Temporal Fairness of Round Robin: Competitive Analysis for Lk-norms of Flow Time

Sungjin Im
University of California, Merced
Merced, CA 95343
sim3@ucmerced.edu

Janardhan Kulkarni
Duke University
Durham, NC 27708
kulkarni@cs.duke.edu

Benjamin Moseley
Washington University in St. Louis.
St. Louis, MO 63130
bmoseley@wustl.edu

ABSTRACT

Fairness is an important criterion considered in scheduling together with overall job latency. Round Robin (RR) is a popular scheduling policy that distributes resources to jobs equally at any point in time guaranteeing instantaneous fairness of jobs. In this paper we give the *first* analysis of RR for the ℓ_2-norm of flow time and show that it is $O(1)$-speed $O(1)$-competitive on multiple machines. The ℓ_2-norm is a popular scheduling objective that makes a natural balance between temporal fairness and jobs latency. Prior to our work, RR has not been analyzed for the ℓ_2-norm even in the single machine setting. Our result establishes that RR is fair not only instantaneously but also *temporarily*.

Categories and Subject Descriptors

F.2.2 [**Nonnumerical Algorithms and Problem**]: Sequencing and scheduling

General Terms

Algorithms, Theory

Keywords

Online scheduling, Round Robin, Fairness.

1. INTRODUCTION

Most server-client scheduling settings can be characterized by a set of servers/machines/processors and clients who send their requests to the servers over time for service. Each request/job j typically has a processing time/size p_j and an arrival time r_j, which is the first time the scheduler is aware of the existence of the job and can start processing it. Commonly, each client wants her job j to be completed as early as possible, i.e. the job j' s flow time to be minimized. A job j's response/flow time is defined as its completion time C_j minus its arrival time r_j, the amount of time the job waits in the system to be satisfied.

SPAA'15, June 13–15, 2015, Portland, OR, USA.

http://dx.doi.org/10.1145/2755573.2755581.

However, when multiple jobs compete for limited resources to get service earlier, the scheduler must prioritize between jobs. While there are many scheduling objectives that could be optimized, the most popular is the objective of minimizing the total (or equivalently average) flow time, i.e. $\sum_j (C_j - r_j)$. Indeed, algorithms have been analyzed in various setting for the total flow objective in the competitive analysis framework. An online scheduling policy is said to be c-competitive for a certain objective if its objective is at most a c factor more than the optimal scheduler's objective for any sequence of jobs. Competitive analysis is of fundamental importance since performance guarantees hold even in the worst scenarios.

Unfortunately, competitive analysis often yields pessimistic results. For example, it is folklore that Shortest Remaining Time Processing (SRPT) is optimal, i.e. 1-competitive, in the single machine setting for the average flow time objective. However, it is known that no online algorithm is $O(1)$-competitive when there are multiple machines [23]. To alleviate this pessimistic view, the resource augmentation model was proposed by [22]. In the resource augmentation model, if an online algorithm is given s times faster machine(s) and has an objective at most c times the optimal objective, then the algorithm is said to be s-speed c-competitive. In general, $O(1)$-speed $O(1)$-competitiveness is considered to be an indicator of "reasonably good" scheduling algorithms although $(1+\epsilon)$-speed is $O(1)$-competitiveness for any fixed $\epsilon > 0$, a.k.a. scalability, is the most desirable. Resource augmentation has enabled the development and analysis of algorithms for various scheduling environments.

Another important scheduling criterion is *fairness* which is not captured by objectives focusing on the overall job latency such as the average flow time objective. There are largely two types of fairness typically considered, *instantaneous* and *global*. For instantaneous fairness, the main goal is to distribute resources to jobs evenly at each moment in time. While the notion of fairness can be somewhat subjective, there is a widely accepted fair algorithm when the resources are homogeneous/identical, Round Robin. Round Robin (RR) is an algorithm that achieves fairness by giving an equal share of the machine(s) to all jobs at all times. This fairness also coincides with maximizing the minimum fairness, which is the most widely accepted fairness notion in many disciplines.

The other type of fairness, which we call temporal fairness, is measured by global scheduling objectives. As mentioned before, average flow time only measures the average latency of job service times, thereby potentially allowing some jobs

to starve for service for an unacceptably long time. However, Silberschatz, Galvin and Gagne's classic textbook on Operating Systems [26] states "*A system with reasonable and predictable response time may be considered more desirable than a system that is faster on the average, but is highly variable.*" and "*... for interactive systems, it is more important to minimize the variance in the response time than it is to minimize the average response time.*"

Motivated by temporal fairness, Bansal and Pruhs analyzed various scheduling policies such as SRPT and Shortest Job First (SJF) for minimizing the ℓ_2-norms of flow time (more generally ℓ_k-norms [1] of flow time, $(\sum_j (C_j - r_j)^k)^{1/k}$) in an influential paper [4], and showed they are $O(1)$-speed $O(1)$-competitive. Intuitively, the ℓ_2-norm of flow time objective attempts to minimize the variance of flow times of jobs as well as the average. However, the aforementioned algorithms do not provide instantaneous fairness since they may schedule only one job for a long time until the priorities change due to job processing or other jobs arrival.

A natural looming question is if there is an algorithm that achieves the instantaneous and global fairness simultaneously while optimizing the overall job latency. The RR algorithm is widely used in situations where fairness is an important criteria. For example, [8, 17, 25]. However, despite the algorithm being used for its fair properties, it is not known to be fair globally. It is known that RR is $O(1)$-speed $O(1)$-competitive for average flow time [11, 13]. However, RR has no known guarantees for the ℓ_2-norm of flow time and it has remained an open question if this algorithm, which is intuitively as fair as possible, is in fact temporally fair even in the single machine scheduling setting.

1.1 Our Results

In this paper we answer the above question in the affirmative by giving the first analysis of the instantaneously fair algorithm Round Robin (RR) for the ℓ_k-norms of flow time. We show that RR has provable guarantees for temporal fairness in the multiple identical machines setting. The algorithm RR has a natural interpretation in this setting: at any point in time when there are more jobs than machines, allocate machines to jobs equally. Otherwise, process each job on one machine exclusively. We show that this algorithm has the following guarantees.

Theorem 1 *The scheduling policy Round Robin (*RR*) is $2k(1+10\epsilon)$-speed $O(\frac{k}{\epsilon})$-competitive for the ℓ_k-norm of flow time for any $0 < \epsilon \leq 1/10$ and all $k \geq 1$. Furthermore, this result holds even when there are multiple identical machines.*

In particular, our analysis shows that RR is $(4+\epsilon)$-speed $O(1)$-competitive for the ℓ_2-norm of flow time for any fixed $\epsilon > 0$. We emphasize that the same algorithm RR is $O(1)$-speed $O(1)$-competitive for both the ℓ_1 and ℓ_2-norms of flow time simultaneously. Prior to our work, it was unknown if RR is $O(1)$-speed $O(1)$-competitive for the ℓ_2-norm even in the simplest single machine setting.

We note that it is known that RR is $\Omega(n^{1-2\epsilon p})$-competitive when given $(1+\epsilon)$-speed [4]. Particularly, this means that RR is not $O(1)$-competitive with speed less than $3/2$ for the ℓ_2-norm objective. On the positive side, our result shows RR is $O(1)$-competitive with speed $4+\epsilon$.

[1] In practice, $k \in [1, 3] \cup \{\infty\}$ are considered.

1.2 Our Techniques and Backstory

Perhaps the reason why the temporal fairness of RR was not studied before is due to the underlying technical challenges arising in the analysis. The first analysis of RR for the ℓ_1-norm of flow time was algebraically very involved [11]. This was in a stark contrast to the fact that other simple scheduling policies such as SRPT and SJF have considerably simpler analysis. This is no surprise considering that the sharing aspect of RR does not result in an mathematically easy expression of total flow time unlike other scheduling policies.

A significantly simpler analysis of RR for the ℓ_1-norm was later given by Edmonds and Pruhs [13]. They gave a very elegant analysis using a novel potential function, and the analysis has been extremely useful for other various scheduling problems; for an overview of potential function based analysis of online scheduling algorithms, see [21]. However, the ℓ_2-norm is very different from the ℓ_1-norm since older jobs contribute more to the objective than young jobs and the analysis of RR for the ℓ_1-norm does not extend to the ℓ_2-norm. Another possible analysis approach to study RR for the ℓ_2-norm is linear programming and dual fitting, which was recently introduced for scheduling analysis [1, 16].

A potential issue with using potential functions or dual fitting is that the analysis seems to require a weighted version of RR. Let's focus on the total square of flow times of jobs, i.e. $\sum_j (C_j - r_j)^2$ by peeling out the square root of the ℓ_2-norm objective. As mentioned before, when jobs arrive over time, RR does not admit a closed form of mathematical expression of the square objective. Hence an alternative approach is to focus on the instantaneous increase of the square objective in the RR's schedule at each time. At any point in time each alive job j contributes by twice the job's current age to the instantaneous increase in the objective. Here if jobs are given machines in proportion to their ages (a 'weighted' version of RR), both the potential function and dual fitting approaches go through relatively easily. However, the analysis of RR is significantly more challenging since it is oblivious to jobs ages.

Even algorithmically, it was not clear if RR was a right algorithm for the ℓ_2-norm of flow time. In fact, in other scheduling environments such as the arbitrary speed-up curves and broadcast settings, RR was shown not to be $O(1)$-speed $O(1)$-competitive [15]. However, the weighted variant of RR that distributes machines to jobs in proportion to their ages was shown to be $O(1)$-speed $O(1)$-competitive for the ℓ_2-norm [12]. These results suggest that there was no strong reason to believe RR would perform well in the standard scheduling setting for the ℓ_2-norm.

Our analysis is based on linear programming and dual fitting. The challenge is how to set up dual variables. While our analysis inspired by the recent work [18, 19], our setting of dual variables is very different from the previous work. Roughly speaking, there are two types of dual variables that one is required to set, how much each job is responsible for the objective, and how much a job contributes to each time t. A naive extension of the previous work fails. In particular, as alluded to before, we had to add "global" quantities to both variables not only just the instantaneous quantities such as jobs ages. This will be further discussed in Section 3.2.

1.3 Related Work

As mentioned before, it is well known that SRPT is optimal for the ℓ_1-norm of flow time in the single machine setting. The algorithms SJF and Shortest Elapsed Time First (SETF) are known to be $O(1+\epsilon)$-speed $O(1)$-competitive [7, 22]. The same objective was extensively studied in the multiple machines setting [2, 3, 6, 9, 10, 14, 23, 27]. In particular, SRPT and SJF both are $O(1+\epsilon)$-speed $O(1)$-competitive on multiple machines [14, 27]. However, only a "fractional" version of SETF was shown to be scalable [5] on multiple machines.

For ℓ_k-norms of flow time, [4] showed various algorithm are scalable ($(1+\epsilon)$-speed $O(1)$-competitive), including SRPT, SJF, and SETF. It is known that SJF and SRPT are scalable for ℓ_k-norms of flow time even on multiple machines [14, 27].

In [13], RR and its extension were analyzed. The setting was the arbitrary speed-up curves setting where each job can be sped up by being assigned more machines, and can have a different degree of parallelizability. In the closely related broadcast scheduling setting, jobs asking for the same data can be processed simultaneously. As mentioned, while RR is $O(1)$-speed $O(1)$-competitive for the ℓ_1-norm in both settings [12], it is not $O(1)$-competitive even with any $O(1)$-speed for the ℓ_2-norm [15].

For recent results for heterogeneous machines, see [1, 19, 20]. For a nice (but slightly outdated) survey on online scheduling, we refer the reader to [24].

2. PROBLEM DEFINITION AND NOTATION

In this section, we formally define the problem along with notation which will be used throughout this paper. There are m identical machines available. Each job j arrives at time r_j, and this is the first time when the online scheduler learns about job j. In a feasible schedule, each machine can schedule at most one job at each moment in time. Let $A(t)$ denote the set of jobs alive at time t in the schedule of the online algorithm we consider. A feasible schedule can alternatively be characterized by $\{m_j(t)\}_{j\in A(t)}$ at each time t such that $\sum_j m_j(t) \le 1$, and $m_j(t) \in [0,1]$ for all jobs j and times $t \ge 0$. That is, any $\{m_j(t)\}$ satisfying these constraints can be easily translated into a feasible schedule that fits in the former definition, and vice versa. It is easy to see that in the RR's schedule,

$$m_j(t) = \min\{1, m/n_t\},$$

where $n_t := |A(t)|$ is the number of jobs alive at time t.

Job j completes at time t when it gets p_j amount of total processing since its arrival. Note that RR does not need to know job j's size, p_j, until its completion, and such an algorithm is called non-clairvoyant. In the ℓ_k norms of flow time, the goal is to minimize $\sqrt[k]{\sum_{j\in[n]}(C_j - r_j)^k}$.

We refine $A(t)$ by adding an extra constraint to refer to jobs that are alive at time t and satisfy the constraint. For example, $A(t, \le r_j)$ refers to jobs alive at time t and arrive before job j. For notational simplicity, we may use RR to denote RR's objective. Likewise, OPT may denote the optimal scheduler's objective, not only the scheduling policy itself. The flow time of job j in RR's schedule will be denoted F_j.

3. ANALYSIS

3.1 LP relaxation

In this section we will prove Theorem 1. We assume throughout the analysis that RR is given $\eta := 2k(1+10\epsilon)$-speed where $0 < \epsilon \le 1/10$. Note that every job $j \in A(t)$ is processed at a rate of $\eta \cdot \min\{1, m/n_t\}$ in RR's schedule at each time t where $n_t := |A(t)|$ denotes the number of jobs alive at time t. As mentioned earlier, our analysis will be based on dual fitting which was first used by [1, 16] in online scheduling literature. It will be more convenient to compare RR's kth power of flow time to the analogous quantity of the optimal scheduler – the competitive analysis of ℓ_k norms of flow time will immediately follow by taking the kth root on the kth power of flow time objective.

A linear programming relaxation of the problem $\mathsf{LP}_{\mathsf{primal}}$ is described as follows. The variables x_{jt} denote the rate at which job j is processed at time t. The first constraint says that every job must be completely processed. The second constraint says the total rate at which all jobs are processed at each time t is upper bounded by m, the number of machines available. (Alternatively, one can think of m machines as one super machine with speed m.) In this LP relaxation, a job is allowed to be processed simultaneously across different machines, but we assume a feasible algorithm can schedule a job on at most one machine at each point in time. In the LP, let $\gamma = k(k/\epsilon)^{k-1}$ be a constant that depends on ϵ and k. For technical reasons in the dual fitting analysis, we add a factor γ to the objective. We note that this simply increases the value of the primal by a factor γ.

$$\text{Min} \quad \sum_j \sum_{t\ge r_j} \gamma \left(\frac{(t-r_j)^k}{p_j} + \frac{p_j^k}{p_j} \right) \cdot x_{jt} \qquad (\mathsf{LP}_{\mathsf{primal}})$$

$$\begin{aligned}
\sum_{t\ge r_j} x_{jt} &\ge p_j && \forall j \\
\sum_{j\,:\,t\ge r_j} x_{jt} &\le m && \forall t \\
x_{jt} &\ge 0 && \forall j, t \,:\, t \ge r_j
\end{aligned}$$

Observe that the above LP lower bounds the optimal flow time of a feasible schedule within factor 2γ. This is becuase $t - r_j \le F_j$ if job j is alive at time t, and job j's flow time is at least p_j in any feasible schedule. Next we write the dual of $\mathsf{LP}_{\mathsf{primal}}$.

$$\text{Max} \quad \sum_j \alpha_j - \sum_t \beta_t \qquad (\mathsf{LP}_{\mathsf{dual}})$$

$$\begin{aligned}
\frac{\alpha_j}{p_j} - \frac{\beta_t}{m} &\le \gamma \left(\frac{(t-r_j)^k}{p_j} + \frac{p_j^k}{p_j} \right) && \forall j, t \,:\, t \ge r_j \\
\alpha_j &\ge 0 && \forall j \\
\beta_t &\ge 0 && \forall t
\end{aligned}$$

The dual has a variable α_j for every job j corresponding to the first constraint in the primal and a variable β_t corresponding to the second constraint. We will show that there is a setting of dual variables such that objective function of the dual is at least $\Omega(\epsilon)$ times RR's kth power of flow time. This will imply that RR's kth power of flow time is at most

$O(1/\epsilon)$ times the dual objective, hence $O(\gamma/\epsilon)$ times the optimal scheduler's kth power of flow time. By taking the kth root, we will have Theorem 1.

3.2 Setting of Dual Variables

Let $T_o := \{t \mid |A(t)| \geq m\}$ denote the set of *overloaded times* where all m machines are busy in RR's schedule (in other words, total number jobs available for processing is at least m), and T_u denote the other times which will be referred to as *underloaded times*. We set α_j as follows:

$$\alpha_j = \left(\sum_{t' \in [r_j, C_j] \cap T_o} \sum_{j' \in A(t', \leq r_j)} \frac{k(t' - r_{j'})^{k-1}}{n_{t'}} \right) + \left(\sum_{t' \in [r_j, C_j] \cap T_u} k(t - r_j)^{k-1} \right) - \epsilon F_j^k \qquad \forall j$$

Recall that $A(t', \leq r_j)$ denotes the set of jobs that are alive at time t and have arrived no later than job j (including job j itself).

We continue to set β_t as follows. Below, for a fixed time t let $\mathbf{1}(t \in [r_j, C_j + \delta F_j])$ be 1 if $t \in [r_j, C_j + \delta F_j])$ and 0 otherwise; δ will be set to ϵ later.

$$\beta_{jt} = (1/2 - 3\epsilon) \cdot \mathbf{1}(t \in [r_j, C_j + \delta F_j]) \cdot F_j^{k-1} \qquad \forall j, t$$
$$\beta_t = \sum_j \beta_{jt} \qquad \forall t$$

Note that a job j can contribute to β_t not only during its lifespan $[r_j, C_j]$ but also even after it gets completed, for $\delta F_j = \delta(C_j - r_j)$ time steps. The amount j adds to β_t at time t (if it does) is F_j^{k-1}, the $(k-1)$th power of j's flow time.

We briefly discuss how our setting of dual variables differentiates from the previous work, starting with dual variables α_j. We had to distinguish between overloaded and underloaded time steps. This is because RR behaves on m machines as it does on 1 machine with speed m when machines are overloaded, and otherwise, it schedules each job on a separate machine exclusively. The summation over overloaded times is inspired by the work [13, 19] – in particular, this amortized accounting is exactly the same when $k = 1$. However, we had to subtract ϵF_j^k from α_j. Hence while the first two terms in α_j are derived from RR's instantaneous status, the last term is a global term which is derived from j's final flow time. Also we had to make job j contribute to β_j for a while even after it completes. This helps compare job j to job j' that is considered in α_j, and turns out to be very useful for our analysis.

3.3 Bounding The Dual Objective

We start with showing that the dual objective for the above setting of α_j and β_t is $\Omega(\epsilon)$ times RR's kth power of flow time. Throughout the analysis we will use the fact that for any job $j \in A(t)$ it is the case that $(t - r_j) \leq F_j$ because the job must be alive at time t to be in $A(t)$.

Lemma 1 $\sum_j \alpha_j \geq (\frac{1}{2} - \epsilon)\mathrm{RR}$.

Proof.

$$\begin{aligned}
\sum_j \alpha_j &= \sum_{t' \in T_o} \sum_{j \in A(t')} k(t' - r_j)^{k-1} \frac{|A(t', \geq r_j)|}{n_{t'}} \\
&\quad + \sum_{t' \in T_u} \sum_{j' \in A(t')} k(t' - r_{j'})^{k-1} - \epsilon \mathrm{RR} \\
&\geq \frac{1}{2} \sum_{t' \in T_o} \sum_{j \in A(t')} k(t' - r_j)^{k-1} \\
&\quad + \sum_{t' \in T_u} \sum_{j' \in A(t')} k(t' - r_{j'})^{k-1} - \epsilon \mathrm{RR} \\
&\geq (\frac{1}{2} - \epsilon)\mathrm{RR}
\end{aligned}$$

Since verifying the above sequence of equations is straightforward for underloaded time steps, we will focus on overloaded time steps. The first equality follows by observing that the term $\frac{k(t'-r_{j'})^{k-1}}{n_{t'}}$ due to job j' is counted by all jobs in $A(t')$ that arrive no earlier than job j'. The second inequality holds for the following reason. Fix time t' and define j's rank as $|A(t', \geq r_j)|$, denoted as π_j. Note that the earliest arriving job in $A(t')$ has rank $n_{t'}$ and the latest arriving job has rank 1. Also observe $k(t - r_i)^{k-1} \geq k(t - r_j)^{k-1}$ if and only if $\pi_i \geq \pi_j$ since earlier arriving jobs have higher ranks. By pairing two jobs i and j such that $\pi_i + \pi_j = n + 1$, we have

$$\begin{aligned}
&(k(t' - r_i)^{k-1}\pi_i + k(t' - r_j)^{k-1}\pi_j)/n_{t'} \\
&\geq \frac{n_{t'} + 1}{2n_{t'}}(k(t' - r_i)^{k-1} + k(t' - r_j)^{k-1}) \\
&\geq \frac{1}{2}(k(t' - r_i)^{k-1} + k(t' - r_j)^{k-1})
\end{aligned}$$

Summing over all pairs yields the inequality. □

Lemma 2 $\sum_t \beta_t \leq (\frac{1}{2} - 2\epsilon) \cdot \mathrm{RR}$.

Proof. From the definiton of β_t, we derive,

$$\begin{aligned}
\sum_t \beta_t &= \sum_{j,t} \beta_{jt} = \sum_j (1 + \delta) F_j \cdot (1/2 - 3\epsilon) F_j^{k-1} \\
&= (1 + \delta)(1/2 - 3\epsilon) \sum_j F_j^k \\
&\leq (1/2 - 2\epsilon) \cdot \mathrm{RR}
\end{aligned}$$

The last inequality follows from the fact that $0 < \delta = \epsilon \leq 1/10$. □

The previous two lemmas show the objective for our setting of the dual variables is $\Omega(\epsilon)$ times RR's kth power flow time. Thus, if we show that the dual constraints are satisfied, this will show that $\Omega(\epsilon)$ times RR's kth power flow time is less than the optimal dual objective. Knowing that the dual objective is at most 2γ times OPT's kth power flow time, we can derive Theorem 1.

3.4 Verifying The Dual Constraints

It now remains to show the dual constraints are satisfied. Clearly the last two sets of constraints are satisfied. Consider the first set of constraints for some fixed job j and time $t \geq r_j$. Define $B(t') := \{j' \mid t' \in [r_{j'}, C_{j'} + \delta F_{j'}]\}$ as the set of jobs j' that contribute to $\beta_{t'}$ by a positive quantity. Note

that $A(t') \subseteq B(t')$, but not necessarily $A(t') = B(t')$. First we bound α_j/p_j.

$$\frac{\alpha_j}{p_j} = \frac{1}{p_j}\left(\sum_{t' \in [r_j, C_j] \cap T_u} k(t-r_j)^{k-1}\right) - \frac{1}{p_j}\epsilon F_j^k \quad (1)$$

$$+ \frac{1}{p_j}\left(\sum_{t' \in [r_j, C_j] \cap T_o} \sum_{j' \in A(t', \le r_j)} \frac{k(t'-r_{j'})^{k-1}}{n_{t'}}\right) \quad (2)$$

It is easy to upper bound Equation (1) that concerns underloaded times:

$$\begin{aligned}
&\frac{1}{p_j}\left(\sum_{t' \in [r_j, C_j] \cap T_u} k(t-r_j)^{k-1}\right) - \frac{1}{p_j}\epsilon F_j^k \\
&\le \frac{1}{p_j}\frac{p_j}{\eta} \cdot k \cdot F_j^{k-1} - \frac{1}{p_j}\epsilon F_j^k \\
&\le \frac{1}{p_j}\left(\epsilon \cdot F_j^k + k\left(\frac{k}{\epsilon}\right)^{k-1} \cdot p_j^k\right) - \frac{1}{p_j}\epsilon F_j^k \\
&\le \frac{\gamma\, p_j^k}{p_j} \qquad (3)
\end{aligned}$$

The first inequality holds due to the fact that job j can be processed by at most p_j and, at each underloaded time, all alive jobs get processed at a rate of $\eta \ge 1$. The last inequality follows considering whether $k \cdot p_j \le \epsilon F_j$ or not.

We shift our attention to upper bounding Equation (2).

$$(2) = \frac{1}{p_j}\left(\sum_{t' \in [r_j, C_j] \cap T_o} \sum_{j' \in A(t', \le r_j) \setminus B(t)} \frac{k(t'-r_{j'})^{k-1}}{n_{t'}}\right) \quad (4)$$

$$+ \frac{1}{p_j}\left(\sum_{t' \in [r_j, C_j] \cap T_o} \sum_{j' \in A(t', \le r_j) \cap B(t)} \frac{k(t'-r_{j'})^{k-1}}{n_{t'}}\right) \quad (5)$$

Note that the two Equations (4) and (5) come from jobs not in $B(t)$ and jobs in $B(t)$, respectively; it is important to note that this partition is based on $B(t)$, not $B(t')$.

We first upper bound Equation (4).

Lemma 3 *For all j and $t \ge r_j$, we have,*

$$\frac{1}{p_j} \sum_{t' \in [r_j, C_j] \cap T_o} \sum_{j' \in A(t', \le r_j) \setminus B(t)} \frac{k(t'-r_{j'})^{k-1}}{n_{t'}} \le \frac{\gamma(t-r_j)^k}{p_j}.$$

Proof. Note that any job j' considered in the summation arrives before job j, is alive at time t', completes before time t, and is not in $B(t)$. Hence we have $r_{j'} \le r_j \le C_{j'} \le C_{j'} + \delta F_{j'} \le t$, which implies that $\delta F_{j'} \le (t - r_j)$. This also implies that we can further restrict t' to the range $[r_j, \min\{t, C_j\}]$ because no job j' contributes to the second summation at time t or later. Thus, we derive,

$$\begin{aligned}
&(LHS) \\
&\le \frac{1}{p_j} \sum_{t' \in [r_j, \min\{t, C_j\}] \cap T_o} \sum_{j' \in A(t', \le r_j) \setminus B(t)} \frac{k(t'-r_{j'})^{k-1}}{n_{t'}} \\
&\le \frac{1}{p_j} \sum_{t' \in [r_j, \min\{t, C_j\}] \cap T_o} \sum_{j' \in A(t', \le r_j) \setminus B(t)} \frac{kF_{j'}^{k-1}}{n_{t'}} \\
&\le \frac{1}{p_j} \sum_{t' \in [r_j, \min\{t, C_j\}] \cap T_o} \sum_{j' \in A(t', \le r_j) \setminus B(t)} \frac{k(1/\delta)^{k-1}(t-r_j)^{k-1}}{n_{t'}} \\
&= \frac{1}{p_j} \cdot k(1/\delta)^{k-1}(t-r_j)^{k-1} \\
&\quad \cdot \sum_{t' \in [r_j, \min\{t, C_j\}] \cap T_o} \sum_{j' \in A(t', \le r_j) \setminus B(t)} \frac{1}{n_{t'}} \\
&\le \frac{1}{p_j} \cdot k(1/\delta)^{k-1}(t-r_j)^k \\
&\le \frac{1}{p_j}\gamma(t-r_j)^k
\end{aligned}$$

The second to last inequality follows from the fact that $|A(t', \le r_j)| \le |A'(t)| \le n_{t'}$. □

We now upper bound Equation (5).

Lemma 4 *For all jobs j and $t \ge r_j$, we have,*

$$\frac{1}{p_j}\left(\sum_{t' \in [r_j, C_j] \cap T_o} \sum_{j' \in A(t', \le r_j) \cap B(t)} \frac{k(t'-r_{j'})^{k-1}}{n_{t'}}\right) \le \frac{\beta_t}{m}.$$

Proof. Consider,

$$\begin{aligned}
\text{(LHS)} &\le \frac{1}{p_j} \sum_{t' \in [r_j, C_j] \cap T_o} \sum_{j' \in A(t') \cap B(t)} \frac{kF_{j'}^{k-1}}{n_{t'}} \\
&\le \frac{1}{p_j} \sum_{j' \in B(t)} \sum_{t' \in [r_{j'}, C_{j'}] \cap [r_j, C_j] \cap T_o} \frac{kF_{j'}^{k-1}}{n_{t'}} \\
&\le \frac{1}{\eta m} \sum_{j' \in B(t)} kF_{j'}^{k-1} \cdot \frac{1}{p_j} \sum_{t' \in [r_{j'}, C_{j'}] \cap [r_j, C_j] \cap T_o} \frac{\eta m}{n_{t'}} \\
&\le \frac{1}{\eta m} \sum_{j' \in B(t)} kF_{j'}^{k-1} \\
&= \frac{k}{(1/2 - 3\epsilon)\eta m} \sum_{j' \in B(t)} (1/2 - 3\epsilon)F_{j'}^{k-1} \\
&= \frac{k}{(1/2 - 3\epsilon)\eta}\frac{\beta_t}{m} \\
&\le \frac{\beta_t}{m} \qquad [\text{Since } \eta := 2k(1+10\epsilon)]
\end{aligned}$$

The second inequality follows from the observation that job j' adds $kF_{j'}^{k-1}/n_{t'}$ to the summation only when jobs j and j' are both alive, and the time step is overloaded. The penultimate inequality follows since while both jobs j and j' are alive, they are processed at an equal rate of $\eta m/n_{t'}$ at each overloaded time t, hence the second summation is upper bounded by job j's size. □

We are now ready to complete the analysis. By combining the previous equations, we have

$$\begin{aligned}
\frac{\alpha_j}{p_j} &\le \frac{1}{p_j}\left(\sum_{t' \in [r_j, C_j] \cap T_u} k(t-r_j)^{k-1}\right) - \frac{1}{p_j}\epsilon F_j^k \\
&\quad + \frac{1}{p_j}\left(\sum_{t' \in [r_j, C_j] \cap T_o} \sum_{j' \in A(t', \le r_j)} \frac{k(t'-r_{j'})^{k-1}}{n_{t'}}\right)
\end{aligned}$$

$$\leq \frac{\gamma\, p_j^k}{p_j} + \frac{\gamma(t-r_j)^k}{p_j} + \frac{\beta_t}{m},$$

proving that the dual constraints are satisfied, as desired.

Acknowledgements. The first author's research was supported in part by NSF grant CCF-1409130. Kulkarni's research was supported by NSF awards CCF-0745761, CCF-1008065, and CCF-1348696.

4. REFERENCES

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

[2] Nir Avrahami and Yossi Azar. Minimizing total flow time and total completion time with immediate dispatching. In *SPAA '03: Proceedings of the fifteenth annual ACM symposium on Parallel algorithms and architectures*, pages 11–18, 2003.

[3] Baruch Awerbuch, Yossi Azar, Stefano Leonardi, and Oded Regev. Minimizing the flow time without migration. *SIAM J. Comput.*, 31(5):1370–1382, 2002.

[4] Nikhil Bansal and Kirk Pruhs. Server scheduling to balance priorities, fairness, and average quality of service. *SIAM Journal on Computing*, 39(7):3311–3335, 2010.

[5] Neal Barcelo, Sungjin Im, Benjamin Moseley, and Kirk Pruhs. Shortest-elapsed-time-first on a multiprocessor. In *Design and Analysis of Algorithms - First Mediterranean Conference on Algorithms, MedAlg 2012, Kibbutz Ein Gedi, Israel, December 3-5, 2012. Proceedings*, pages 82–92, 2012.

[6] Luca Becchetti and Stefano Leonardi. Nonclairvoyant scheduling to minimize the total flow time on single and parallel machines. *J. ACM*, 51(4):517–539, 2004.

[7] Luca Becchetti, Stefano Leonardi, Alberto Marchetti-Spaccamela, and Kirk Pruhs. Online weighted flow time and deadline scheduling. *Journal of Discrete Algorithms*, 4(3):339–352, 2006.

[8] Hemant M. Chaskar and Upamanyu Madhow. Fair scheduling with tunable latency: a round-robin approach. *IEEE/ACM Trans. Netw.*, 11(4):592–601, 2003.

[9] Chandra Chekuri, Ashish Goel, Sanjeev Khanna, and Amit Kumar. Multi-processor scheduling to minimize flow time with epsilon resource augmentation. In *STOC*, pages 363–372, 2004.

[10] Chandra Chekuri, Sanjeev Khanna, and An Zhu. Algorithms for minimizing weighted flow time. In *STOC*, pages 84–93, 2001.

[11] Jeff Edmonds. Scheduling in the dark. *Theor. Comput. Sci.*, 235(1):109–141, 2000.

[12] Jeff Edmonds, Sungjin Im, and Benjamin Moseley. Online scalable scheduling for the ℓ_k-norms of flow time without conservation of work. In *A*, 2011.

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

[14] Kyle Fox and Benjamin Moseley. Online scheduling on identical machines using srpt. In *SODA*, pages 120–128, 2011.

[15] Anupam Gupta, Sungjin Im, Ravishankar Krishnaswamy, Benjamin Moseley, and Kirk Pruhs. Scheduling jobs with varying parallelizability to reduce variance. In *Syposium on Parallel Algorithms and Architectures*, pages 11–20, 2010.

[16] Anupam Gupta, Ravishankar Krishnaswamy, and Kirk Pruhs. Online primal-dual for non-linear optimization with applications to speed scaling. In *WAOA*, 2012.

[17] Ellen L. Hahne. Round-robin scheduling for max-min fairness in data networks. *IEEE Journal on Selected Areas in Communications*, 9(7):1024–1039, 1991.

[18] Sungjin Im, Janardhan Kulkarni, and Kamesh Munagala. Competitive algorithms from competitive equilibria: non-clairvoyant scheduling under polyhedral constraints. In *Symposium on Theory of Computing, STOC 2014, New York, NY, USA, May 31 - June 03, 2014*, pages 313–322, 2014.

[19] 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.

[20] Sungjin Im and Benjamin Moseley. Online scalable algorithm for minimizing ℓ_k-norms of weighted flow time on unrelated machines. In *SODA '11: Proceedings of the Twenty-first Annual ACM -SIAM Symposium on Discrete Algorithms*, pages 95–108, 2011.

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

[22] Bala Kalyanasundaram and Kirk Pruhs. Speed is as powerful as clairvoyance. *Journal of the ACM*, 47(4):617–643, 2000.

[23] Stefano Leonardi and Danny Raz. Approximating total flow time on parallel machines. *J. Comput. Syst. Sci.*, 73(6):875–891, 2007.

[24] Kirk Pruhs, Jiri Sgall, and Eric Torng. *Handbook of Scheduling: Algorithms, Models, and Performance Analysis*, chapter Online Scheduling. 2004.

[25] M. Shreedhar and George Varghese. Efficient fair queueing using deficit round-robin. pages 375–385, 1996.

[26] Abraham Silberschatz, Peter B Galvin, and Greg Gagne. *Operating system concepts*, volume 8. Wiley, 2013.

[27] Eric Torng and Jason McCullough. Srpt optimally utilizes faster machines to minimize flow time. *ACM Transactions on Algorithms*, 5(1), 2008.