Weighted Flowtime on Capacitated Machines

Kyle Fox*
University of Illinois at Urbana-Champaign
Urbana, IL
kylefox2@illinois.edu

Madhukar Korupolu
Google Research
Mountain View, CA
mkar@google.com

Abstract

It is well-known that SRPT is optimal for minimizing flow time on machines that run one job at a time. However, running one job at a time is a big under-utilization for modern systems where sharing, simultaneous execution, and virtualization-enabled consolidation are a common trend to boost utilization. Such machines, used in modern large data centers and clouds, are powerful enough to run multiple jobs/VMs at a time subject to overall CPU, memory, network, and disk capacity constraints.

Motivated by this prominent trend and need, in this work, we give the first scheduling algorithms to minimize weighted flow time on such capacitated machines. To capture the difficulty of the problem, we show that without resource augmentation, no online algorithm can achieve a bounded competitive ratio. We then investigate algorithms with a small resource augmentation in speed and/or capacity. Our first result is a simple $(2+\varepsilon)$-capacity $O(1/\varepsilon)$-competitive greedy algorithm. Using only speed augmentation, we then obtain a 1.75-speed $O(1)$-competitive algorithm. Our main technical result is a *n*ear-optimal $(1+\varepsilon)$-speed, $(1+\varepsilon)$-capacity $O(1/\varepsilon^3)$-competitive algorithm using a novel combination of knapsacks, densities, job classification into categories, and potential function methods. We show that our results also extend to the multiple unrelated capacitated machines setting.

1 Introduction

Four trends are becoming increasingly prominent in modern data centers. Modern machines are increasingly powerful and able to run multiple jobs at the same time subject to overall capacity constraints. Running only one job at a time on them leads to significant under-utilization. Virtualization technologies such as VMware [3] and Xen [13] are enabling workloads to be packaged as VMs and multiple VMs run on the same physical machine [5]. This packing of VMs on the same machine enables server consolidation [6, 7, 15] and data center cost reduction in the form of reduced hardware, floorspace, energy, cooling, and maintenance costs. This utilization boost and the resulting cost savings has been a driving force for virtualization adoption and transformation of IT in data centers.

This adoption, in turn, is driving a proliferation of compute farms and public, private, and hybrid clouds which share resources and machines among multiple users and jobs [1, 2, 4, 32]. Large compute farms and clouds often have excess capacity provisioned for peak workloads and these can be shared with new users to improve utilization during off-peak times. Tasks such as MapReduce [21] which are I/O driven, are often run with multiple instances (of different MapReduces) on the same machine so they all progress toward completion simultaneously. A characterization of workloads in Google compute clusters is given in [20, 32, 34].

A common scenario in all of these situations is that of jobs (or VMs) arriving in an online manner with their resource requirements and processing time requirements and a scheduler scheduling them on one of the machines in the cluster. Jobs leave when they are completed. The problem is interesting even with a single machine. A common goal in such a scenario is that of minimizing average weighted flow time, defined as the weighted average of the job completion times minus arrival times, i.e., the total time spent by jobs in the system. Other common goals include load-balancing (minimizing maximum load on machines), minimizing average stretch (stretch of a job defined as completion time minus arrival time divided by processing time), or completing as many jobs as possible before their deadlines if there are deadlines. Adding weights lets us prioritize the completion of certain jobs over others. In fact, weighted flow time subsumes the stretch metric if we set the weight of a job to be the reciprocal of its processing time. In this paper we focus on the weighted flow time objective.

1.1 Prior work on minimizing flow time: Single job at a time model.

Traditionally, almost all the known flow time results have been for machines that run at most one job at a time. Consider the following model. A server sees n jobs arrive over time. Each job j arrives at time r_j

*Portions of this work were done while this author was visiting Google Research.

and requires a processing time p_j. At each point in time t, the server may choose to schedule one job j that has already arrived ($r_j \leq t$). We say a job j has been completed when it has been scheduled for p_j time units. Preemption is allowed, so job j may be scheduled, paused, and then later scheduled without any loss of progress. Each job j is given a weight w_j. The goal of the scheduler is to minimize the total weighted flow time of the system, defined as $\sum_j w_j(C_j - r_j)$ where C_j is the completion time of job j. For this paper in particular, we are interested in the clairvoyant *online* version of this problem where the server is not aware of any job j until its release, but once job j arrives, the server knows p_j and w_j.

It is well known that Shortest Remaining Processing Time (SRPT) is the optimal algorithm for minimizing flow time with unit weights in the above single-job machine model. In fact, SRPT is the best known algorithm for unit weights on multiple machines in both the online and offline setting. Leonardi and Raz [30] show SRPT has competitive/approximation ratio $O(\log(\min\{n/m, P\}))$ where m is the number of machines and P is the ratio of the longest processing time to shortest. They also give a matching lower bound for any randomized online algorithm. Chekuri, Khanna, and Zhu [19] show every randomized online algorithm has competitive ratio $\Omega(\min\{W^{1/2}, P^{1/2}, (n/m)^{1/4}\})$ when the jobs are given arbitrary weights and there are multiple machines. Bansal and Chan [11] partially extended this result to show there is no *deterministic* online algorithm with constant competitive ratio when the jobs are given arbitrary weights, even with a *single* machine. However, there are some positive online results known for a single machine where the competitive ratio includes a function of the processing times and weights [12, 19][1].

The strong lower bounds presented above motivate the use of resource augmentation, originally proposed by Kalyanasundaram and Pruhs [28], which gives an algorithm a fighting chance on worst case scheduling instances when compared to an optimal offline algorithm. We say an algorithm is s-speed c-competitive if for any input instance I, the algorithm's weighted flow time while running at speed s is at most c times the optimal schedule's weighted flow time while running at unit speed. Ideally, we would like an algorithm to be constant factor competitive when given $(1 + \varepsilon)$ times a resource over the optimal offline algorithm for any constant $\varepsilon > 0$. We call such algorithms [resource]-scalable. Chekuri *et al.* [18] gave the first speed-scalable online algorithms for minimizing flow time on multiple machines with unit weights. Becchetti *et al.* [14] gave the first analysis that the simple greedy algorithm Highest Density First (HDF) is speed-scalable with arbitrary job weights on a single machine. HDF always schedules the alive job with the highest weight to processing time ratio. Bussema and Torng [16] showed the natural generalization of HDF to multiple machines is also speed-scalable with arbitrary job weights. Table 1 gives a summary of these results.

[1]The result of [19] is actually a *semi-online* algorithm in that P is known in advance.

1.2 The capacitated flow time problem. The above results assume that any machine can run at most one job at a time. This assumption is wasteful and does not take advantage of the sharing, simultaneous execution, and virtualization abilities of modern computers. The current trend in industry is not only to focus on distributing large workloads, but also to run multiple jobs/VMs at the same time on individual machines subject to overall capacity constraints. By not addressing the sharing and virtualization trend, theory is getting out of touch with the industry.

The problem of scheduling an online sequence of jobs (VMs) in such scenarios necessitates the "capacitated machines" variant of the flow time problem. In addition to each job j having a processing time and weight, job j also has a *height* h_j that represents how much of a given resource (e.g. RAM, network throughput) the job requires to be scheduled. The server is given a capacity that we assume is 1 without loss of generality. At each point in time, the server may schedule an arbitrary number of jobs such that the sum of heights for jobs scheduled is at most the capacity of the server. We assume all jobs have height at most 1, or some jobs could never be scheduled. The setting can be easily extended to use multiple machines and multi-dimensional heights, but even the single-machine, single-dimension case given above is challenging from a technical standpoint. We focus on the single-machine, single-dimension problem first, and later show how our results can be extended to multiple machines. We leave further handling of multiple dimensions as an interesting open problem.

The case of jobs having sizes related to machine capacities has been studied before in the context of load-balancing [9, 10] and in operations research and systems [6, 15, 31, 33]. However, this paper is the first instance we are aware of that studies capacitated machines with the objective of minimizing flow time. We call this the Capacitated Flow Time (CFT) problem. As flow time is a fundamental scheduling

Capacitated	Weights	Machines	Competitive Ratio
no	unit	single	1 [folklore]
no	unit	multiple	$\Omega(\log n/m + \log P)$ [30] $O(\log \min\{n/m, P\})$ [30] $(1+\varepsilon)$-speed $O(1/\varepsilon)$-comp. [18]
no	arbitrary	single	$\omega(1)$ [11] $O(\log^2 P)$ [19, (semi-online)] $O(\log W)$ [12] $O(\log n + \log P)$ [12] $(1+\varepsilon)$-speed $O(1/\varepsilon)$-comp. [14]
no	arbitrary	multiple	$\Omega\left(\min\left\{W^{1/2}, P^{1/2}, (n/m)^{1/4}\right\}\right)$ [19] $(1+\varepsilon)$-speed $O(1/\varepsilon^2)$-comp. [16]
yes	unit	single	$(1.2-\varepsilon)$-capacity $\Omega(P)$ [this paper] $(1.5-\varepsilon)$-capacity $\Omega(\log n + \log P)$ [this paper]
yes	weighted	single	$(2-\varepsilon)$-capacity $\omega(1)$ [this paper]
yes	weighted	multiple	$(2+\varepsilon)$-capacity $O(1/\varepsilon)$-comp. [this paper] 1.75-speed $O(1)$-comp. [this paper] $(1+\varepsilon)$-speed $(1+\varepsilon)$-capacity $O(1/\varepsilon^3)$-comp. [this paper]

Table 1. A summary of prior and new results in online scheduling to minimize flowtime. Here, W denotes the number of distinct weights given to jobs, and P denotes the ratio of the largest to least processing time.

objective, we believe this is an important problem to study and bridge the gap to practice.

1.3 Our results and outline of the paper. In Section 2, we show that online scheduling in the capacitated machines case is much more difficult than scheduling in the single-job model. In particular, Theorem 2.1 shows there exists no online randomized algorithm for the CFT problem with a bounded competitive ratio, even when all jobs have unit weight. This result stands in stark contrast to the same situation where SRPT is actually an optimal algorithm for the single job at a time model.

Motivated by the difficulty of online scheduling, we consider the use of resource augmentation in speed and capacity. In a capacity augmentation analysis, an algorithm is given capacity greater than 1 while being compared to the optimal schedule on a unit capacity server. We define u-capacity c-competitive accordingly. We also say an algorithm is s-speed u-capacity c-competitive if it is c-competitive when given *both* s-speed and u-capacity over the optimal unit resource schedule. We show in Corollary 2.2 and Theorem 2.3 that our lower bound holds even when an algorithm is given a small constant factor extra capacity without additional speed. In fact, strong lower bounds continue to exist even as the augmented capacity approaches 2.

In Section 3, we propose and analyze a greedy algorithm Highest Residual Density First (HRDF) for the CFT problem and show it is $(2+\varepsilon)$-capacity $O(1/\varepsilon)$-competitive for any small constant $\varepsilon > 0$. HRDF uses greedy knapsack packing with job residual densities (defined later). The algorithm and analysis are tight for capacity-only augmentation given the lower bound in Theorem 2.4. We then turn to the question of whether speed augmentation helps beat the $(2-\varepsilon)$-capacity lower bound. Our initial attempt and result (not described in this paper) was an s-speed u-capacity constant-competitive algorithm for all s and u with $s \cdot u \geq 2+\varepsilon$. We subsequently obtained an algorithm High Priority Category Scheduling (HPCS) which breaks through the 2 bound. HPCS is 1.75-speed $O(1)$-competitive and is described in Section 4.

Our main result is a $(1+\varepsilon)$-speed $(1+\varepsilon)$-capacity $O(1/\varepsilon^3)$-competitive algorithm called Knapsack Category Scheduling (KCS). KCS builds on HPCS and is presented in Section 5. The KCS result shows the power of combining both speed and capacity augmentation and is the first of its kind.[2] Pseudocode descriptions of all our algorithms are given in Figures 1, 3, and 4. We conclude with a brief summary of our results as well as some open problems related to the CFT problem in Section 6.

All of our results can be extended to the multiple unrelated capacitated machine setting as detailed in Appendix D. In the unrelated machines model, each machine-job pair is given a weight, processing time, and height unrelated to any other machine-job pairs.

[2] In practice, machines are rarely filled to full capacity, so small augmentations in capacity are not unreasonable.

Technical novelty: Our algorithms use novel combinations and generalizations of knapsacks, job densities, and grouping of jobs into categories. While HRDF uses knapsacks and job densities and HPCS uses job densities and categories, KCS combines all three. In fact, our algorithms are the first we are aware of that do not make flow time scheduling decisions based on a single ordering of jobs. Our analysis methods combining potential functions, knapsacks, and categories are the first of their kind and may have other applications.

2 Lower Bounds

We begin the evaluation of the CFT problem by giving lower bounds on the competitive ratio for any online scheduling algorithm. Some lower bounds hold even when jobs are given unit weights. Recall P denotes the ratio of the largest processing time to the least processing time.

Theorem 2.1. *Any randomized online algorithm $\mathcal{A}$ for the CFT problem has competitive ratio $\Omega(P)$ even when all jobs have unit weight.*

Proof: Garg and Kumar [24] consider a variant on the single-job capacity identical machines model they call Parallel Scheduling with Subset Constraints. Our proof very closely follows the proof of [24, Theorem 3.2]. While our proof is a lower bound on any *deterministic* algorithm, it easily generalizes to randomized algorithms with oblivious adversaries. Further, the proof continues to hold even with small amounts of capacity augmentation. Fix a deterministic online algorithm $\mathcal{A}$. We use only one machine and all jobs have unit weight. There are two types of jobs: Type I jobs have unit processing time and height 0.4; Type II jobs have processing time $P \gg 1$ and height 0.6. Let $T \gg P$ be a large constant. On each time step $t = 0, \ldots, T-1$, we release two jobs of type I. At each time step $t = 0, P, 2P, \ldots, T-1$, we release one job of type II (we assume $T-1$ is a multiple of P). The total processing time of all released jobs is $3T$. The server can schedule two type I jobs at once, and one type I job with a type II job, but it cannot schedule two type II jobs at once. Therefore, one of the following must occur: (a) at least $T/2$ jobs of type I remain alive at time T, or (b) at least $T/2 \cdot P$ jobs of type II remain alive at time T.

Suppose case (a) occurs. We release 2 jobs of type I at each time step $t = T, \ldots, L, L \gg T$. The total flow time of $\mathcal{A}$ is $\Omega(T \cdot L)$. However, the optimal schedule finishes all jobs of type I during $[0, T]$. It has only T/P jobs (of type II) waiting at each time step during $[T, L]$, so the optimal flow time is $O(T \cdot L/P)$.

Suppose case (b) occurs. We release one job of type II at each time step $t = T, T+P, T+2P, \ldots, L, L \gg T$. There are always $T/2 \cdot P$ jobs alive in $\mathcal{A}$'s schedule during $[T, L]$. So $\mathcal{A}$'s total flow time is $\Omega(T \cdot L/P)$. The optimal schedule finishes all jobs of type II during $[0, T]$. It has $2T$ jobs of type I alive at time T, but it finishes these jobs during $[T, 3T]$ while also processing jobs of type II. So, after time $3T$, the optimal schedule has no more alive jobs and its total flow time is $O(L)$. We get the theorem by setting $T \geq P^2$. □

Corollary 2.2. *Any randomized online algorithm $\mathcal{A}$ for the CFT problem is $(1.2-\varepsilon)$-capacity $\Omega(P)$-competitive for any constant $\varepsilon > 0$ even when all jobs have unit weight.*

In fact, superconstant lower bounds continue to exist even when an online algorithm's server is given more than 1.2 capacity. When we stop requiring input instances to have unit weights, then the lower bounds become resilient to even larger capacity augmentations.

Theorem 2.3. *Any randomized online algorithm $\mathcal{A}$ for the CFT problem is $(1.5-\varepsilon)$-capacity $\Omega(\log n + \log P)$-competitive for any constant $\varepsilon > 0$ even when all jobs have unit weight.*

Proof: There is a simple reduction from online scheduling in the single-job identical machines model to the capacitated server model. Let I be any input instance for the single-job identical machines model where there are two machines and all jobs have unit weight. We create an input instance I' using the same jobs as I with each job given height 0.5. Any schedule of I' is a schedule for I. The lower bound of [30, Theorem 9] immediately implies the theorem. □

Theorem 2.4. *No deterministic online algorithm $\mathcal{A}$ for the CFT problem is constant competitive even when given $(2-\varepsilon)$-capacity for any constant $\varepsilon > 0$ assuming jobs are allowed arbitrary weights.*

Proof: We apply a similar reduction to that given in the proof of Theorem 2.3. Let I be any input instance for the single-job identical machines model where there is one machine and jobs have arbitrary weight. We create an input instance I' using the same jobs as I with each job given height 1. Any schedule of I' is a schedule for I. The lower bound of [11, Theorem 2.1] immediately implies the theorem. □

HRDF(I, t):
/* $Q^A(t)$: Jobs alive at time t. $p_j^A(t)$: Remaining processing time of j */
1. For each $j \in Q^A(t)$: $d_j^A(t) \leftarrow w_j/(h_j p_j^A(t))$ /* Assign residual densities */

2. Sort $Q^A(t)$ by $d_j^A(t)$ descending and pack jobs from $Q^A(t)$ up to capacity

Figure 1. The algorithm HRDF.

3 Analysis Warmup: Highest Residual Density First

In this section, we analyze the deterministic online algorithm Highest Residual Density First (HRDF) for the CFT problem. A description of HRDF appears in Figure 1. Despite the simplicity of the algorithm, we are able to show the following result matching the lower bound of Theorem 2.4.

Theorem 3.1. *HRDF is $(2+\varepsilon)$-capacity $O(1/\varepsilon)$-competitive for any constant ε where $0 < \varepsilon < 1$.*

3.1 Analysis of HRDF. We use a potential function analysis to prove Theorem 3.1. Set an input instance I, and consider running HRDF with $2+\varepsilon$ capacity alongside the optimal schedule without resource augmentation. We begin our analysis by defining some notation. For any job j let C_j^A be the completion time of j in HRDF's schedule and let C_j^O be the completion time of j in the optimal schedule. For any time $t \geq r_j$, let $A_j(t) = \min\{C_j^A, t\} - r_j$ be the accumulated weighted flow time of job j at time t and let $A_j = A_j(C_j^A)$. Finally, let $A = \sum_j A_j$ be the total weighted flow time for HRDF's schedule. Let $\mathrm{OPT}_j(t)$, OPT_j, and OPT be defined similarly for the optimal schedule.

Our analysis of HRDF builds on the potential function approach outlined in [27], specifically [17, 22, 23], for the single-job at a time problem and extends it to the capacitated case. Unlike in [22,23], our method supports arbitrary job weights and capacities, and unlike in [17], we do not use the notion of *fractional flow time*.[3] We give a potential function Φ that is almost everywhere differentiable such that $\Phi(0) = \Phi(\infty) = 0$ and then focus on bounding all continuous and discrete increases to $A + \Phi$ across time by a function of OPT in order to bound A by a function of OPT.[4]

We give the remaining notation for our potential function. Let $p_j^O(t)$ be the remaining processing time of job j at time t in the optimal schedule. Let $Q^O(t)$ be the set of alive jobs in the optimal schedule at time t. Let $Q^{Aj}(t) = \{k \in Q^A(t) : d_k^A(t) > d_j^A(t)\}$ be the set of alive jobs in HRDF's schedule more dense than job j. Let $S^A(t)$ and $S^O(t)$ be the set of jobs processed at time t by HRDF's and the optimal schedule respectively. Finally, let $W^A(t) = \sum_{j \in Q^A(t)} w_j$ be the total weight of all jobs alive in HRDF's schedule at time t. The rest of the notation we use in our potential function is given in Figure 1 and Table 2.

The potential function we use has two terms for each job j:

$$\Phi_j^A(t) = \frac{w_j}{\varepsilon}\big(R_j(t) + (1+\varepsilon)p_j^A(t) - V_j^{>}(t) + Y_j(t) + F_j(t)\big)$$
$$\Phi_j^O(t) = \frac{w_j}{\varepsilon} R_j^{<}(t)$$

The potential function is defined as

$$\Phi(t) = \sum_{j \in Q^A(t)} \Phi_j^A(t) - \sum_{j \in Q^O(t)} \Phi_j^O(t).$$

3.2 Main proof idea: Using extra capacity to keep up with the optimal. We give full details of the potential function analysis in Appendix A including the effects from job arrivals and completions, but briefly mention here some of the continuous changes made to $A + \Phi$ due to processing from HRDF and the optimal schedule. Intuitively, these changes compare $S^A(t)$ and $S^O(t)$, the jobs processed at time t by HRDF's and the optimal schedule respectively. As is the case for many potential function arguments, we must show that HRDF removes the remaining area of jobs more efficiently than the optimal schedule. The extra efficiency shows Φ falls fast enough to counteract changes in A. Showing an algorithm's schedule removes area more efficiently than the optimal schedule is often the most difficult part of a potential

[3] Avoiding the use of fractional flow time allows us to analyze HRDF using capacity-only augmentation without speed augmentation [17]. It also saves us a factor of $\Omega(1/\varepsilon)$ in the competitive ratio for KCS.

[4] An analysis of an HRDF-like algorithm for the unrelated single-job machine problem was given in [8] by relaxing an integer program and applying dual fitting techniques. We focus on potential function arguments in this work as the integrality gap of the natural linear program for our setting is unbounded for the small amounts of resource augmention we use in later algorithms.

Term	Definition	Explanation
$R_j(t)$	$\sum_{k \in Q^{A_j}(t)} h_k p_k^A(t)$	Total remaining area of jobs more dense than j. Continuous decreases in R_j counteract continuous increases in A.
$R_j^<(t)$	$\sum_{k \in Q^{A_j}(r_j)} h_k p_k^A(t)$	Total remaining area of jobs more dense than j *upon j's arrival*. The decrease from $R_j^<$ cancels the increase to Φ caused by R_j when j arrives.
$V_j^>(t)$	$\sum_{k \in Q^O(t) \cap Q^{A_j}(r_k)} h_k p_k^O(t)$	The optimal schedule's total remaining area for each job k more dense than j upon k's arrival. The increase to R_j caused by the arrival of a job k is canceled by an equal increase in $V_j^>$.
$Y_j(t)$	$Y_j(t) = 0$ for all $t < r_j$ and $\frac{d}{dt}Y_j(t) = \sum_{k \in (S^O(t) \setminus Q^{A_j}(r_k))} h_k$ for all $t \geq r_j$	Captures work done by the optimal schedule that does not affect $V_j^>$. Used to bound increases in Φ from the removal of $R_j^<$ terms when jobs are completed.
$F_j(t)$	$F_j(t) = 0$ for all $t < r_j$, $\frac{d}{dt}F_j(t) = 0$ if $j \notin S^A(t)$, and $\frac{d}{dt}F_j(t) = h_j \sum_{k \in Q^O(t)} \frac{w_k}{w_j}$ otherwise.	Captures the increase in flow time by the optimal schedule while our schedule processes j. Used to bound increases in Φ from the removal of V_j terms when jobs are completed.

Table 2. A summary of notation used in our potential functions.

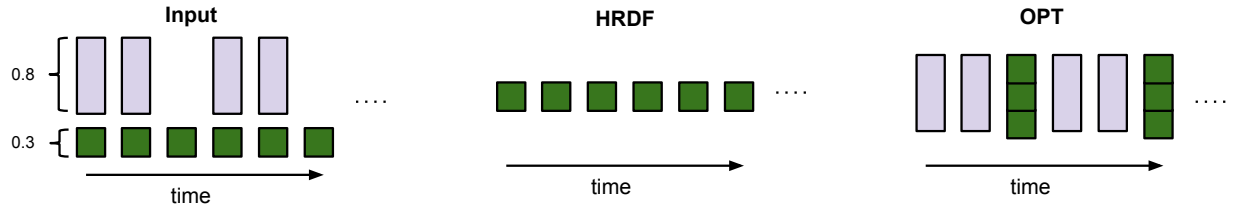

Figure 2. A lower bound example for HRDF. Type I jobs appear every time step as green squares with height 0.3 and weight 2 while type II jobs appear at time steps $\{0, 1\} \mod 3$ as purple rectangles with height 0.8 and weight 1. The process continues until time $T \gg 1$. Left: The input set of jobs. Center: As type I jobs have higher residual density, HRDF picks one at each time step as it arrives. Right: Optimal schedule uses type II jobs at time steps $\{0, 1\} \mod 3$ and three type I jobs at time steps $2 \mod 3$.

function analysis and greatly motivates the other algorithms given in this paper. In our potential function, rate of work is primarily captured in the R_j, p_j^A, V_j, and Y_j terms.

Lemma 3.2. *If HRDF's schedule uses $2+\varepsilon$ capacity and the optimal schedule uses unit capacity, then*

$$\sum_{j \in Q^A(t)} \frac{d}{dt}\left(\Phi_j^A - \frac{w_j}{\varepsilon}F_j\right)(t) = \sum_{j \in Q^A(t)} \frac{d}{dt}\frac{w_j}{\varepsilon}(R_j + (1+\varepsilon)p_j^A - V_j^> + Y_j)(t) \leq -W^A(t).$$

Proof: Consider any time t. For any job j, neither R_j nor $(1+\varepsilon)p_j^A$ can increase due to continuous processing of jobs. If HRDF's schedule processes j at time t, then $\frac{d}{dt}(1+\varepsilon)p_j^A(t) = -(1+\varepsilon)$. If HRDF's schedule does not process j, then HRDF's schedule processes a set of more dense jobs of height at least $(1+\varepsilon)$. Otherwise, there would be room to process j. Therefore, $\frac{d}{dt}R_j(t) \leq -(1+\varepsilon)$ and

$$\sum_{j \in Q^A(t)} \frac{d}{dt}\frac{w_j}{\varepsilon}(R_j + (1+\varepsilon)p_j^A)(t) \leq -\frac{1+\varepsilon}{\varepsilon}W^A(t).$$

The optimal schedule is limited to scheduling jobs with a total height of 1. No job contributes to both $V_j^>$ and Y_j, so

$$\sum_{j \in Q^A(t)} \frac{d}{dt}\frac{w_j}{\varepsilon}(-V_j^> + Y_j)(t) \leq \frac{1}{\varepsilon}W^A(t).$$

□

4 Beyond Greedy: High Priority Category Scheduling

The most natural question after seeing the above analysis is whether there exists an algorithm that is constant factor competitive when given only speed augmentation. In particular, a $(1+\varepsilon)$-speed constant competitive algorithm for any small constant ε would be ideal.

HPCS(I, t, κ):
/* κ is an integer determined in analysis */
1. For each $1 \le i < \kappa$: /* Divide jobs into categories by height */
$Q_i^A(t) \leftarrow \{j \in Q^A(t) : 1/(i+1) < h_j \le 1/i\}$ /* Assign alive category i jobs */
$W_i(t) \leftarrow \sum_{j \in Q_i^A(t)} w_j$ /* Assign queued weight of category i */
For each $j \in Q_i^A(t)$: $h'_j \leftarrow 1/i$ /* Assign modified height of job j */

$Q_X^A(t) \leftarrow \{j \in Q^A(t) : h_j \le 1/\kappa\}$ /* X is the small category */
$W_X(t) \leftarrow \sum_{j \in Q_X^A(t)} w_j$
For each $j \in Q_X^A(t)$: $h'_j \leftarrow h_j$

2. For each $j \in Q^A(t)$: $d_j^A(t) \leftarrow w_j/(h'_j p_j^A(t))$ /* Assign densities by modified height */

3. $i^* \leftarrow$ highest queued weight category
Sort $Q_{i^*}^A(t)$ by $d_j^A(t)$ descending and pack jobs from $Q_{i^*}^A(t)$ up to capacity

Figure 3. The algorithms HPCS.

Unfortunately, the input sequence given in Figure 2 shows that greedily scheduling by residual density does not suffice. HRDF's schedule will accumulate two type II jobs every three time steps, giving it a total flow time of $\Omega(T^2)$ by time T. The optimal schedule will have flow time $O(T)$, giving HRDF a competitive ratio of $\Omega(T)$. Even if we change HRDF to perform an *optimal* knapsack packing by density or add some small constant amount of extra speed, the lower bound still holds.

4.1 Categorizing jobs. The above example shows that greedily scheduling the high value jobs and ignoring others does not do well. In fact, type II jobs, though lower in value individually, cannot be allowed to build up in the queue forever. To address this issue, we use the notion of categories and group jobs of similar heights together. The intuition is that once a category or group of jobs accumulates enough weight, it becomes important and should be scheduled. The algorithm HPCS using this intuition is given in Figure 3. Note that HPCS schedules from one category at a time and packs as many jobs as possible from that category.

Within the category HPCS chooses, we can bound the performance of HPCS compared to the optimal. The only slack then is the ability of the optimal schedule to take jobs from multiple categories at once. However, using a surprising application of a well known bound for knapsacks, we show that the optimal schedule cannot affect the potential function too much by scheduling jobs from multiple categories. In fact, we are able to show HPCS is constant competitive with strictly less than double speed.[5]

[5]We note that we can achieve constant competitiveness with speed strictly less than 1.75, but we do not have a closed form expression for the minimum speed possible using our techniques.

Theorem 4.1. *HPCS is 1.75-speed $O(1)$-competitive.*

4.2 Analysis of HPCS. We use a potential function argument to show the competitiveness of HPCS. The main change in our proof is that the terms in the potential function only consider jobs within a single category at a time. For any fixed input instance I, consider running HPCS's schedule at speed 1.75 alongside the optimal schedule at unit speed. For a non-small category i job j, let $Q_i^{Aj}(t) = \{k \in Q_i^A(t) : d_k^A(t) > d_j^A(t)\}$ be the set of alive category i jobs more dense than job j. Define $Q_X^{Aj}(t)$ similarly. We use the notation in Figure 3 along with the notation given in the proof for HRDF, but we use the modified height in every case height is used. Further, we restrict the definitions of R_j, $R_j^<$, $V_j^>$, Y_j, and F_j to include only jobs in the same category as j. For example, if job j belongs to category i then $R_j(t) = \sum_{k \in Q_i^{Aj}(t)} h'_k p_k^A(t)$.

Let λ be a constant we set later. The potential function we use has two terms for each job j.

$$\Phi_j^A(t) = \lambda \kappa w_j \big(R_j(t) + p_j^A(t) - V_j^>(t) + Y_j(t) + F_j(t)\big)$$
$$\Phi_j^O(t) = \lambda \kappa w_j R_j^<(t)$$

The potential function is again defined as

$$\Phi(t) = \sum_{j \in Q^A(t)} \Phi_j^A(t) - \sum_{j \in Q^O(t)} \Phi_j^O(t).$$

4.3 Main proof idea: Knapsack upper bounds on the optimal efficiency. As before, we defer the full analysis for Appendix B, but we mention here the same types of changes mentioned in Section 3. Our goal is to show the optimal schedule cannot perform too much work compared to HPCS's schedule even if the optimal schedule processes jobs from several categories at once. Our proof uses a well known upper bound argument for knapsack packing. Consider any time t and let $W^*(t)$ be the queued weight of the category scheduled by HPCS at time t.

Lemma 4.2. *If HPCS's schedule uses* 1.75 *speed and the optimal schedule uses unit speed, then for all* $\kappa \geq 200$ *and* $\lambda \geq 100$

$$\sum_{j \in Q^A(t)} \frac{d}{dt}\left(\Phi_j^A(t) - \lambda\kappa w_j F_j\right)(t) = \sum_{j \in Q^A(t)} \frac{d}{dt}\lambda\kappa w_j (R_j + p_j^A - V_j + Y_j)(t) \leq -\kappa W^*(t).$$

There are κ categories, so this fall counteracts the increase in A.

Proof: Suppose HPCS schedules jobs from small category X. Let j be some job in category X. If HPCS's schedule processes j at time t, then $\frac{d}{dt}p_j^A(t) = -1.75$. If HPCS's schedule does not process j at time t, then it does process jobs from category X with height at least $1 - 1/\kappa$. Therefore, $\frac{d}{dt}R_j^X(t) \leq -1.75(1 - 1/\kappa)$. Similar bounds hold if HPCS's schedule processes jobs from non-small category i, so

$$\sum_{j \in Q^A(t)} \frac{d}{dt}\lambda\kappa w_j (R_j + p_j^A)(t) \leq -1.75\lambda\left(1 - \frac{1}{\kappa}\right)\kappa W^*(t).$$

Bounding the increases to Φ from changes to the V_j and Y_j terms takes more subtlety than before. We pessimistically assume every category has queued weight $W^*(t)$. We also imagine that for each non-small category i job j processed by the optimal schedule, job j has height $1/(i+1)$. The optimal schedule can still process the same set of jobs with these assumptions, and the changes due to V_j and Y_j terms only increase with these assumptions.

We now model the changes due to the V_j and Y_j terms as an instance K to the standard knapsack packing problem. For each non-small category i, there are i items in K each with value $\lambda\kappa W^*(t)/i$ and size $1/(i+1)$. These items represent the optimal schedule being able to process i jobs at once from category i. Knapsack instance K also contains one item for each category X job j alive in the optimal schedule with value $\lambda\kappa W^*(t)h_j$ and size h_j. The capacity for K is 1. A valid solution to K is a set U of items of total size at most 1. The value of a solution U is the sum of values for items in U. The optimal solution to K has a value at least as large as the largest possible increase to Φ due to changes in the V_j and Y_j terms. Note that we cannot pack an item of size 1 and two items of size $1/2$ in K. Define K_1 and K_2 to be K with one item of size 1 removed and one item of size $1/2$ removed respectively. The larger of the optimal solutions to K_1 and K_2 is the optimal solution to K.

Let the knapsack density of an item be the ratio of its value to its size. Let $v_1, v_2, \ldots$ be the values of the items in K in descending order of knapsack density. Let $s_1, s_2, \ldots$ be the sizes of these same items. Greedily pack the items in decreasing order of knapsack density and let k be the index of the first item that does not fit. Folklore states that $v_1 + v_2 + \cdots + \alpha v_k$ is an upper bound on the optimal solution to K where $\alpha = \frac{1-(s_1+s_2+\cdots+s_{k-1})}{s_k}$. This fact applies to K_1 and K_2 as well.

In all three knapsack instances described, ordering the items by size gives their order by knapsack density. Indeed, items of size $1/i$ have knapsack density $\lambda\kappa W^*(t)(i+1)/i$. Items created from category X jobs have knapsack density $\lambda\kappa W^*(t)$. We may now easily verify that $1.45\lambda\kappa W^*(t)$ and $1.73\lambda\kappa W^*(t)$ are upper bounds for the optimal solutions to K_1 and K_2 respectively. Therefore,

$$\sum_{i \in Q^A(t)} \frac{d}{dt}\lambda\kappa w_j (-V_j + Y_j)(t) \leq 1.73\lambda\kappa W^*(t).$$

□

5 Main Algorithm: Knapsack Category Scheduling

As stated in the previous section, a limitation of HPCS is that it chooses jobs from one category at a time, putting it at a disadvantage. In this section, we propose and analyze a new algorithm called Knapsack Category Scheduling (KCS) which uses a fully polynomial-time approximation scheme (FPTAS) for the knapsack problem [25,29] to choose jobs from multiple categories. Categories still play

KCS(I, t, ε):
/* ε is any real number with $0 < \varepsilon \leq 1/2$ */
1. For each integer $i \geq 0$ with $(1+\varepsilon/2)^i < 2/\varepsilon$ /* Divide jobs into categories by height */
$Q_i^A(t) \leftarrow \{j \in Q^A(t) : 1/(1+\varepsilon/2)^{i+1} < h_j \leq 1/(1+\varepsilon/2)^i\}$
$W_i(t) \leftarrow \sum_{j \in Q_i^A(t)} w_j$
For each $j \in Q_i^A(t)$: $h'_j \leftarrow 1/(1+\varepsilon/2)^i$
$Q_X^A(t) \leftarrow Q^A(t) \setminus \cup_i Q_i^A(t)$ /* Category X jobs have height at most $\varepsilon/2$ */
$W_X(t) \leftarrow \sum_{j \in Q_X^A(t)} w_j$
For each $j \in Q_X^A(t)$: $h'_j \leftarrow h_j$

2. For each $j \in Q^A(t)$: $d_j^A(t) \leftarrow w_j/(h'_j p_j^A(t))$ /* Assign densities by modified height */

3. Initialize knapsack $K(t)$ with capacity $(1+\varepsilon)$/* Setup a knapsack instance to choose jobs */
For each category i and category X; for each $j \in Q_i^A(t)$
$Q_i^{Aj}(t) \leftarrow \left\{k \in Q_i^A(t) : d_k^A(t) > d_j^A(t)\right\}$ /* Assign category i jobs more dense than j */
Add item j to $K(t)$ with size h_j and value $w_j + h'_j \sum_{k \in (Q_i^A(t) \setminus Q_i^{Aj}(t) \setminus \{j\})} w_k$

4. Pack jobs from a $(1-\varepsilon/2)$-approximation for $K(t)$

Figure 4. The algorithm KCS.

a key role in KCS, as they help assign values to jobs when making scheduling decisions.

Detailed pseudocode for KCS is given in Figure 4. Steps 1 and 2 of KCS are similar to those of HPCS, except that KCS uses geometrically decreasing heights for job categories.[6] Another interesting aspect of KCS is the choice of values for jobs in step 4. Intuitively, the value for each knapsack item j is chosen to reflect the total weight of less dense jobs waiting on j's completion in a greedy scheduling of j's category. More practically, the values are chosen so the optimal knapsack solution will affect the potential function Φ as much as possible.

KCS combines the best of all three techniques: knapsack, densities, and categories, and we are able to show the following surprising result illustrating the power of combining speed and capacity augmentation.

Theorem 5.1. *KCS is* $(1+\varepsilon)$*-speed* $(1+\varepsilon)$*-capacity* $O(1/\varepsilon^3)$*-competitive.*

5.1 Main proof idea: Combining categories optimally. We use a potential function analysis to prove Theorem 5.1. Fix an input instance I and consider running KCS's schedule with $(1+\varepsilon)$ speed and capacity alongside the optimal schedule with unit resources. We continue to use the same notation used for HPCS, but modified when necessary for KCS. Let κ be the number of categories used by KCS. Observe

$$\kappa \leq 2 + \log_{1+\varepsilon/2}(2/\varepsilon) = O(1/\varepsilon^2).$$

The potential function has two terms for each job j.

$$\Phi_j^A(t) = \frac{8\kappa w_j}{\varepsilon}\big(R_j(t) + p_j^A(t) - V_j^{>}(t) + Y_j(t) + F_j(t)\big)$$
$$\Phi_j^O(t) = \frac{8\kappa w_j}{\varepsilon} R_j^{<}(t)$$

The potential function is defined as

$$\Phi(t) = \sum_{j \in Q^A(t)} \Phi_j^A(t) - \sum_{j \in Q^O(t)} \Phi_j^O(t).$$

Picking categories: Full details of the analysis appear in Appendix C, but we mention here the same types of changes mentioned in Section 3. The optimal schedule may cause large continuous increases to Φ by scheduling jobs from multiple 'high weight' categories. We show that KCS has the option of choosing jobs from these same categories. Because KCS approximates an optimal knapsack packing, KCS's schedule performs just as much work as the optimal schedule. Fix a time t. Let $W^*(t)$ be the maximum queued weight of any category in KCS's schedule at time t.

Lemma 5.2. *If KCS's schedule uses* $1+\varepsilon$ *speed and capacity and the optimal schedule uses unit*

[6] As we shall see in the analysis, knapsacks and capacity augmentation enable KCS to use geometrically decreasing heights, where as HPCS needs harmonically decreasing heights to pack jobs tightly and get a bound below 2.

speed and capacity, then

$$\sum_{j\in Q^A(t)} \frac{d}{dt}\left(\Phi_j^A - \frac{8\kappa w_j}{\varepsilon}F_j\right)(t) = \sum_{j\in Q^A(t)} \frac{d}{dt}\frac{8\kappa w_j}{\varepsilon}(R_j + p_j^A - V_j^> + Y_j)(t) \leq -\kappa W^*(t).$$

Proof: Suppose the optimal schedule processes T_i jobs from non-small category i which have a total unmodified height of H_i. Then,

$$(1) \qquad \sum_i \sum_{j\in Q_i^A(t)} \frac{d}{dt}\frac{8\kappa w_j}{\varepsilon}\left(-V_j^> + Y_j\right)(t) \leq \sum_i T_i \frac{8\kappa W_i(t)}{(1+\varepsilon/2)^i\varepsilon}.$$

Let H_X be the capacity *not* used by the optimal schedule for jobs in non-small categories. We see

$$(2) \qquad \sum_{j\in Q_X^A(t)} \frac{d}{dt}\frac{8\kappa w_j}{\varepsilon}\left(-V_j^> + Y_j\right)(t) \leq H_X \frac{8\kappa W_X(t)}{\varepsilon}.$$

Let Δ^O be the sum of the right hand sides of (1) and (2). Note that Δ^O may be larger than the actual rate of increase in Φ due to the V_j and Y_j terms.

Now, consider the following set of jobs $U \in Q^A(t)$. For each non-small category i, set U contains the min $\left\{|Q_i^A(t)|, T_i\right\}$ most dense jobs from $Q_i^A(t)$. Each of the jobs chosen from $Q_i^A(t)$ has height at most $1/(1+\varepsilon/2)^i$ and the optimal schedule processes H_i height worth of jobs, each with height at least $1/(1+\varepsilon/2)^{i+1}$. Therefore, the jobs chosen from $Q_i^A(t)$ have total height at most $(1+\varepsilon/2)H_i$. Set U also contains up to $H_X + \varepsilon/2$ height worth of jobs from $Q_X^A(t)$ chosen greedily in decreasing order of density. The jobs of U together have height at most $\sum_i (1+\varepsilon/2)H_i + H_X + \varepsilon/2 \leq 1+\varepsilon$.

Now, suppose KCS schedules the jobs from set U at unit speed or faster. Consider a non-small category i. Observe $T_i \leq (1+\varepsilon/2)^i$. Using similar techniques to those given earlier, we observe

$$(3) \qquad \sum_i \sum_{j\in Q_i^A(t)} \frac{d}{dt}\frac{8\kappa w_j}{\varepsilon}\left(R_j + p_j^A\right)(t) \leq \sum_i -T_i \frac{8\kappa W_i(t)}{(1+\varepsilon/2)^i\varepsilon}.$$

Now, consider any job j from small category X. If $j \in U$, then $\frac{d}{dt}p_j^A(t) =\leq -1 \leq -H_X$. If $j \notin U$, then there are jobs with higher height density in U with total height at least H_X.

$$(4) \qquad \sum_{j\in Q_X^A(t)} \frac{d}{dt}\frac{8\kappa w_j}{\varepsilon}\left(R_j + p_j^A\right)(t) \leq -H_X \frac{8\kappa W_i(t)}{\varepsilon}.$$

Let Δ^U be the sum of the right hand sides of (3) and (4).

Recall the knapsack instance $K(t)$ used by KCS. Let U' be a solution to $K(t)$ and let $f(U')$ be the value of solution U'. Let $\Delta^A = \sum_{j\in Q^A(t)} \frac{d}{dt}\frac{8\kappa w_j}{\varepsilon}(R_j + p_j^A)(t)$. If KCS schedules the jobs corresponding to the U', then $\Delta^A = -\frac{8(1+\varepsilon)\kappa}{\varepsilon}f(U')$. Let U^* be the optimal solution to $K(t)$ and let $\Delta^* = -\frac{8(1+\varepsilon)\kappa}{\varepsilon}f(U^*)$. Because U as defined above is one potential solution to $K(t)$ and KCS runs an FPTAS on $K(t)$ to decide which jobs are scheduled, we see

$$\begin{aligned}\Delta^O + \Delta^A &\leq \Delta^O + (1-\varepsilon/2)\Delta^* \\ &\leq \Delta^O - \frac{8(1+\varepsilon/4)\kappa}{\varepsilon}f(U^*) \\ &\leq \Delta^O + \Delta^U - 2\kappa f(U^*) \\ &= -2\kappa f(U^*).\end{aligned}$$

If category X has the maximum queued weight, then one possible solution to $K(t)$ is to greedily schedule jobs from category X in descending order of height density. Using previously shown techniques, we see $f(U^*) \geq W^*(t)$. If a non-small category i has maximum queued weight, then a possible solution is to greedily schedule jobs from category i in descending order of height density. Now we see $f(U^*) \geq (1/2)W^*(t)$. Using our bound on $\Delta^O + \Delta^A$, we prove the lemma. □

6 Conclusions and Open Problems

In this paper, we introduce the problem of weighted flow time on capacitated machines and give the first upper and lower bounds for the problem. Our main result is a $(1+\varepsilon)$-speed $(1+\varepsilon)$-capacity $O(1/\varepsilon^3)$-competitive algorithm that can be extended to the unrelated multiple machines setting. However, it remains an open problem whether $(1+\varepsilon)$-speed and unit capacity suffices to be constant competitive for any $\varepsilon > 0$. Further, the multi-dimensional case remains open, where jobs can be scheduled simulateously given that no single dimension of the server's capacity is overloaded.

Acknowledgements. The authors would like to thank Chandra Chekuri, Sungjin Im, Benjamin Moseley, Aranyak Mehta, Gagan Aggarwal, and the anonymous reviewers for helpful discussions and comments on an earlier version of the paper. Research by the first author is supported in part by the Department of Energy Office of Science Graduate Fellowship Program (DOE SCGF), made possible in part by the American Recovery and Reinvestment Act of 2009, administered by ORISE-ORAU under contract no. DE-AC05-06OR23100.

References

[1] Amazon elastic compute cloud (amazon ec2). ⟨http://aws.amazon.com/ec2/⟩.

[2] Google appengine cloud. ⟨http://appengine.google.com/⟩.

[3] Vmware. ⟨http://www.vmware.com/⟩.

[4] Vmware cloud computing solutions (public/private/hybrid). ⟨http://www.vmware.com/solutions/cloud-computing/index.html⟩.

[5] Vmware distributed resource scheduler. ⟨http://www.vmware.com/products/drs/overview.html⟩.

[6] Vmware server consolidation solutions. ⟨http://www.vmware.com/solutions/consolidation/index.html⟩.

[7] Server consolidation using VMware ESX Server, 2005. ⟨http://www.redbooks.ibm.com/redpapers/pdfs/redp3939.pdf⟩.

[8] S. Anand, N. Garg, and A. Kumar. Resource augmentation for weighted flow-time explained by dual fitting. *Proc. 23rd Ann. ACM-SIAM Symp. Discrete Algorithms*, pp. 1228–1241. 2012.

[9] B. Awerbuch, Y. Azar, S. Plotkin, and O. Waarts. Competitive routing of virtual circuits with unknown duration. *Proc. 5th Ann. ACM-SIAM Symp. Discrete Algorithms*, pp. 321–327. 1994.

[10] Y. Azar, B. Kalyanasundaram, S. Plotkin, K. Pruhs, and O. Waarts. On-line load balancing of temporary tasks. *Proc. 3rd Workshop on Algorithms and Data Structures*, pp. 119–130. 1993.

[11] N. Bansal and H.-L. Chan. Weighted flow time does not admit $O(1)$-competitive algorithms. *Proc. 20th Ann. ACM-SIAM Symp. Discrete Algorithms*, pp. 1238–1244. 2009.

[12] N. Bansal and K. Dhamdhere. Minimizing weighted flow time. *ACM Trans. Algorithms* 3(4):39:1–39:14, 2007.

[13] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neuge-bauer, I. Pratt, and A. Warfield. Xen and the art of virtualization. *Proc. 19th Symp. on Operating Systems Principles (SOSP)*, pp. 164–177. 2003.

[14] L. Becchetti, S. Leonardi, A. Marchetti-Spaccamela, and K. Pruhs. Online weighted flow time and deadline scheduling. *Proc. 5th International Workshop on Randomization and Approximation*, pp. 36–47. 2001.

[15] N. Bobroff, A. Kochut, and K. Beaty. Dynamic placement of virtual machines for managing sla violations. *Proc. 10th Ann. IEEE Symp. Integrated Management (IM)*, pp. 119 – 128. 2007.

[16] C. Bussema and E. Torng. Greedy multiprocessor server scheduling. *Oper. Res. Lett.* 34(4):451–458, 2006.

[17] J. Chadha, N. Garg, A. Kumar, and V. N. Muralidhara. A competitive algorithm for minimizing weighted flow time on unrelated machines with speed augmentation. *Proc. 41st Ann. ACM Symp. Theory Comput.*, pp. 679–684. 2009.

[18] C. Chekuri, A. Goel, S. Khanna, and A. Kumar. Multi-processor scheduling to minimize flow time with epsilon resource augmentation. *Proc. 36th Ann. ACM Symp. Theory Comput.*, pp. 363–372. 2004.

[19] C. Chekuri, S. Khanna, and A. Zhu. Algorithms for minimizing weighted flow time. *Proc. 33rd Ann. ACM Symp. Theory Comput.*, pp. 84–93. 2001.

[20] V. Chudnovsky, R. Rifaat, J. Hellerstein, B. Sharma, and C. Das. Modeling and synthesizing task placement constraints in google compute clusters. *Symp. on Cloud Computing*. 2011.

[21] J. Dean and S. Ghemawat. Mapreduce: Simplified data processing on large clusters. *Proc. 6th Symp. on Operating System Design and Implementation*. 2004.

[22] K. Fox. Online scheduling on identical machines using SRPT. Master's thesis, University of Illinois, Urbana-Champaign, 2010.

[23] K. Fox and B. Moseley. Online scheduling on identical machines using SRPT. *Proc. 22nd ACM-SIAM Symp. Discrete Algorithms*, pp. 120–128. 2011.

[24] N. Garg and A. Kumar. Minimizing average flow-time : Upper and lower bounds. *Proc. 48th Ann. IEEE Symp. Found. Comput. Sci.*, pp. 603–613. 2007.

[25] H. Ibarra and E. Kim. Fast approximation algorithms for the knapsack and sum of subset problems. *J. ACM* 22(4):463–468. ACM, Oct. 1975.

[26] S. Im and B. Moseley. An online scalable algorithm for minimizing lk-norms of weighted flow time on unrelated machines. *Proc. 22nd ACM-SIAM Symp. Discrete Algorithms*, pp. 95–108. 2011.

[27] S. Im, B. Moseley, and K. Pruhs. A tutorial on amortized local competitiveness in online scheduling. *SIGACT News* 42(2):83–97. ACM, June 2011.

[28] B. Kalyanasundaram and K. Pruhs. Speed is as powerful as clairvoyance. *Proc. 36th Ann. IEEE Symp. on Found. Comput. Sci.*, pp. 214–221. 1995.

[29] E. Lawler. Fast approximation algorithms for the knapsack problems. *Mathematics of Operations Research* 4(4):339–356, 1979.

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

[31] S. Martello and P. Toth. *Knapsack Problems: Algorithms and Computer Implementations*. John Wiley, 1990.

[32] A. Mishra, J. Hellerstein, W. Cirne, and C. Das. Towards characterizing cloud backend workloads: Insights from Google compute clusters, 2010.

[33] A. Singh, M. Korupolu, and D. Mohapatra. Server-storage virtualization: Integration and load-balancing in data centers. *Proc. IEEE-ACM Supercomputing Conference (SC)*, pp. 53:1–53:12. 2008.

[34] Q. Zhang, J. Hellerstein, and R. Boutaba. Characterizing task usage shapes in Google compute clusters. *Proc. 5th International Workshop on Large Scale Distributed Systems and Middleware*. 2011.

Appendix

A Analysis of HRDF

In this appendix, we complete the analysis of HRDF. We use the definitions and potential function as given in Section 3. Consider the changes that occur to $A + \Phi$ over time. The events we need to consider follow:

Job arrivals: Consider the arrival of job j at time r_j. The potential function Φ gains the terms Φ_j^A and $-\Phi_j^O$. We see

$$\begin{aligned}\Phi_j^A(r_j) - \Phi_j^O(r_j) &= \frac{(1+\varepsilon)w_j}{\varepsilon} p_j \\ &\leq \frac{1+\varepsilon}{\varepsilon}\mathrm{OPT}_j\end{aligned}$$

For any term Φ_k^A with $k \neq j$, job j may only contribute to R_k and $-V_k^>$, and its contributions to the two terms cancel. Further, job j has no effect on any term Φ_k^O where $k \neq j$. All together, job arrivals increase $A + \Phi$ by at most

$$\frac{1+\varepsilon}{\varepsilon}\mathrm{OPT}. \tag{5}$$

Continuous job processing: Consider any time t. We have the following:

- By Lemma 3.2,

$$\sum_{j \in Q^A(t)} \frac{d}{dt}\frac{w_j}{\varepsilon}(R_j + (1+\varepsilon)p_j^A - V_j^> + Y_j)(t) \leq -W^A(t).$$

- If HRDF's schedule processes job j at time t, then

$$\begin{aligned}\frac{d}{dt}\frac{w_j}{\varepsilon}F_j(t) &= \frac{h_j w_j}{\varepsilon}\sum_{k \in Q^O(t)}\frac{w_k}{w_j} \\ &= \frac{h_j}{\varepsilon}\frac{d}{dt}\mathrm{OPT}(t).\end{aligned}$$

Recall $S^A(t)$ is the set of jobs processed by HRDF's schedule at time t. Observe that

$$\sum_{j \in S^A(t)} h_j \leq (2+\varepsilon).$$

Therefore,

$$\sum_{j \in S^A(t)} \frac{d}{dt}\frac{w_j}{\varepsilon}F_j(t) \leq \frac{2+\varepsilon}{\varepsilon}\frac{d}{dt}\mathrm{OPT}(t).$$

- Finally,

$$\begin{aligned}\sum_{j \in Q^O(t)} \frac{d}{dt} - \frac{w_j}{\varepsilon}(R_j^<)(t) &\leq \sum_{j \in Q^O(t)}\frac{(2+\varepsilon)w_j}{\varepsilon} \\ &= \frac{2+\varepsilon}{\varepsilon}\frac{d}{dt}\mathrm{OPT}(t)\end{aligned}$$

Combining the above inequalities gives us the following:

$$\begin{aligned}\frac{d}{dt}(A+\Phi)(t) &= \sum_{j \in Q^A(t)}\left(\frac{d}{dt}A_j(t) + \frac{d}{dt}\Phi_j^A(t)\right) \\ &\quad - \sum_{j \in Q^O(t)}\frac{d}{dt}\Phi_j^O(t) \\ &\leq W^A(t) - W^A(t) + \frac{2+\varepsilon}{\varepsilon}\frac{d}{dt}\mathrm{OPT}(t) \\ &\quad + \frac{2+\varepsilon}{\varepsilon}\frac{d}{dt}\mathrm{OPT}(t) \\ &= \frac{4+2\varepsilon}{\varepsilon}\frac{d}{dt}\mathrm{OPT}(t)\end{aligned}$$

Therefore, the total increase to $A + \Phi$ due to continuous processing of jobs is at most

$$\int_{t \geq 0}\frac{4+2\varepsilon}{\varepsilon}\frac{d}{dt}\mathrm{OPT}(t)dt = \frac{4+2\varepsilon}{\varepsilon}\mathrm{OPT}. \tag{6}$$

Job completions: The completion of job j in HRDF's schedule increases Φ by

$$-\Phi_j^A(C_j^A) = \frac{w_j}{\varepsilon}\left(V_j^>(C_j^A) - Y_j(C_j^A) - F_j(C_j^A)\right).$$

Similarly, the completion of job j by the optimal schedule increases Φ by

$$-\Phi_j^O(C_j^O) = \frac{w_j}{\varepsilon}\left(R_j^<(C_j^O)\right).$$

We argue that the negative terms from these increases cancel the positive terms. Fix a job j and consider any job k contributing to $V_j^>(C_j^A)$. By definition of $V_j^>$, job k arrives after job j and is more height dense than job j upon its arrival. We see

$$\frac{w_k}{h_k p_k} > \frac{w_j}{h_j p_j^A(r_k)} \implies \frac{h_j p_j^A(r_k) w_k}{w_j} > h_k p_k.$$

Job k is alive to contribute a total of $\frac{h_j p_j^A(r_k) w_k}{w_j} > h_k p_k$ to F_j during times HRDF's schedule processes job j. The decrease in Φ of k's contribution to $F_j(C_j^A)$ negates the increase caused by k's contribution to $V_j^>(C_j^A)$.

Now, consider job k contributing to $R_j^<(C_j^O)$. By definition we know k arrives *before* job j and that job j is less height dense than job k at the time j arrives. We see

$$\frac{w_k}{h_k p_k^A(r_j)} > \frac{w_j}{h_j p_j} \implies w_k h_j p_j > w_j h_k p_k^A(r_j).$$

Job j does not contribute to $V_k^>$, because it is less height dense than job k upon its arrival. Further, job k is alive in HRDF's schedule when the optimal schedule completes job j. Therefore, all processing done to j by the optimal schedule contributes to Y_k. So, removing job j's contribution to $Y_k(C_k^A)$ causes Φ to decrease by $\frac{w_k}{\varepsilon} h_j p_j > \frac{w_j}{\varepsilon} h_k p_k^A(r_j)$, an upper bound on the amount of increase caused by removing k's contribution to $-R_j^<(C_j^O)$. We conclude that job completions cause no overall increase in $A + \Phi$.

Density inversions: The final change we need to consider for Φ comes from HRDF's schedule changing the ordering of the most height dense jobs. The R_j terms are the only ones changed by these events. Suppose HRDF's schedule causes some job j to become less height dense than another job k at time t. Residual density changes are continuous, so

$$\frac{w_j}{h_j p_j^A(t)} = \frac{w_k}{h_k p_k^A(t)} \implies w_j h_k p_k^A(t) = w_k h_j p_j^A(t).$$

At time t, job k starts to contribute to R_j, but job j stops contributing to R_k. Therefore, the change to Φ from the inversion is

$$\frac{w_j}{\varepsilon} h_k p_k^A(t) - \frac{w_k}{\varepsilon} h_j p_j^A(t) = 0.$$

We conclude that these events cause no overall increase in $A + \Phi$.

A.1 Completing the proof. We may now complete the proof of Theorem 3.1. By summing the increases to $A + \Phi$ from (5) and (6), we see

$$A \le \frac{5 + 3\varepsilon}{\varepsilon} \mathrm{OPT} = O\left(\frac{1}{\varepsilon}\right) \mathrm{OPT}. \tag{7}$$

B Analysis of HPCS

In this appendix, we complete the analysis of HPCS. We use the definitions and potential function as given in Section 4. Consider the changes that occur to $A + \Phi$ over time. As is the case for HRDF, job completions and density inversions cause no overall increase to $A + \Phi$. The analyses for these changes are essentially the same as those given in Appendix A, except the constants may change. The remaining events we need to consider follow:

Job arrivals: Consider the arrival of job j at time r_j. The potential function Φ gains the terms Φ_j^A and $-\Phi_j^O$. We see

$$\begin{aligned}\Phi_j^A(r_j) - \Phi_j^O(r_j) &= \lambda\kappa w_j p_j \\ &\le \lambda\kappa \mathrm{OPT}_j.\end{aligned}$$

For any term Φ_k^A with $k \neq j$, job j may only contribute to R_k and $-V_k^>$, and its contributions to the two terms cancel. Further, job j has no effect on any term Φ_k^O where $k \neq j$. All together, job arrivals increase $A + \Phi$ by at most

$$\lambda\kappa \mathrm{OPT}. \tag{8}$$

Continuous job processing: Consider any time t. We have the following:

- By Lemma 4.2,

  $$\sum_{j \in Q^A(t)} \frac{d}{dt} \lambda\kappa w_j (R_j + p_j^A - V_j + Y_j)(t) \le -\kappa W^*(t)$$

  if $\kappa \ge 200$ and $\lambda \ge 100$.

- Note $\sum_{j \in S^A(t)} h_j' \le 1$. Therefore,

  $$\begin{aligned}\sum_{j \in S^A(t)} \frac{d}{dt} \lambda\kappa w_j F_j(t) &= \sum_{j \in S^A(t)} \lambda\kappa h_j' w_j \sum_{k \in Q^O(t)} \frac{w_k}{w_j} \\ &\le \lambda\kappa \frac{d}{dt} \mathrm{OPT}(t).\end{aligned}$$

- Observe:

  $$\begin{aligned}\sum_{j \in Q^O(t)} \frac{d}{dt} - \lambda\kappa w_j (R_j^<)(t) &\le \sum_{j \in Q^O(t)} 1.75 \lambda\kappa w_j \\ &= 1.75 \lambda\kappa \frac{d}{dt} \mathrm{OPT}(t)\end{aligned}$$

Combining the above inequalities gives us the following:

$$\begin{aligned}\frac{d}{dt}(A+\Phi)(t) &= \sum_{j\in Q^A(t)}\left(\frac{d}{dt}A_j(t)+\frac{d}{dt}\Phi_j^A(t)\right)\\ &\quad-\sum_{j\in Q^O(t)}\frac{d}{dt}\Phi_j^O(t)\\ &\le \kappa W^*(t)-\kappa W^*(t)+\lambda\kappa\frac{d}{dt}\mathrm{OPT}(t)\\ &\quad+1.75\lambda\kappa\frac{d}{dt}\mathrm{OPT}(t)\\ &\le 2.75\lambda\kappa\frac{d}{dt}\mathrm{OPT}(t)\end{aligned}$$

Therefore, the total increase to $A+\Phi$ due to continuous processing of jobs is at most

$$\int_{t\ge 0}2.75\lambda\kappa\frac{d}{dt}\mathrm{OPT}(t)dt = 2.75\lambda\kappa\mathrm{OPT}. \tag{9}$$

B.1 Completing the proof. We may now complete the proof of Theorem 4.1. Set $\kappa = 200$ and $\lambda = 100$. By summing the increases to $A+\Phi$ from (8) and (9), we see

$$A \le 3.75\lambda\kappa\mathrm{OPT} = O(\mathrm{OPT}). \tag{10}$$

C Analysis of KCS

In this appendix, we complete the analysis of KCS. We use the definitions and potential function as given in Section 5. Consider the changes that occur to $A+\Phi$ over time. As is the case for HRDF, job completions and density inversions cause no overall increase to $A+\Phi$. The analyses for these changes are essentially the same as those given in Appendix A, except the constants may change. The remaining events we need to consider follow:

Job arrivals: Consider the arrival of job j at time r_j. The potential function Φ gains the terms Φ_j^A and $-\Phi_j^O$. We see

$$\begin{aligned}\Phi_j^A(r_j)-\Phi_j^O(r_j) &= \frac{8\kappa w_j}{\varepsilon}p_j\\ &\le O\left(\frac{1}{\varepsilon^3}\right)\mathrm{OPT}_j.\end{aligned}$$

Again, the changes to other terms cancel. All together, job arrivals increase $A+\Phi$ by at most

$$O\left(\frac{1}{\varepsilon^3}\right)\mathrm{OPT}. \tag{11}$$

Continuous job processing: Consider any time t. We have the following:

- By Lemma 5.2,
$$\sum_{j\in Q^A(t)}\frac{d}{dt}\frac{8\kappa w_j}{\varepsilon}(R_j+p_j^A-V_j^{>}+Y_j)(t) \le -\kappa W^*(t).$$
- For any job j in non-small category i, we have $h_j' \le (1+\varepsilon/2)h_j$ by definition. Also, $\sum_{j\in S^A(t)}h_j \le 1+\varepsilon$. Therefore,
$$\begin{aligned}\sum_{i\in S^A(t)}\frac{d}{dt}\frac{8\kappa w_j}{\varepsilon}F_j(t) &= \sum_{i\in S^A(t)}\frac{8\kappa h_j' w_j}{\varepsilon}\frac{w_k}{w_j}\\ &\le \frac{8(1+\varepsilon)^2\kappa}{\varepsilon}\frac{d}{dt}\mathrm{OPT}(t).\end{aligned}$$
- The number of non-small category i jobs processed by KCS's schedule is at most $(1+\varepsilon)^2/h_j'$. Therefore,
$$\begin{aligned}\sum_{i\in Q^O(t)}\frac{d}{dt}-\frac{8\kappa w_j}{\varepsilon}(R_j^{<})(t) &\le \sum_{i\in Q^O(t)}\frac{8(1+\varepsilon)^3\kappa w_j}{\varepsilon}\\ &= \frac{8(1+\varepsilon)^3\kappa}{\varepsilon}\frac{d}{dt}\mathrm{OPT}(t)\end{aligned}$$

Combining the above inequalities gives us the following:

$$\begin{aligned}\frac{d}{dt}(A+\Phi)(t) &= \sum_{j\in Q^A(t)}\left(\frac{d}{dt}A_j(t)+\frac{d}{dt}\Phi_j^A(t)\right)\\ &\quad-\sum_{j\in Q^O(t)}\frac{d}{dt}\Phi_j^O(t)\\ &\le \kappa W^*(t)-\kappa W^*(t)\\ &\quad+\frac{8(1+\varepsilon)^2\kappa}{\varepsilon}\frac{d}{dt}\mathrm{OPT}(t)\\ &\quad+\frac{8(1+\varepsilon)^3\kappa}{\varepsilon}\frac{d}{dt}\mathrm{OPT}(t)\\ &\le \frac{16(1+\varepsilon)^3\kappa}{\varepsilon}\frac{d}{dt}\mathrm{OPT}(t)\end{aligned}$$

Therefore, the total increase to $A+\Phi$ due to continuous processing of jobs is at most

$$\int_{t\ge 0}\frac{16(1+\varepsilon)^3\kappa}{\varepsilon}\frac{d}{dt}\mathrm{OPT}(t)dt = O\left(\frac{1}{\varepsilon^3}\right)\mathrm{OPT} \tag{12}$$

C.1 Completing the proof. We may now complete the proof of Theorem 5.1. By summing the increases to $A+\Phi$ from (11) and (12), we see

$$A \le O\left(\frac{1}{\varepsilon^3}\right)\mathrm{OPT}. \tag{13}$$

D Extensions to Multiple Machines

Here, we describe how our results can be extended to work with multiple unrelated machines. In the unrelated machines variant of the CFT problem, we have m machines. Job j has weight $w_{\ell j}$, processing time $p_{\ell j}$, and height $h_{\ell j}$ if it is assigned to machine ℓ. At each point in time, the server may schedule an arbitrary number of jobs assigned to each machine ℓ as long as the sum of their heights does not exceed 1. For each machine ℓ and job j, we assume either $h_{\ell j} \leq 1$ or $p_{\ell j} = \infty$.

The algorithms in this paper can be extended easily to the unrelated machines model. These extensions are non-migratory, meaning a job is fully processed on a single machine. In fact, our extensions are immediate dispatch, meaning each job is assigned to a machine upon its arrival. Let $\mathcal{A}$ be one of the algorithms given above. In order to extend $\mathcal{A}$, we simply apply an idea of Chadha *et al.* [17] also used by Im and Moseley [26]. Namely, we run $\mathcal{A}$ on each machine independently and assign jobs to the machine that causes the least increase in $\mathcal{A}$'s potential function.

A more precise explanation follows. We use all the definitions given for $\mathcal{A}$ and its analysis. Let $Q_\ell^A(t)$ be the set of alive jobs assigned to machine ℓ at time t. Upon job j's arrival, assign job j to the machine ℓ that causes the least increase in $\sum_{k \in Q_\ell^A(t)} w_{\ell k}(R_k + p_k^A)$ where each term R_k and p_k^A is modified to consider only jobs assigned to ℓ and their processing times and heights on ℓ. Each machine independently runs $\mathcal{A}$ on the jobs to which it is assigned using processing times, weights, and heights as given for that machine. The extensions to each algorithm $\mathcal{A}$ given above have the same competitive ratios as their single machine counterparts, given the amount of resource augmentation required in the single machine model for $\mathcal{A}$.

D.1 Analysis. Here, we sketch the analysis of our extension for multiple unrelated machines. Our analysis is essentially the same as the single machine case, except the terms in the potential functions are slightly changed. Let $\mathcal{A}$ be the algorithm we are extending. We begin by assuming the optimal schedule is non-migratory and immediate dispatch; whenever a job j arrives, it is immediately assigned to a machine ℓ and all processing for j is performed on ℓ. For each term R_j, $V_j^>$, Y_j, and F_j, we change its definition to only include contributions by jobs scheduled on the machine chosen for job j by our extension to $\mathcal{A}$. For $R_j^<$, we change its definition to only include contributions by jobs scheduled on the machine chosen for job j by the optimal schedule. For example, if $\mathcal{A}$ is HRDF, then let $p_{\ell j}^A(t)$ be the remaining processing time of j on machine ℓ. Let $d_{\ell j}^A(t) = w_{\ell j}/(h_{\ell j} p_{\ell j}^A(t))$. Let $Q_\ell^{Aj}(t) = \left\{k \in Q_\ell^A(t) : d_{\ell k}^A(t) > d_{\ell j}^A(t)\right\}$ Finally, we have $R_j(t) = \sum_{k \in Q_\ell^{Aj}(t)} h_{\ell k} p_{\ell k}^A(t)$. The definition of the potential function for each algorithm $\mathcal{A}$ then remains unchanged.

The analysis for continuous job processing, job completitions, and density inversions is exactly the same as those given above for $\mathcal{A}$ as we restrict our attention to a single machine in each case. We only need to consider job arrivals. Let c_j be the coefficient of $p_j^A(t)$ in the potential function for $\mathcal{A}$ (for example, $c_j = (1+\varepsilon)w_j/\varepsilon$ for HRDF). Recall that in each analysis, the increase to Φ from job arrivals was at most $c_j \mathrm{OPT}_j$. Consider the arrival of job j at time r_j. Let ℓ_j^A be the machine chosen for j by our extension and let ℓ_j^O be the machine chosen by the optimal algorithm. By our extension's choice of machine, the potential function Φ increases by $c_j p_{\ell_j^O j}$ upon job j's arrival. This value is at most $c_j \mathrm{OPT}_j$. Summing over all jobs, we see job arrivals increase Φ by the same amount they did in the earlier analyses.

Finally, we compare against optimal schedules that can be migratory. In this case, we model a migratory optimal schedule as assigning an $\alpha_{\ell j}$ fraction of job j to each machine ℓ ($\sum_\ell \alpha_{\ell i} = 1$) upon job j's arrival. We further adjust our definitions of $V_j^>$ and $R_j^<$ to account for these fractional jobs; $Y_j^>$ only includes fractional jobs that are more dense than j and $R_j^<$ includes contributions by jobs that are more height dense than the fractional job the optimal schedule assigns in place of j. Now by our extension's choice in machine, the arrival of job j increases Φ by $\sum_\ell \alpha_{\ell j} c_j p_{\ell j} \leq c_j \mathrm{OPT}_j$. Summing over all jobs completes our analysis of job arrivals. Continous job processing, job completions, and density inversions work as before.