = \exp(-\Delta_j)$$

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

$$\epsilon \cdot \frac{d}{dt}\Phi|_A \leq -\sum_{j \in L} w_j \frac{y_j^*(A)}{y_j^*(A^{\leq j})} \leq -\sum_{j \in L} w_j \exp(-\Delta_j)$$

Since $\sum_{j \in L} w_j \Delta_j \leq W(A)$, the RHS is maximized when $\Delta_j = W(A)/W(L) \leq 1/(1-\epsilon)$. This implies:

$$\epsilon \cdot \frac{d}{dt}\Phi|_A \leq -\sum_{j \in L} w_j \exp(-W(A)/W(L)) \leq -W(L)\exp(-1/(1-\epsilon)) \leq -\frac{1-2\epsilon}{e} W(A)$$

for $0 < \epsilon < 1/2$. Therefore, if we run PF at speed $(e + 3\epsilon)$, we have: $\frac{d}{dt}\Phi|_A \leq -\left(1 + \frac{1}{\epsilon}\right) W(A)$. Therefore,

$$W(A) + \frac{d}{dt}\Phi|_O + \frac{d}{dt}\Phi|_A \leq W(A) + \frac{1}{\epsilon} W(A) - \left(1 + \frac{1}{\epsilon}\right) W(A) \leq 0 \leq W(O)$$

This completes the proof of Inequality (3) and hence of Theorem I.2.

B. Monotonicity of RA-S: Proof of Corollary I.3

We will show the following theorem, which when combined with Theorem I.2 shows Corollary I.3.

Theorem II.4. *The RA-S utility functions defined in Eq (2) are concave (which implies the space $\mathcal{P}$ is convex). Furthermore, the PF algorithm is monotone for these functions.*

The first part follows by easy algebra. The CES utility function given by Eq (1), when $\rho \in (0, 1]$, is homogeneous of degree one and quasi-concave. This implies it is concave [9]. The RA-S utilities are obtained by a monotone concave transformation of the variables and the entire function. This preserves concavity. This implies the space $\mathcal{P}$ of feasible utilities is convex.

The remainder of this section is devoted to proving the second part of the theorem. Recall that for RA-S, the space $\mathcal{P}$ is given by the following (where $\rho \in [0,1]$ and $\rho' \geq \rho$):

$$\mathcal{P} = \left\{ y_j = u_j(\mathbf{x}_j) = \left(\sum_{d=1}^{D} (f_{jd}(x_{jd}))^{\rho_j} \right)^{1/\rho'_j}, \qquad \sum_j x_{jd} \leq 1 \ \forall d \right\}$$

Let $h_{jd}(x_{jd}) = (f_{jd}(x_{jd}))^{\rho_j}$. This function is increasing and strictly concave, assuming the same is true for f_{jd}. Further $h_{jd}(0) = 0$. Define:

$$v_j(\mathbf{x}_j) = w_j \log u_j(\mathbf{x}_j) = \frac{w_j}{\rho'_j} \log \left(\sum_{d=1}^{D} h_{jd}(x_{jd}) \right)$$

For price vector $\mathbf{p} = \{p_1, p_2, \ldots, p_D\} \geq \mathbf{0}$, define the demand function $X_j(\mathbf{p})$ as follows:

$$X_j(\mathbf{p}) = \text{argmax}_{\mathbf{x}_j \geq 0} \left(v_j(\mathbf{x}) - \mathbf{p} \cdot \mathbf{x}_j \right) \qquad \text{and} \qquad U_j(\mathbf{p}) = u_j(X_j(\mathbf{p}))$$

Note that $X_j(\mathbf{p})$ is uniquely defined for given $\mathbf{p}$ due to the strict concavity of $v_j(\mathbf{x})$ (see the first part of Theorem II.4), so $U_j(\mathbf{p})$ is well-defined.

Lemma II.5. *Consider an arbitrary price vector $\mathbf{p}$, and a different price vector $\mathbf{p}'$ that only differs from $\mathbf{p}$ in the r^{th} dimension. Assume $p'_r > p_r$. Let $\mathbf{x}_j = X_j(\mathbf{p})$ and $\mathbf{x}'_j = X_j(\mathbf{p}')$. Then:*

1) *If $x_{jr} > 0$, then $U_j(\mathbf{p}') < U_j(\mathbf{p})$. Furthermore, $x'_{jr} < x_{jr}$, and for all $d \neq r$, $x'_{jd} \geq x_{jd}$.*
2) *If $x_{jr} = 0$, then $U_j(\mathbf{p}') = U_j(\mathbf{p})$. Furthermore, for all d, $x'_{jd} = x_{jd}$.*

Further, we have a stronger property that if $x_{jr} > c$, then $U_j(\mathbf{p}) - U_j(\mathbf{p}') \geq c'(p_r - p'_r)$ for a finite $c' > 0$ when the following conditions are satisfied:

- *For all j, d, h_{jd} has a bounded curvature over the domain $[c, C]$ for finite values $c, C > 0$, i.e. there exist $\gamma = \gamma(c, C)$ and $\delta = \delta(c, C)$ such that $\gamma(y_2 - y_1) \leq h'_{jd}(y_1) - h'_{jd}(y_2) \leq \delta(y_2 - y_1)$ for all $c \leq y_1 \leq y_2 \leq C$.*
- *Both vectors $\mathbf{p}$ and $\mathbf{p}'$ are upper bounded by a finite vector.*

Proof: Since we focus on a single job j, we omit the subscript j in the proof. Focus on dimension r. Let $q = w_j/\rho'_j$. Let $W(\mathbf{x}) = (U_j(\mathbf{x}))^{\rho'_j}$; since $U_j(\mathbf{x})$ is strictly monotone in $\mathbf{x}$, the same holds for $W(\mathbf{x})$. Note that $W(\cdot)$ also can be viewed as a function of $\mathbf{p}$ since $\mathbf{p}$ uniquely determines $\mathbf{x}$. Partially differentiating $v_j(\mathbf{x}) - \mathbf{p} \cdot \mathbf{x}_j$ w.r.t. x_d, we have the following (sufficient and necessary) optimality condition:

$$x_d > 0 \quad \Rightarrow \quad \frac{q}{W(\mathbf{x})} h'_d(x_d) = p_d \qquad \text{and} \qquad x_d = 0 \quad \Rightarrow \quad \frac{q}{W(\mathbf{x})} h'_d(x_d) \leq p_d$$

Consider price vector $\mathbf{p}'$ with $p'_r > p_r$ and $p'_d = p_d$ for all $d \neq r$. If $x_r = 0$, this does not change the optimality condition above for any dimension d, so that the second part of the lemma follows.

If $x_r > 0$, suppose $W(\mathbf{p}') \geq W(\mathbf{p})$. This implies $h'(x_r) < h'(x'_r)$ since $p'_r > p_r$. To satisfy the optimality condition, we must therefore have $x'_r < x_r$ by the strict concavity of h_r. The same argument shows that for all dimensions $d \neq r$, $x'_d \leq x_d$. But this implies $W(\mathbf{p}') < W(\mathbf{p})$, which is a contradiction. Therefore, we must have $W(\mathbf{p}') < W(\mathbf{p})$.

Now consider any dimension $d \neq r$. Since $p'_d = p_d$ and $W(\mathbf{p}') < W(\mathbf{p})$, the optimality condition implies that $h'_d(x'_d) \leq h'_d(x_d)$. This implies $x'_d \geq x_d$. However, since the utility strictly decreased, this must imply $x'_r < x_r$.

It now remains to show the stronger property under the extra conditions. Imagine that we increase $\mathbf{p}$ to $\mathbf{p}'$ continutously by slowly increasing p_r to p'_r. For simplicity, we assume that for all d, p_d remains either non-zero or zero throughout this process, excluding the start and the end – the general case can be shown by starting a new process when p_d's status, whether it is non-zero or zero, changes. Since if p_d remains 0 in the process, d has no effect on $W(\mathbf{x})$, let's focus on d with non-zero p_d.

Observe that boundedness of $\mathbf{p}$ implies boundedness of $\mathbf{x}$ since $\mathbf{x}$ minimizes $v_j(\mathbf{x}) - \mathbf{p} \cdot \mathbf{x}_j$ and $v_j(\mathbf{x})$ is strictly concave. Since the remaining proof follows from a tedious basic algebra, we only give a sketch here. In the following we crucially use the boundedness of $\mathbf{p}, \mathbf{p}', \mathbf{x}, \mathbf{x}'$. For the sake of contradiction, suppose $W_j(\mathbf{p})$ and $W_j(\mathbf{p}')$ are very close such that the claim is not true for any fixed c'. Then, for all $d \neq r$ with non-zero p_d, the bounded curvature of h_d and the optimality condition imply that x_d and x'_d are very close. Likewise, we can argue that x_r and x'_r are significantly different so that the differnece is lower bounded by $c''(p'_r - p_r)$ for a fixed c''. This leads to the conclusion that $W_j(\mathbf{p}')$ and $W_j(\mathbf{p})$ are significantly different, which is a contradiction. An easy algebra gives the desired claim. ∎

Proof of Theorem II.4.: We now use Lemma II.5 to show the second part of the theorem. The KKT conditions applied to the PF convex program imply the following:

1) There exists a price vector $\mathbf{p}$ such that $\{X_j(\mathbf{p})\}$ define the optimal solution to PF.
2) For this price vector $\mathbf{p}$, if $p_d > 0$, then $\sum_j x_{jd} = 1$.

Start with this optimal solution. Suppose a new job arrives. At the price vector $\mathbf{p}$, compute the quantities $X_j(\mathbf{p})$. If some resource is over-demanded, we continuously increase its price. We perform this tatonnement process until no resource is over-demanded. By Lemma II.5, any job that demands a resource whose price is increasing, sees its overall utility strictly decrease, while jobs that do not demand this resource see their utility remain unchanged. Therefore, if we define the potential function to be the total utility of the jobs, this potential strictly decreases. Further, by Lemma II.5, the total demand for the resource whose price is increasing strictly decreases, while the demands for all other resources weakly increase. Therefore, any resource with price strictly positive must have total demand at least one at all points of time.

Now parameterize the tatonnement process by the total price of resources. When the price of over-allocated resource r is raised, there must exist a job j such that $x_{jr} \geq 1/n$. This, when combined with the optimal condition, implies that p_r is bounded. Since we only increae the price of over-demanded resources, the boundness of $\mathbf{p}$ follows. Hence by the stronger property of Lemma II.5, the potential must decrease by at least c' times the increase of the total price for some finite $c' > 0$; the potential decreases at least as much as j's utility does. This implies that the process must terminate since the potential is lower bounded by zero. When it terminates, suppose the price vector is $\mathbf{p}'$, and let $\mathbf{x}'_j = X_j(\mathbf{p}')$. Any resource d with $p'_d > 0$ must have $\sum_j x'_{jd} = 1$. If $p_d = 0$, we must have $\sum_j x'_{jd} \leq 1$. This therefore is the new optimal solution to the PF program. Since the utilities of all existing jobs either stay the same or decrease in the tatonnement process, this shows the PF algorithm is monotone. This completes the proof of Theorem II.4.

A similar proof to the above shows the following; we omit the details.

Corollary II.6. *The PF algorithm is monotone for utility functions of the form* $u_j(\mathbf{x}_j) = g_j\left(\sum_{d=1}^{D} f_{jd}(x_{jd})\right)$, *where* g_j, f_{jd} *are increasing, smooth, and strictly concave functions.*

III. Algorithms for PSP-Q

In this section we develop two algorithms, one non-clairvoyant and one clairvoyant, for the PSP-Q problem. The two algorithms are derived by combining Normalized Max-Weight algorithm with the following two widely used scheduling algorithms, respectively: 1) (an extension of) Weighted Round Robin (WRR) and 2) Highest Density First (HDF).

A. Non-Clairvoyant Algorithm: Normalized Max-Weight + Weighted Round Robin

This section is devoted to proving Theorem I.5. We show that the algorithm Normalized Max-Weight gives us the desired result when combined with Weighted Latest Arrival Processor Sharing (WLAPS), which is a scalable generalization of WRR [16]. We begin by describing the algorithm.

1) Algorithm Description: Recall that z_{qt} denotes the rate assigned to q at time t, and g_q is the maximum rate that can be assigned to queue q ($g_q = \max\{z_q \mid \mathbf{z} \in \mathcal{P}_q\}$). For the sake of analysis we scale the input instance such that $g_q = 1$ for all queues $q \in [K]$. This can be done by scaling the polytope such that $g_q = 1$ for all queues and modifying the size of each job j in q to p_j/g_q. It is easy to verify that an optimal solution remains unchanged due to this scaling. We emphasize that this scaling and modification is done only for the sake of analysis, and our algorithm does not have to know the job sizes – this conversion can be thought as being done at the end of algorithm's run. Indeed, our algorithm will only need to know the set of alive jobs in each queue along with their weights, but not their remaining sizes. Hence our algorithm will remain non-clairvoyant. For the remainder of this section we assume that $g_q = 1$ for all $q \in [K]$.

The Normalized Max-Weight algorithm calculates a feasible set of rates z_{qt} for each queue at each time instant t by solving the following convex program. Recall that W_{qt} is the total weight of jobs alive in the algorithm's queue q.

$$\max \sum_{q \in [K]} W_{qt} \cdot z_{qt} \quad \text{s.t} \quad \mathbf{z_t} \in \mathcal{P}_q$$

The total rate allocated to a queue q is distributed among the jobs in the queue q using the WLAPS policy. Let A_{qt} denote the set of jobs that are in the algorithm's queue q at time t. Let A^{ϵ}_{qt} denote the *minimal set of latest*

arriving jobs in A_{qt} such that their total weight is at least $\epsilon \cdot W_{qt}$, for some parameter $\epsilon \in [0,1]$. Our algorithm distributes the total rate z_{qt} among the jobs in the set A^{ϵ}_{qt}.

$$y_{jt} = z_{qt} \cdot \frac{w_j}{\epsilon \cdot W_{qt}} \quad \forall j \in A^{\epsilon}_{qt},$$

with the exception that the rate assigned to the earliest arriving job j' in A^{ϵ}_{qt} is

$$y_{j't} = z_{qt} \cdot \frac{\epsilon \cdot W_{qt} - \sum_{i \in A^{\epsilon}_{qt} \setminus \{j'\}} w_i}{\epsilon \cdot W_{qt}}$$

In other words, job j' gets the remaining rate from z_{qt} after assigning rates to other jobs in A^{ϵ}_{qt}. Observe that if $\epsilon = 1$, then the algorithm distributes total available rate z_{qt} using WRR. This completes the description of our algorithm. To make our analysis more transparent, we will assume that $\sum_{i \in A^{\epsilon}_{qt}} w_i = \epsilon W_{qt}$ which allows us to ignore the exception. This simplifying assumption can be easily removed.

We observe two simple properties. Let $W_{\max,t}$ denote the highest weight over all queues; that is, $W_{\max,t} = \max_{q \in [K]} W_{qt}$.

Observation 1. $\sum_{q \in [K]} W_{qt} \cdot z_{qt} \geq W_{\max,t}$

Observation 2. $\sum_{q \in [K]} W_{qt} \cdot z_{qt} \geq \frac{1}{K} \cdot \sum_q W_{qt}$

The above observations follow by considering two feasible schedules of assigning a rate of 1 to the highest weight queue, and assigning a rate of $\frac{1}{K}$ to each queue.

Since our algorithm computes an optimal feasible solution to $\max_{\mathbf{z}'} \sum_q W_{qt} \cdot z'_{qt}$, we immediately have the following optimality condition. Recall that z_{qt} is the rate our algorithm assigns to queue q at time t.

Proposition III.1 (Optimality Condition)**.** *For any* $\mathbf{z}'_t \in \mathcal{P}_q, \sum_{q \in [K]} W_{qt} \cdot z_{qt} \geq \sum_{q \in [K]} W_{qt} \cdot z'_{qt}$

Next, we will show that this algorithm, when given a speed of $(1+\epsilon)$, is $O(\frac{K^3}{\epsilon^3})$-competitive for the PSP-Q problem.

2) Analysis: Proof of Theorem I.5: Our analysis is based on a potential function argument. Towards defining a potential function we will set up some notation. Define W^O_{qt} as the total weight of jobs that are alive in the queue q in a fixed optimal schedule OPT. Let p_{jt}, p^O_{jt} denote the job j's remaining size in our algorithm's schedule and the optimal schedule at time t, respectively. Define a job j's *lag* as $\tilde{p}_{jt} := \max(p_{jt} - p^O_{jt}, 0)$. The quantity $\tilde{p}_{jt}$ indicates how much our algorithm is behind the optimal schedule in terms of job j's processing. Note that jobs always have non-negative lags. Furthermore, if a job is alive in the algorithm's schedule and has zero lag then it implies that the job is also alive in the optimal schedule.

It is assumed w.l.o.g. that no two jobs arrive at the same time. For a job $j \in A_{qt}$, define $A_{qt,\leq j}$ as the set of jobs in A_{qt} that arrive no later than job j. That is, $A_{qt,\leq j} = \{j' \mid j' \in A_{qt} \text{ and } r'_j \leq r_j\}$. Let $W_{qt,\leq j}$ denote the total weight of jobs in $A_{qt,\leq j}$. In other words, $W_{qt,\leq j}$ denotes the total weight of jobs in our algorithm's queue q at time t that arrive no later than job j.

We now define the potential function:

$$\Phi(t) = \frac{K}{\epsilon} \cdot \sum_{q \in [K]} \sum_{j \in A_{qt}} W_{t,\leq j} \cdot \tilde{p}_{jt} \tag{4}$$

Since jobs have non-negative lags, $\Phi(t) \geq 0$ at all time instants t. Let T be a sufficiently large time when all jobs are completed by our algorithm and OPT. Clearly, $\Phi(0) = \Phi(T) = 0$.

We first consider non-continuous changes of the potential.

Lemma III.2. *When a job arrives or is completed by our algorithm or OPT, the potential does not increase.*

Proof: First, observe that arrival of a new job j does not change the potential as $\tilde{p}_{jt} = 0$ for the job j, and for all other jobs $j' \in A_{qt}$ the quantity $W_{t,\leq j'}$ remains the same since $r_{j'} < r_j$. OPT completing a job has no effect on the potential. Finally, when our algorithm completes a job j, the potential can only decrease as $W_{t,\leq j'}$ decreases for each $r_{j'} > r_j$ and remains the same for each $r_{j'} < r_j$. ∎

Therefore, we have $\int_{t=0}^{T} \frac{d}{dt}\Phi(t)dt \geq 0$. Our main goal is to show the following lemma that involves continuous changes of the potential.

Lemma III.3. *For all time instants t in the execution of our algorithm where no jobs arrive or are completed by our algorithm or OPT, $\sum_{q\in[K]} W_{qt} + \frac{d}{dt}\Phi(t) \le O(\frac{K^3}{\epsilon^3}) \cdot \sum_{q\in[K]} W^O_{qt}$.*

Indeed, integrating the inequality in the lemma over the time period $[0, T]$ yields Theorem I.5 since $\int_{t=0}^{T} \sum_q W_{qt}\, dt$, and $\int_{t=0}^{T} \sum_q W^O_{qt}\, dt$ are our algorithm's total weighted flow time and OPT's, respectively.

It now remains to prove Lemma III.3. Consider a fixed time instant t where no discontinuous changes occur. Let $\tilde{A}^\epsilon_{qt}$ denote the set of jobs in A^ϵ_{qt} that have a positive lag; recall that A^ϵ_{qt} denotes the set of latest arriving jobs whose total weight is $\epsilon \cdot W_{qt}$. We consider two cases depending on the total weight of jobs in the set $\tilde{A}^\epsilon_{qt}$.

Case 1:: For all queues $q \in [K], \sum_{j\in\tilde{A}^\epsilon_{qt}} w_j \ge \epsilon \cdot W_{qt} - \frac{\epsilon^2}{K} \cdot W_{\max,t}$.

Roughly speaking, in this scenario, our algorithm is behind OPT for almost all jobs in every queue q. We first consider the decrease of the potential due to our algorithm's processing.

$$\begin{aligned}
\frac{d}{dt}\Phi(t)|_A &= \frac{K}{\epsilon} \cdot \sum_{q\in[K]} \sum_{j\in A_{qt}} W_{t,\le j} \cdot \frac{d}{dt}\tilde{p}_{jt}|_A \\
&= -\frac{K}{\epsilon} \cdot \sum_{q\in[K]} \sum_{j\in\tilde{A}^\epsilon_{qt}} W_{t,\le j} \cdot (1+5\epsilon) \cdot z_{qt} \cdot \frac{w_j}{\epsilon \cdot W_{qt}} && \text{[def. of WLAPS and speed augmentation]} \\
&\le -\frac{K}{\epsilon} \cdot \sum_{q\in[K]} \sum_{j\in\tilde{A}^\epsilon_{qt}} (1-\epsilon) \cdot W_{qt} \cdot (1+5\epsilon) \cdot z_{qt} \cdot \frac{w_j}{\epsilon \cdot W_{qt}} && [W_{t,\le j} \ge (1-\epsilon)W_{qt} \text{ for all } j \in \tilde{A}^\epsilon_{qt}] \\
&\le -\frac{K}{\epsilon^2} \cdot (1+3\epsilon) \cdot \sum_{q\in[K]} z_{qt} \cdot \sum_{j\in\tilde{A}^\epsilon_{qt}} w_j
\end{aligned}$$

We now use the condition of Case (1), as well as Observation 1 to complete the proof. Note that this part crucially requires the normalization step.

$$\begin{aligned}
\frac{d}{dt}\Phi(t)|_A &\le -\frac{K}{\epsilon^2} \cdot (1+3\epsilon) \cdot \sum_{q\in[K]} z_{qt} \cdot (\epsilon \cdot W_{qt} - \frac{\epsilon^2}{K} \cdot W_{\max,t}) && \text{[Condition of Case 1]} \\
&\le -\frac{K}{\epsilon} \cdot (1+3\epsilon) \cdot \left(\Big(\sum_{q\in[K]} z_{qt} \cdot W_{qt} \Big) - \epsilon \cdot W_{\max,t} \right) && [z_{qt} \le 1 \text{ for all } q \in [K]] \\
&\le -\frac{K}{\epsilon} \cdot (1+3\epsilon) \cdot \left(\sum_{q\in[K]} z_{qt} \cdot W_{qt} - \epsilon \cdot \sum_{q\in[K]} z_{qt} \cdot W_{qt} \right) && \text{[Observation 1]} \\
&\le -\frac{K}{\epsilon} \cdot (1+\epsilon) \cdot \sum_{q\in[K]} z_{qt} \cdot W_{qt} && (5)
\end{aligned}$$

Next, consider the increase of the potential due to OPT's processing. Let z^O_{qt} denote the rate at which OPT processes q. We note that the last inequality is where the optimality condition of Proposition III.1 plays a crucial role.

$$\begin{aligned}
\frac{d}{dt}\Phi(t)|_O &= \frac{K}{\epsilon} \cdot \sum_{q\in[K]} \sum_{j\in A_{qt}} W_{t,\le j} \cdot \frac{d}{dt}\tilde{p}_{jt}|_O \\
&\le \frac{K}{\epsilon} \cdot \sum_{q\in[K]} \sum_{j\in A_{qt}} W_{qt} \cdot \frac{d}{dt}\tilde{p}_{jt}|_O && [W_{qt} \ge W_{t,\le j} \text{ for all } j] \\
&\le \frac{K}{\epsilon} \cdot \sum_{q\in[K]} W_{qt} \cdot z^O_{qt} \;\le\; \frac{K}{\epsilon} \cdot \sum_{q\in[K]} W_{qt} \cdot z_{qt} && \text{[Proposition III.1]} \quad (6)
\end{aligned}$$

Therefore, from the equations (5) and (6), the total decrease in the potential is at least

$$\begin{aligned}\frac{d}{dt}\Phi(t) = \frac{d}{dt}\Phi(t)|_O + \frac{d}{dt}\Phi(t)|_A &\le -\frac{K}{\epsilon}\cdot(1+\epsilon)\cdot\sum_{q\in[K]} z_{qt}\cdot W_{qt} + \frac{K}{\epsilon}\cdot\sum_{q\in[K]} z_{qt}\cdot W_{qt}\\ &\le -K\cdot\sum_{q\in[K]} z_{qt}\cdot W_{qt} \le -K\cdot\sum_{q\in[K]}\frac{1}{K}\cdot W_{qt} \qquad \text{[Observation 2]}\\ &\le -\sum_{q\in[K]} W_{qt}\end{aligned}$$

Therefore, we have $\sum_{q\in[K]} W_{qt} + \frac{d}{dt}\Phi(t) \le 0$, proving Lemma III.3 for the first case.

Case 2:: For some $q \in [K], \sum_{j\in\tilde{A}^{\epsilon}_{qt}} w_j \le \epsilon\cdot W_{qt} - \frac{\epsilon^2}{K}\cdot W_{\max,t}$

In this scenario, the total weight of jobs in OPT's queue is comparable to that of our algorithm. So, we charge the cost incurred by our algorithm to OPT directly. For this case, we ignore the change of the potential due to our algorithm's processing as it is always at most 0. Let q' be a queue for which the above inequality in the condition holds. Rearranging terms, we have

$$\frac{\epsilon^2}{K}\cdot W_{\max,t} \le \epsilon\cdot W_{q't} - \sum_{j\in\tilde{A}^{\epsilon}_{q't}} w_j = \sum_{j\in A^{\epsilon}_{q't}} w_j - \sum_{j\in\tilde{A}^{\epsilon}_{q't}} w_j \le W^O_{q't},$$

since all jobs in $A^{\epsilon}_{q't}\setminus\tilde{A}^{\epsilon}_{q't}$ are alive in the optimal schedule's queue q; note that $j\in A^{\epsilon}_{q't}\setminus\tilde{A}^{\epsilon}_{q't}$ implies the algorithm is ahead of OPT in terms of job j's processing but still hasn't completed the job. With this observation in mind, we derive

$$\begin{aligned}&\sum_{q\in[K]} W_{qt} + \frac{d}{dt}\Phi(t)\\ &\le \sum_{q\in[K]} W_{qt} + \frac{d}{dt}\Phi(t)|_O \le \sum_{q\in[K]} W_{qt} + \frac{K}{\epsilon}\cdot\sum_{q\in[K]} W_{qt}\cdot z_{qt} && \text{[Equation (6)]}\\ &\le \sum_{q\in[K]} W_{qt} + \frac{K}{\epsilon}\cdot\sum_{q\in[K]} W_{qt} && [z_{qt}\le 1 \text{ for all } q\in[K]]\\ &\le \frac{K+1}{\epsilon}\cdot K\cdot W_{\max,t} && [W_{qt}\le W_{\max,t} \text{ for all } q\in[K]]\\ &= O(\frac{K^3}{\epsilon^3})\cdot\sum_{q\in[K]} W^O_{qt}\end{aligned}$$

This completes the proof of Lemma III.3, and gives us Theorem I.5.

B. Clairvoyant Algorithm: Normalized Max-Weight + Highest Density First

In this section, we consider the algorithm Normalized Max-Weight combined with Highest Density First (HDF). This algorithm assigns rates z_{qt} to queues giving each queue q a weight that is equal to the total fractinal weight of jobs in the queue q while fully devoting the rate z_{qt} to the highest density job in the queue. Our main goal is to show the following theorem.

Theorem III.4. *For the PSP-Q problem with K queues, for any constant $\epsilon > 0$, the algorithm* Normalized Max-Weight+Highest Density First *with speed $(1+\epsilon)$ is $O(K/\epsilon)$ competitive for fractional weighted flow time. Therefore, there is an algorithm that is $(1+\epsilon)$ is $O(K/\epsilon^2)$ competitive for (integeral) weighted flow time.*

Fractional weighted flow time is a relaxation of its integral counterpart, letting a job j only incur cost $w_j p_{jt}/p_j$ at time instant t as opposed to w_j incurred in the integral objective while j is alive. Hence the fractional weighted flow lower bounds the integral weighted flow. Although the integral weighted flow can be significantly greater than the fractional weighted flow, it is known that an algorithm that is c-competitive for the fractional objective can be converted online into an algorithm that is $O(c/\epsilon)$-competitive with an extra speed augmentation of $(1+\epsilon)$. For example, see [27]. Henceforth we will focus on proving the first part of Theorem III.4.

1) Algorithm: For simplicity, assume w.l.o.g. that all jobs have size 1. This can be done following the standard conversion from arbitrary jobs sizes to unit jobs sizes: each job j is replaced with p_j jobs with weight w_j/p_j. It is well known that the fractional objective remains the same after this conversion. Then, in this simplified instance, processing the highest density job implies processing the highest weight job. For notational convenience, we will assume that in the given instance, jobs have unit sizes, i.e. $p_j = 1$ for all j from the beginning. We assume w.l.o.g. that all jobs have distinct weights. Also as in the previous section, we assume w.l.o.g. that $g_q = 1$ for all $q \in [K]$.

Let J_{qt} denote the jobs that have arrived into queue q by time t, and p_{jt} job j's remaining size at time t in the algorithm's schedule. The NORMALIZED MAX-WEIGHT algorithm calculates a feasible set of rates z_{qt} for each queue at each time instant t by solving the following mathematical program.

$$\max \sum_{q\in[K]} \left(\sum_{j\in J_{qt}} w_j p_{jt} \right) \cdot z_{qt} \quad \text{s.t} \quad \mathbf{z_t} \in \mathcal{P}_q$$

We reserve z_{qt} throughout this section to denote the rate our algorithm assigns to queue q at time t. Note that $\sum_{j\in J_{qt}} w_j p_{jt}$ is the total fractional weight of jobs alive in the algorithm's queue q since if $j \in J_{qt} \setminus A_{qt}$, then $p_{jt} = 0$, so $\sum_{j\in J_{qt}} w_j p_{jt} = \sum_{j\in A_{qt}} w_j p_{jt}$. Then, the algorithm processes the highest weight job 1_{qt} that is alive in each queue q at a rate of z_{qt}, i.e. $\frac{d}{dt} p_{1_{qt}t} = -z_{qt}$.

Similar to the analysis in the previous section, we make the following observations. Let $\hat{W}_{qt} = \sum_{j\in J_{qt}} w_j p_{jt}$ and $\hat{W}^O_{qt} = \sum_{j\in J_{qt}} w_j p^O_{jt}$.

Observation 3. $\sum_{q\in[K]} \hat{W}_{qt} \cdot z_{qt} \geq \frac{1}{K} \cdot \sum_{q\in[K]} \hat{W}_{qt}$

Proposition III.5. *For any* $\mathbf{z}'_t \in \mathcal{P}_q, \sum_{q\in[K]} \hat{W}_{qt} \cdot z_{qt} \geq \sum_{q\in[K]} \hat{W}_{qt} \cdot z'_{qt}$

2) Analysis: Our analysis will be based on a potential function argument which is inspired by [12]. The potential is defined as,

$$\Phi(t) := \sum_{q\in[K]} \tilde{C}_q(t)$$

where

$$\tilde{C}_q(t) := \sum_{j\in J_{qt}} w_j p_{jt} \left(\sum_{j'\in J_{qt}, w_{j'} \geq w_j} p_{j't} - \sum_{j'\in J_{qt}, w_{j'} \geq w_j} p^O_{j't} \right) - \sum_{j\in J_{qt}} w_j p^O_{jt} \left(\sum_{j'\in J_{qt}, w_{j'} \geq w_j} p_{j't} \right)$$

We will proceed our analysis by taking a close look at how the potential changes over time. We first consider *discontinuous* changes. Say a new job i arrives into queue q. So far jobs J_{qt} and i have arrived into queue q. For each $j \in J_{qt}$, the quantity in the first summation does not change since $p_{it} = p^O_{it}$ and either $p_{it} - p^O_{it}$ or nothing is added to the quantity in the first parenthesis. Also job i appear in the first summation which can increase the potential by at most $w_i p_{it} \sum_{j'\in J_{qt}, w_{j'} \geq w_i} p_{j't}$. However, it is offset by $w_i p^O_{it} \sum_{j'\in J_{qt}, w_{j'} \geq w_i} p_{j't}$. Hence jobs arrival can only decrease the potential. Jobs completion either in the algorithm's schedule or the optimal schedule has no effect on the potential. Hence the potential can only decrease when discontinuous changes occur.

We now focus on *continuous* changes of $\Phi(t)$ which occur when no jobs arrive or complete. As before, let $\frac{d}{dt}(.)|_A$ denote the derivative of the quantity in the parenthesis due to A's processing freezing OPT's processing. We define $\frac{d}{dt}(.)|_O$ similarly. So $\frac{d}{dt}\Phi(t) = \frac{d}{dt}\Phi(t)|_A + \frac{d}{dt}\Phi(t)|_O$. Knowing that the potential is 0 both at time 0 and at a time T when all jobs have been completed by our algorithm and OPT, and no discontinuous changes increase the potential, we have $\int_{t=0}^{T} \frac{d}{dt}\Phi(t)dt \geq 0$. Our main goal is to show,

$$\int_{t=0}^{T} \frac{d}{dt}\Phi(t)dt \leq -\frac{\epsilon}{K} \int_{t=0}^{T} \sum_{q\in[K]} \hat{W}_{qt}dt + O(1) \int_{t=0}^{T} \sum_{q\in[K]} \hat{W}^O_{qt}dt \tag{7}$$

where $\hat{W}_{qt} := \sum_{j\in J_{qt}} w_j p_{jt}$ and $\hat{W}^O_{qt} := \sum_{j\in J_{qt}} w_j p^O_{jt}$. This, combined with the fact that $\int_{t=0}^{T} \frac{d}{dt}\Phi(t)dt \geq 0$, will immediately yield the first part of Theorem III.4.

It now remains to show Equation (7). Assuming that the algorithm is given $(1+\epsilon)$-speed, it processes only the highest weight job 1_q, at a rate of $(1+\epsilon)z_{qt}$; here we omitted t from 1_{qt} for notational simplicity. Considering

each queue q, we derive,

$$\begin{aligned}
&\frac{d}{dt}\tilde{C}_q(t)|_A \\
&\leq \left(\sum_{j\in J_{qt}} w_j p_{jt}\right)\cdot(-(1+\epsilon)z_{qt}) + w_{1_q}(1+\epsilon)z_{qt}\cdot\left(\sum_{j'\in J_{qt}, w_{j'}\geq w_{1_q}} p^O_{j't}\right) + \left(\sum_{j'\in J_{qt}, w_j\leq w_{1_q}} w_j p^O_{jt}\right)(1+\epsilon)z_{qt} \\
&\leq -(1+\epsilon)z_{qt}\cdot\hat{W}_{qt} + 2(1+\epsilon)z_{qt}\cdot\hat{W}^O_{qt}
\end{aligned}$$

Say OPT processes job i in queue q at a rate of z^O_{qt}.

$$\begin{aligned}
\frac{d}{dt}\tilde{C}_q(t)|_O &\leq \left(\sum_{j\in J_{qt}, w_j\leq w_i} w_j p_{jt}\right) z^O_{qt} + w_i z^O_{qt}\left(\sum_{j'\in J_{qt}, w_{j'}\geq w_i} p_{j't}\right) \\
&\leq z^O_{qt}\cdot\hat{W}_{qt} + z^O_{qt}\cdot w_i p_{it}
\end{aligned}$$

For a while, we proceed our analysis ignoring the term $z^O_{qt}\cdot w_i p_{it}$ – we will bring it in the picture at the end of the analysis. Then, we have,

$$\begin{aligned}
\frac{d}{dt}\Phi(t) &= \frac{d}{dt}\Phi(t)|_A + \frac{d}{dt}\Phi(t)|_O \\
&\leq -(1+\epsilon)\sum_{q\in[K]} z_{qt}\cdot\hat{W}_{qt} + \sum_{q\in[K]} z^O_{qt}\cdot\hat{W}_{qt} + 2(1+\epsilon)\sum_{q\in[K]} z_{qt}\cdot\hat{W}^O_{qt} \\
&\leq -\epsilon\sum_{q\in[K]} z_{qt}\cdot\hat{W}_{qt} + 2(1+\epsilon)\sum_{q\in[K]} z_{qt}\cdot\hat{W}^O_{qt} && \text{[Proposition III.5]} \\
&\leq -\frac{\epsilon}{K}\sum_{q\in[K]}\hat{W}_{qt} + 2(1+\epsilon)\sum_{q\in[K]}\hat{W}^O_{qt} && \text{[Observation 3 and } z_{qt}\leq 1\text{]}
\end{aligned}$$

Integrating this inequality gives Equation (7).

We now bring in the term $z^O_{qt}\cdot w_i p_{it}$ we ignored. Note that we need to add this term to $\frac{d}{dt}\Phi(t)$ only when the optimal scheduler processes job i at time t. What this means that OPT processing job i by δ units contributes to $\int_{t=0}^{T}\frac{d}{dt}\Phi(t)$ by at most δw_i; recall that $p_{it}\leq 1$. Hence each job i can add at most w_i to $\int_{t=0}^{T}\frac{d}{dt}\Phi(t)$ when the ignored term is considered. Observe that in OPT with 1-speed, each job i's weighted fractional flow time is no smaller than $w_i/2$; here we used the fact $z_{qt}\leq 1$. Hence the ignored term can only increase $\int_{t=0}^{T}\frac{d}{dt}\Phi(t)dt$ by at most twice OPT's fractional weighted flow time. This shows Equation (7), completing the proof of Theorem III.4.

C. Resource Allocation with Complementarities (RA-C)

Recall the RA-C problem from Section I-C1. There are D divisible resources (or dimensions), numbered $1, 2, \ldots, D$. We assume w.l.o.g. (by scaling and splitting resources) that each resource is available in unit supply. If job j is assigned a non-negative vector of resources $\mathbf{x} = \{x_1, x_2, \ldots, x_D\}$, then the rate at which the job executes is given by $y_j = u_j(\mathbf{x})$ where $u_j(\mathbf{x}_j) = \min_{d=1}^{D}(c_{jd}x_{jd})$ are Leontief utilities. Here, $\mathbf{c_j}$ is the *resource vector* of the job.

When the number of distinct resource vectors (or job types) is bounded by K, then this problem is clearly a special case of PSP-Q with K queues. In this section, we show that even when the resource vectors can be arbitrary, the RA-C problem can be reduced to PSP-Q with a bounded number of queues. In the reduction, jobs of "similar" resource vectors will be discretized to the same type.

Lemma III.6. *If we use a $(1+\epsilon)$ factor more speed, then we can w.l.o.g. assume that for all jobs j: (1) $\min_{d=1}^{D} c_{jd} = 1$; and (2) $c_{jd}\in[1, D/\epsilon]\cup\{\infty\}$ for all $d\in[D]$.*

Here if $c_{jd}=\infty$, $c_{jd}\cdot x_{jd}$ is defined to be ∞ for any value of x_{jd} (meaning that d is never a bottleneck in determining j's utility). More precisely, if we have a s-speed f-competitive algorithm for this simplified instance, then we can derive an algorithm that is $s(1+\epsilon)$-speed f-competitive for the original instance.

Proof: Fix a job j. Let $h = \min_{d=1}^{D} c_{jd}$. We first normalize $\mathbf{c_j}$ so that the first property is satisfied. Towards this end, we scale down $\mathbf{c}'_j$ and p_j by the same factor h. Note that these changes do not affect the schedule. More precisely, any feasible schedule $\mathbf{x}_{jt}$ for the original instance is also feasible for the new modified instance, and

further all jobs completion times remain the same. So far we have shown the first property can be satisfied w.l.o.g. Henceforth, for notational convenience, let's assume that the original instance of jobs j with $\mathbf{c_j}$ satisfies the first property.

Now turning to the second property, if $c_{jd} \geq D/\epsilon$, then we simply set c_{jd} to ∞. Let $\mathbf{c'_j}$ denote the new vector for job j. To see why we can make this change w.l.o.g., consider any feasible resource allocation x'_{jd} at a fixed time t for this new instance. Here, we can assume w.l.o.g. that $x'_{jd} = 0$ if $c'_{jd} = \infty$, and $c'_{jd}x'_{jd}$ is the same for all d with $c'_{jd} < \infty$ since otherwise resources would be wasted. We would like to construct $\{x_{jd}\}$ such that each job gets processed at the same rate under the schedule $\{x_{jd}\}$ for the original instance as it does under the schedule $\{x'_{jq}\}$ for the modified instance. Towards this end, for any d with $c'_{jd} = \infty$ and $c_{jd} < \infty$, i.e. $D/\epsilon < c_{jd} < \infty$, we set x_{jd} so that $c_{jd}x_{jd}$ are the same for all d with $c_{jd} < \infty$. We say that a job j' is tightest for dimension d' if $c_{j'd'} = 1$. The first property implies that every job is tightest for some dimensions; a job can be tightest for multiple dimensions. Let $J_{d'}$ denote the jobs that are the tightest for dimension d'.

The issue is that the resource allocation $\{x_{jd}\}$ may over-allocate resources. However, we will show that over-allocation can be fixed by a small amount of speed augmentation. We now show that $\sum_j (x_{jd} - x'_{jd}) \leq \epsilon$ for all $d \in [D]$. This will imply that $\sum_j x_{jd} \leq \epsilon + \sum_j x'_{jd} \leq 1 + \epsilon$. Hence if we use a resource allocation $x_{jd}/(1+\epsilon)$ with a $(1+\epsilon)$ factor more speed, we will have a feasible schedule while processing each job's rate. Considering any fixed d^*, we have,

$$\begin{aligned}
\sum_j (x_{jd^*} - x'_{jd^*}) &\leq \sum_{d' \in [D]} \sum_{j \in J_{d'} : D/\epsilon \leq c_{jd^*} < \infty} (x_{jd^*} - x'_{jd^*}) && [x_{jd^*} - x'_{jd^*} > 0 \text{ only if } D/\epsilon \leq c_{jd^*} < \infty] \\
&= \sum_{d' \in [D]} \sum_{j \in J_{d'} : D/\epsilon \leq c_{jd^*} < \infty} x_{jd^*} && [x'_{jd^*} = 0 \text{ if } D/\epsilon \leq c_{jd^*}] \\
&= \sum_{d' \in [D]} \sum_{j \in J_{d'} : D/\epsilon \leq c_{jd^*} < \infty} (1/c_{jd^*}) x_{jd'} && [c_{jd^*} x_{jd^*} = c_{jd'} x_{jd'} = x_{jd'}] \\
&\leq \sum_{d' \in [D]} \sum_{j \in J_{d'}} (\epsilon/D) x_{jd'} \leq (\epsilon/D) \sum_{d' \in [D]} 1 \leq \epsilon
\end{aligned}$$

■

We are now ready to describe the reduction. Assume that the given resource vectors satisfy the properties stated in the above lemma. We discretize each job's resource vectors as follows. For each $d \in [D]$, if $c_{jd} < \infty$, then round up c_{jd} to the closest power of $(1+\epsilon)$. We know that x_{jd} can have at most $\lceil \log_{(1+\epsilon)}(D/\epsilon) \rceil + 2) \leq O((1/\epsilon)\log D)$ different values. Hence there are at most $(O(1/\epsilon)\log D)^D$ distinct resource vectors. Jobs of the same resource vector are placed into the same queue. It is easy to see using another $(1+\epsilon)$ factor more speed takes care of the rate loss due to the discretization. Hence we have shown a reduction of arbitrary RA-C instances with D dimensions to PSP-Q with $K = O((1/\epsilon)\log D)^D$ queues by using $(1+\epsilon)$ factor more speed.

IV. Conclusions

We conclude with some open questions. An immediate open question is to extend Theorem I.2 so that the speed is $1+\epsilon$ for any $\epsilon > 0$. Though this can be done in the single-machine setting [18], that analysis crucially required a closed form for the allocations and rates. Our analysis is based on optimality conditions, and extending it seems to encounter fundamental roadblocks. The next open question is to design an algorithm for RA-C whose competitive ratio is poly(D), where D is the number of dimensions. Note that in full generality, the number of queues K can be exponential in D. Reducing this dependence will require fundamentally new ideas. A related question is to show tight lower bounds for PSP-Q even with clairvoyance – the current lower bound [24] that is polynomial in K holds only for non-clairvoyant algorithms.

A more open-ended question is to characterize the class Monotone PSP and study its precise connection to market clearing, extending the work of [28]. As we emphasized, we crucially require the optimality condition of PF. In most cases, even if a market clearing solution exists, the PF algorithm will not find this solution, and therefore we cannot use monotonicity characterizations from market clearing literature. A related question will be to study other algorithms whose optimality conditions have simple characterizations.

Acknowledgments.: We thank Benjamin Lee and Seyed Zahedi for many helpful discussions and for pointing us to [42]. Im is supported in part by NSF grant CCF-1409130. Kulkarni is supported by NSF grants CCF-0745761, CCF-1008065, and CCF-1348696. Munagala is supported in part by NSF grants CCF-1348696, CCF-1408784, and IIS-1447554, and by grant W911NF-14-1-0366 from the Army Research Office (ARO).

REFERENCES

[1] http://aws.amazon.com/ec2/spot-instances/.

[2] http://hadoop.apache.org.

[3] Faraz Ahmad, Srimat T. Chakradhar, Anand Raghunathan, and T. N. Vijaykumar. Tarazu: optimizing mapreduce on heterogeneous clusters. In *ASPLOS*, pages 61–74. ACM, 2012.

[4] S. Anand, Naveen Garg, and Amit Kumar. Resource augmentation for weighted flow-time explained by dual fitting. In *SODA*, pages 1228–1241, 2012.

[5] Nikhil Bansal and Ho-Leung Chan. Weighted flow time does not admit o(1)-competitive algorithms. In *Proceedings of the Twentieth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '09, pages 1238–1244, Philadelphia, PA, USA, 2009. Society for Industrial and Applied Mathematics.

[6] Nikhil Bansal, Moses Charikar, Ravishankar Krishnaswamy, and Shi Li. Better scalable algorithms for broadcast scheduling. In *SODA*, pages 55–71, 2014.

[7] Nikhil Bansal, Don Coppersmith, and Maxim Sviridenko. Improved approximation algorithms for broadcast scheduling. *SIAM J. Comput.*, 38(3):1157–1174, 2008.

[8] Nikhil Bansal, Ravishankar Krishnaswamy, and Viswanath Nagarajan. Better scalable algorithms for broadcast scheduling. In *ICALP (1)*, pages 324–335, 2010.

[9] Ted Bergstrom. Proving that a cobb-douglas function is concave if the sum of exponents is no bigger than 1, Manuscript, 2014.

[10] S. Bikhchandani, S. de Vries, J. Schummer, and R. V Vohra. An ascending vickrey auction for selling bases of a matroid. *Operations research*, 59(2):400–413, 2011.

[11] T. Bonald, L. Massoulié, A. Proutière, and J. Virtamo. A queueing analysis of max-min fairness, proportional fairness and balanced fairness. *Queueing Syst. Theory Appl.*, 53(1-2):65–84, June 2006.

[12] 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.

[13] Ho-Leung Chan, Jeff Edmonds, and Kirk Pruhs. Speed scaling of processes with arbitrary speedup curves on a multiprocessor. In *SPAA*, pages 1–10, 2009.

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

[15] Jeff Edmonds, Sungjin Im, and Benjamin Moseley. Online scalable scheduling for the ℓ_k-norms of flow time without conservation of work. In *ACM-SIAM Symposium on Discrete Algorithms*, 2011.

[16] Jeff Edmonds and Kirk Pruhs. Scalably scheduling processes with arbitrary speedup curves. *ACM Transactions on Algorithms*, 8(3):28, 2012.

[17] J. Feldman, S. Muthukrishnan, E. Nikolova, and M. Pál. A truthful mechanism for offline ad slot scheduling. In *SAGT*, pages 182–193, 2008.

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

[19] Rajiv Gandhi, Samir Khuller, Srinivasan Parthasarathy, and Aravind Srinivasan. Dependent rounding and its applications to approximation algorithms. *J. ACM*, 53(3):324–360, 2006.

[20] 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.

[21] 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.

[22] F. Gul and E. Stacchetti. Walrasian equilibrium with gross substitutes. *Journal of Economic Theory*, 87(1):95 – 124, 1999.

[23] Anupam Gupta, Sungjin Im, Ravishankar Krishnaswamy, Benjamin Moseley, and Kirk Pruhs. Scheduling heterogeneous processors isn't as easy as you think. In *SODA*, pages 1242–1253, 2012.

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

[25] 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.

[26] Sungjin Im and Benjamin Moseley. An online scalable algorithm for average flow time in broadcast scheduling. *ACM Transactions on Algorithms*, 8(4):39, 2012.

[27] Sungjin Im, Benjamin Moseley, and Kirk Pruhs. A tutorial on amortized local competitiveness in online scheduling. *SIGACT News*, 42(2):83–97, 2011.

[28] K. Jain and V. V. Vazirani. Eisenberg-Gale markets: Algorithms and game-theoretic properties. *Games and Economic Behavior*, 70(1):84–106, 2010.

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

[30] F. P. Kelly, A. K. Maulloo, and D. K. H. Tan. Rate control for communication networks: Shadow prices, proportional fairness and stability. *The Journal of the Operational Research Society*, 49(3):pp. 237–252, 1998.

[31] F.P. Kelly, L. Massoulié, and N.S. Walton. Resource pooling in congested networks: proportional fairness and product form. *Queueing Systems*, 63(1-4):165–194, 2009.

[32] 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.

[33] S. T. Maguluri and R. Srikant. Scheduling jobs with unknown duration in clouds. In *INFOCOM*, pages 1887–1895, 2013.

[34] Nick McKeown, Adisak Mekkittikul, Venkat Anantharam, and Jean Walrand. Achieving 100% throughput in an input-queued switch. *IEEE Transactions on Communications*, 47(8):1260–1267, 1999.

[35] N. Megiddo. Optimal flows in networks with multiple sources and sinks. *Mathematical Programming*, 7(1):97–107, 1974.

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

[37] 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.

[38] Julien Robert and Nicolas Schabanel. Non-clairvoyant scheduling with precedence constraints. In *Proceedings of the nineteenth annual ACM-SIAM symposium on Discrete algorithms*, SODA '08, pages 491–500, 2008.

[39] K. Shvachko, H. Kuang, S. Radia, and R. Chansler. The hadoop distributed file system. In *IEEE MSST*, pages 1–10, 2010.

[40] Leandros Tassiulas and Anthony Ephremides. Stability properties of constrained queueing systems and scheduling policies for maximum throughput in multihop radio networks. *IEEE Transactions on Automatic Control*, 37(12):1936–1948, 1992.

[41] 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.

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