Scheduling Jobs with Dwindling Resource Requirements in Clouds

Sivan Albagli-Kim* Hadas Shachnai* Tami Tamir †

* Computer Science Department, Technion, Haifa 32000, Israel
E-mail: {hadas,sivanal}@cs.technion.ac.il
† School of Computer Science, The Interdisciplinary Center, Herzliya, Israel
E-mail: tami@idc.ac.il

***Abstract*—We consider a job-scheduling problem arising on cloud systems and in broadcasting networks, where the goal is to optimally utilize a limited amount of a resource (e.g., cloud servers, bandwidth, or storage capacity) available along a given time interval. The resource is utilized by a set of weighted jobs. The processing of a job consists of several contiguous stages, each having a specific length and a specific resource-demand, such that the set of demands forms a decreasing sequence. Each job is associated with a release time and a deadline, defining the time interval in which it can be processed. Some notable applications for this scenario include progressive download, QuickStart and prefetching methods, hierarchical image reconstruction, and routine security and maintenance tasks. The goal is to find a feasible schedule of a maximum-weight subset of the jobs. In a feasible schedule, at any time, the total amount of resource allocated to the active jobs does not exceed the available amount of resource.**

Since this problem is NP-hard already for highly restricted inputs, we focus on obtaining approximation algorithms and heuristics and present a comparative study among them. Our main result, the first constant-factor approximation algorithm for the problem, generalizes the state of art for the fundamental problem of resource constrained real-time scheduling, to scenarios where jobs may have *dwindling* resource requirements. Our empirical study shows that this algorithm is in fact nearly optimal for realistic inputs.

I. Introduction

The emergence of cloud systems as a common computation resource gives rise to plenty of optimization problems. Many of these problems deal with scheduling jobs that require cloud services, or with optimal utilization of the limited cloud resources. This paper considers one such resource-utilization problem. Specifically, the service provider has a limited amount of some resource (e.g., bandwidth, computational power, storage), and is presented with a set of demands for the resource. Since the amount of resource is limited, only a subset of the demands can be serviced, and the goal is to select the most profitable subset and to grant service in a feasible way.

Several variants of this scheduling problem were studied in the past (see below), however, all of these previous works refer to inputs where each job has a fixed resource demand throughout its processing. In practice, the resource consumption of jobs is not necessarily fixed and tends to decrease along the processing of the job. That is, jobs require the largest amount of resource for the initial part of their processing, and their resource demands dwindle as they make progress towards completion.

For example, a progressive download is the transfer of digital media files from a server to a client, typically using HTTP protocol. The consumer may begin playback of the media before the download is complete. Since, unlike streaming, the digital media data is stored at the end-user device, the bandwidth requirement of the user reduces with time − initially, many segments are downloaded, and as the playback advances, less segments are missing (see, e.g., [3], [11]). Similarly, in image processing, hierarchical bitstream structures are used for progressive image reconstruction. Thus, the required bit rate reduces with time [7]. Dwindling resource requirements characterize also several applications in network security − files that are uploaded to a cloud or to a shared server need to undergo a sequence

978-1-4799-3360-0/14/$31.00 ©2014 IEEE

of tests. These tests are performed in parallel, and their total capacity reduces with time.

While such scenarios are common, not much attention was given to resource allocation in servicing jobs with dwindling requirements. In this paper, we formulate the problem theoretically, study its complexity and present an approximation algorithm with guaranteed performance ratio, as well as several heuristics.

Assume that B units of some resource are available during the time interval $[0, T]$. The system receives a set of n service requests. Each request corresponds to a job whose processing consists of several contiguous stages. Each stage has a specific length and a specific resource-demand, such that the set of demands forms a decreasing sequence. Formally, the resource requirement of job j is given by a tuple $\langle (p_j^1, c_j^1), \ldots, (p_j^{b_j}, c_j^{b_j}) \rangle$, where $b_j \geq 1$ denotes the number of stages job j consists of, and (p_j^k, c_j^k) denote the processing time and the capacity (resource demand) of stage k. The total processing time of job j is $p_j = \sum_{k=1}^{b_j} p_j^k$. In addition, each job has a release time, $r_j \geq 0$, a deadline, $d_j \leq T$ and a weight, w_j, corresponding to the profit gained if job j completes processing by its deadline.

Example: Assume that the system has $B = 100$GByte of bandwidth available from time 0 to time 10. Four clients require broadcasting services, each presenting a single job to the system. Job 1 consists of three stages $\langle (1, 90), (2, 60), (1, 20) \rangle$. The job is released at time 0 and its deadline is 8. That is, it should be processed in the time interval $[0, 8]$. Job 2 consists of four stages $\langle (2, 80), (1, 50), (2, 20), (1, 10) \rangle$, and it should be processed in $[2, 10]$. Job 3 consists of two stages $\langle (3, 70), (1, 60) \rangle$, and it should be processed in $[1, 10]$. Finally, Job 4 consists of three stages $\langle (2, 60), (2, 30), (2, 20) \rangle$, and it should be processed in $[2, 9]$. Assume that job weights are $5, 4, 3, 2$, respectively. Fig. 1 presents a possible feasible schedule of the first three jobs. The profit from this schedule is $5 + 4 + 3 = 12$. Note that there is no meaning to the physical location of the jobs (i.e., the 'stairs' in the processing of J_2^1, J_3^1 do not really exist), and the only constraint is that the total demand of job-stages processed at any time is at most $B = 100$.

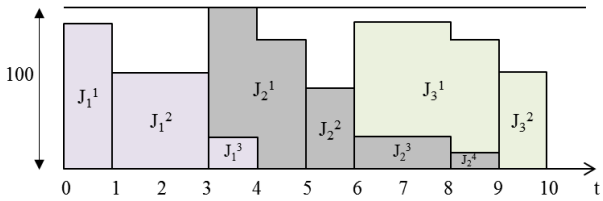

Fig. 1. A feasible schedule of three jobs.

A. Related Work

Scheduling has been a perpetual field of research in operations research and computer science (see e.g., [5], [1], [2], [9] and references therein). Of specific relevance to our work are [9], [1], which consider variations of the interval-scheduling problem. In particular, these papers consider the special case of our problem where the amount of resources allocated to each job is *fixed* throughout its execution. The best known results for this problem are due to [1]. The authors distinguish between discrete inputs, where each job has a finite set of possible start times, and continuous inputs, where jobs are associated with time windows in which they can be processed. The authors present approximation algorithms that achieve for the two types of inputs the ratios of 5 and $5 + \varepsilon$, respectively.

Our results relate also to previous work on resource allocation to batch jobs in cloud systems. Jain et al. [6] studied the *Bounded Flexible Scheduling (BFS)* problem that is directly motivated by the cloud computing paradigm. A cloud containing B servers receives a set of job requests with heterogeneous demand and values per deadline, where the objective is to maximize the sum of the values of the scheduled jobs. The scheduling of a job is flexible, i.e., it can be allocated a different number of servers per time unit and in a possibly preemptive (non-contiguous) manner, under parallelism thresholds. The parallelism threshold represents the job's limitations on parallelized execution. For any job j, denote by k_j the maximum number of servers that can be allocated to job j at any given time unit, and let $k_{max} = \max_j k_j$ be the maximal parallelism threshold across jobs. The paper [6] presents an LP-based approximation algorithm for BFS that is shown to yield a ratio of $(1 + \frac{B}{B - k_{max}})(1 + \varepsilon)$ to the optimal, for any $\varepsilon > 0$. This ratio approaches 2 for instances where k_{max} is much smaller than the cloud capacity B; however, the existence of efficient approximation for instances where k_{max} is large remained open.

Another resource constrained scheduling problem in which jobs consist of stages was studied by Bar-Yehuda and Rawitz [4]. In their setting, each job consists of at most b uniform-capacity stages that may

occur in non-consecutive time intervals. The authors present a $6b$-approximation algorithm, using an extension of the primal-dual schema.

B. Our Contribution

The *Dwindling-Jobs Scheduling* problem is a generalization of several NP-hard problems. It combines the hardness of classical subset-selection problems such as Knapsack (corresponding to instances where all single-stage jobs require the resource for the same interval and have demands and weights), with the hardness of the classic interval-scheduling problem [10] (corresponding to the case of uniform-width single-stage jobs). Thus, we focus on obtaining polynomial-time approximation algorithms and heuristics. For $r \geq 1$, a polynomial-time r-approximation algorithm for a maximization problem P is a polynomial-time algorithm that outputs, for any instance I of P, a solution whose value is equal to at least $1/r$ times the value of an optimal solution for I.

We present a comparative study of approximation algorithms and heuristics for scheduling jobs with dwindling resource requirements. Our main result (in Section II) is an approximation algorithm that yields a 5.15-approximation for the problem. This generalizes the state of art for the fundamental problem of resource constrained real-time scheduling, to scenarios where the amount of resources allocated to a job can change over time.

In solving the BFS problem, it was shown in [6] that there exists an optimal schedule that is non-preemptive, i.e., each job is scheduled contiguously, and the number of servers allotted to each job is monotonically non-increasing throughout its execution. Thus, the results in Section II can be used to obtain for the BFS problem an improved ratio of $\min(5.15, 1 + \frac{B}{B-k_{max}}(1+\varepsilon))$, where k_{max} is the maximum number of servers allocated to any job. In particular, we can apply the rounding method described in Section II to the non-preemptive monotone (fractional) solution obtained in [6] for the linear programming relaxation of BFS. This gives a constant approximation ratio for *any* value of $1 \leq k_{max} \leq B$.

Our empirical study (in Section III), in which we compare the performance of our approximation algorithm with several natural heuristics for the problem, shows that this algorithm is in fact nearly optimal for realistic inputs.

The heuristics suggested in our empirical study perform well on average on random inputs and are very simple to implement. We analyze and compare their performance and characterize the inputs for which each of them is expected to output good solutions. We also studied the effect of changing the density of the request set, the window size of the jobs, and the number of job-stages on the performance of the proposed heuristics. One of the heuristics constructs the schedule greedily, considering the jobs in non-decreasing order by their release times. The relatively poor performance of this heuristic hints that the on-line version of this problem, where jobs should be admitted to the system (or rejected) upon arrival, is significantly more challenging than the off-line one.

Technique: Bar-Noy et. al. [2] introduced an elegant method of rounding fractional schedules to obtain valid (integral) schedules. Applying this method, they derived the first constant-factor approximation algorithms for the problem of non-preemptively scheduling jobs with weights, release times and deadlines on one machine so as to maximize the weight of those jobs completed by their due dates. Using the common scheduling notation, this problem is denoted $1|r_j|\sum w_j(1-U_j)$. Their technique in essence decomposes the fractional solution into a convex combination of integral solutions and chooses the best, which is guaranteed to have value within a constant-factor of optimal. We present more details of their method in Section II. Our analysis builds on a refined analysis for this rounding method given by Phillips [8], for the problem of real-time scheduling with resource requirements, where the amount of resources allocated to a job is fixed throughout its processing.

II. Approximation Algorithm

In this section we give a constant factor approximation algorithm for the *Dwindling-Jobs Scheduling* problem. The algorithm is based on rounding an optimal (fractional) solution for the linear programming relaxation of the problem.

Theorem 2.1: There is a polynomial-time 5.15-approximation algorithm for the *Dwindling-Jobs Scheduling* problem.

Proof: The proof is based on the framework of [2], [9]. We first find an optimal solution for a linear programming relaxation of the problem, and then round it into a feasible integral solution. In rounding the (fractional) solution we generalize the *coloring subroutine*

of [2]. We assume that time is slotted and jobs can begin their processing only at the beginning of a time slot. The number of time-slots is polynomial. As in [2], this assumption can be removed at the cost of increasing the approximation ratio by one.

Let K be the set of start times for the slots. For every job j, let $P(j) = K \cup [r_j, d_j - p_j]$ be the set of time-points in which job j can start. For any job j, we define $|P(j)| + |P(j)|b_j$ variables: $y_{j,t}$ for every $t \in P(j)$, and $x^i_{j,t}$ for every $t \in P(j)$ and $1 \leq i \leq b_j$. The variable $y_{j,t}$ indicates whether job j was selected to be processed starting at time point t. The variable $x^i_{j,t}$ indicates whether the i-th stage of job j that begins at time point t is processed. In any integral solution, $y_{j,t} = \bigwedge_{i=1}^{b_j} x^i_{j,t}$.

For every time-point $\ell \in K$, let $I(\ell)$ be the set of intervals j^i_t that are active at time ℓ. Formally,

$$j^i_t \in I(\ell) \text{ if and only if } t + \sum_{k=1}^{i-1} p^k_j < \ell \leq t + \sum_{k=1}^{i} p^k_j,$$

where j^i_t denotes the i-th stage of job j's instance when starting at time t.

We solve $LP_{\mathcal{D}}$, the linear programming relaxation of an integer program describing our problem. The objective function is to maximize the profit from jobs selected to be processed. The set of constraints (1) guarantee that the resource utilization does not exceed the available amount. Constraints (2) guarantee that every job is processed at most once. Constraints (3) reflect the fact that a job is considered processed only if all of its stages are processed.

$$(LP_{\mathcal{D}}) \qquad \max \sum_{j=1}^{n} \sum_{t \in P(j)} w_j \cdot y_{j,t}$$

$$\text{subject to:} \quad \sum_{j^i_t \in I(\ell)} x^i_{j,t} \cdot c^i_j \leq B \quad \forall\ 0 \leq \ell \leq |K| \quad (1)$$

$$\sum_{t \in P(j)} y_{j,t} \leq 1 \qquad \forall j \quad (2)$$

$$y_{j,t} \leq x^i_{j,t} \qquad \forall j, i, t \quad (3)$$

$$0 \leq y_{j,t} \leq 1 \qquad \forall j, t \quad (4)$$

$$0 \leq x^i_{j,t} \leq 1 \qquad \forall j, i, t \quad (5)$$

Before presenting the rounding technique of an optimal fractional solution, we note the following property.

Observation 2.2: There exists an optimal solution for $LP_{\mathcal{D}}$ in which for all j, t, $y_{j,t} = x^1_{j,t} = x^2_{j,t} = \cdots = x^{b_j}_{j,t}$.

Proof: For any feasible solution x, y, by the set of constraints (3), it holds that $y_{j,t} = min(x^1_{j,t}, x^2_{j,t}), \cdots, x^{b_j}_{j,t}$. By setting $x^i_{j,t} = y_{j,t}$ for all $x^i_{j,t}$, there is no harm to the objective function, and all the constraints are satisfied. ■

Given an optimal solution for $LP_{\mathcal{D}}$, satisfying the conditions of Observation 2.2, we first round down every variable $x^i_{j,t}$ to the form

$$x^i_{j,t} = \frac{a^i_{j,t}}{n^2|K|^2},$$

$1 \leq a^i_{j,t} \leq n^2|K|^2$. We then define the following set of intervals: Every variable $x^i_{j,t}$ contributes $a^i_{j,t}$ copies of the interval j^i_t. The set of constraints (1) in $LP_{\mathcal{D}}$ implies the following.

Observation 2.3: For every time point t, the total number of copies of intervals active at time t is at most $Bn^2|K|^2$.

By Observation 2.2, we assume that $a^1_{j,t} = a^2_{j,t} = \cdots = a^{b_j}_{j,t}$. We denote all these values by $a_{j,t}$, and refer to the intervals as a_{j_t} copies of job j_t.

We now show that it is possible to split the set of duplicated intervals into $A = 5.15n^2|K|^2$ feasible schedules. In order to count the number of feasible schedules, we color the intervals such that every color-set induces a feasible schedule: for every instance j_t of job j, all intervals $j^i_t \in j_t$ receive the same color. In addition, for every time point t, and for every color γ, the total resource demand of intervals that are colored γ and are active at time t is at most B. We now describe the coloring procedure in detail.

Greedy Interval Coloring: Sort the duplicated intervals in non-decreasing order by their start times, breaking ties arbitrarily. Consider the intervals in the sorted order. We describe the color selection for intervals corresponding to the first stage of each job, since whenever a first-stage interval is colored, the same color is given to all subsequent stages of this instance.

1) Let j^1_t be the next uncolored interval. Color j^1_t in the first legal color. A color γ is not legal for job j^1_t either if another copy of j^1_t

is already colored γ (recall that there are $a_{j,t}$ copies of j_t^1), or if one of the copies of j_r^1 for $r < t$ is colored γ, or if coloring it with color γ will violate the resource availability constraint.

2) Color the intervals $j_t^2, \ldots, j_t^{b_j}$ corresponding to the same instance as j_t^1 in the same color as j_t^1.

We show that the greedy interval-coloring procedure produces a legal coloring using at most $A = 5.15n^2|K|^2$ colors.

Lemma 2.4: The greedy interval-coloring procedure outputs a coloring in which the intervals in every color-set form a feasible solution for the *Dwindling-Jobs Scheduling* problem.

Proof: Consider the coloring of a copy of j_t^1. If the instance is colored by a new color, then in Step 2 the remaining stages of this instance are also colored in the same new color and the new color-set contains a single job, which is clearly a valid schedule. If the instance is colored by color γ that is used already by other job-instances, then since γ is valid for this copy of j_t^1, no other instance of this job - in an earlier time window or another copy of j_t^1, is colored γ. Also, the addition of c_j^1 to the total resource demand of intervals having color γ that are active at time t does not exceed the resource capacity B. Since resource demands are dwindling and jobs are colored in non-decreasing order of their start times, the total resource demand by jobs colored γ at any time point $t' > t$ is at most $B - c_j^1$. Thus, it is safe to color with γ also the intervals $j_t^2, \ldots, j_t^{b_j}$ of this job instance. ■

Lemma 2.5: The greedy interval-coloring procedure uses at most $A = 5.15n^2|K|^2$ colors.

Proof: We count the maximal number of illegal colors when examining the interval j_t^1. We distinguish between two types of intervals.

(i) Interval j_t^1 with resource demand $c_j^1 < B$. We show that this interval will find a valid color in the set of the first $n^2|K|^2 + \frac{Bn^2|K|^2}{B-c_j^1}$ colors. First, combining constraints (2) and (3), with the fact that $x_{j,t}^1 = \frac{a_{j,t}^1}{n^2|K|^2}$, we have that at most $n^2|K|^2$ colors are assigned already to intervals corresponding to job j. Second, by Observation 2.3, at most $\frac{Bn^2|K|^2}{B-c_j^1}$ colors are illegal due to the resource availability constraint.

This implies that only intervals with capacity at least c_j^1 will be colored with a color indexed z, such that

$$z \geq n^2|K|^2 + \frac{Bn^2|K|^2}{B - c_j^1}. \tag{6}$$

Let c_z be the minimal capacity of the first stage of a job in color-set z. For the color-set with index z that satisfies (6), we have

$$c_z \geq c_j^1(z) = B - \frac{Bn^2|K|^2}{z - n^2|K|^2} \tag{7}$$

Therefore, if the color set z is not empty, the minimal resource demand of some interval in z is at least the right hand side of Equation (7). By setting $z = 2n^2|K|^2 + 1$ in Equation (7) we get that $c_z \geq B - \frac{Bn^2|K|^2}{z-n^2|K|^2} > 0$.

(ii) Interval j_t^1 with resource demand $c_j^1 = B$. Such an interval will have to get a new color δ. By the above discussion, we distinguish between two cases:
(1) If δ is in the range $[1, z]$, where $z = 2n^2|K|^2 + 1$, we are done. Otherwise,
(2) Let c_{min}^1 be the minimal resource demand of a first stage of any of the input jobs. If $\delta > z$ then in every color-set in the range $1, 2, \cdots, z$ there is at least one interval with demand at least c_{min}^1. We bound the number of color-sets larger than z that are blocked for j_t^1.

In the worst case, each of the $2n^2|K|^2 + 1$ first color-sets contains a single interval with demand c_{min}^1, and j_t^1 cannot be added to none of these color-sets. We turn to examine how many color-sets with index larger than $2n^2|K|^2+1$ are non-empty, assuming each of the color-sets contains a single interval with the minimal demand, c_{min}^1. Let $C_{max}n^2|K|^2$ be the index of the maximal color-set using this assumption. We want to find the maximal value C_{max} can get.

Suppose that every color-set in the range $1, \cdots, C_{max}n^2|K|^2$ contains a single interval with minimal resource demand. Also, all the color-sets with index larger than $C_{max}n^2|K|^2$ are empty, except maybe for the first $n^2|K|^2$ color-sets with index larger than $C_{max}n^2|K|^2$, which are color-sets with self conflict. By self-conflicts we refer to all other copies of j_t^1 that need to receive distinct colors. The number of such copies is at most $n^2|K|^2$. This adds at most $n^2|K|^2$ colors. Hence, the maximum index for a non-empty color-set is at most $C_{max}n^2|K|^2+n^2|K|^2$. Now, we find an upper bound for C_{max}. After calculating the integral of the minimal demand in every color-set

between $2n^2|K|^2+2$ and $C_{max}n^2|K|^2$, that is:

$$\int_{2n^2|K|^2+2}^{C_{max}n^2|K|^2}(B-\frac{Bn^2|K|^2}{z-n^2|K|^2})dz, \quad (8)$$

we get that $C_{max} \leq 4.15$.

Thus, the interval j_t^1 will find a valid color whose index is at most $(C_{max}+1)n^2|K|^2 = 5.15n^2|K|^2$. ■

Note that $OPT = \sum_{j,t} w_j y_{j,t} = \sum_{j,t} w_j x_{j,t}^1$. In addition, $x_{j,t}^i \leq \frac{a_{j,t}^i+1}{n^2|K|^2}$ which implies that $a_{j,t}^i \geq x_{j,t}^i n^2|K|^2 - 1$. Therefore,

$$\begin{aligned}\sum_{j,t} w_j a_{j,t}^i &\geq \sum_{j,t} w_j x_{j,t}^i n^2|K|^2 - 1 \\ &\geq OPTn^2|K|^2 - n|K|b_j. \quad (9)\end{aligned}$$

By Lemma 2.5, there exists a group constrained coloring that uses at most $5.15n^2T^2$ colors. Since each color-set defines a feasible schedule, the coloring induces $5.15n^2|K|^2$ feasible schedules. By Equation (9), at least one of these schedules has weight at least $OPT/5.15$. ■

III. Empirical results

We have implemented the LP-based algorithm and compared it with some natural heuristics. The general scheme for all the heuristics is given below. The set S describes the formed schedule; S is a collection of pairs $\{j,t\}$ where j is a job and t is the time $r_j \leq t \leq d_j - p_j$ in which j starts its execution.

General Greedy Scheme

(1) $S = \emptyset$
(2) Use a certain rule to sort the jobs; consider the jobs in this order.
  a) Let j be the next job in the sorted list. If j can be added to S, schedule j in the earliest feasible time point in $[r_j, d_j - p_j]$ and update S.
1) Return S.

For a job j, the *area* of j is the total resource capacity j requires along its stages. Formally, $area(j) = \sum_{k=1}^{b_j} p_j^k \cdot c_j^k$. We have implemented Step (2) in four different ways: sorting the jobs in (i) non-increasing order of weights, (ii) non-increasing order of the ratio weight/area, (iii) non-decreasing order of area, and (iv) non-decreasing order of release times.

The heuristics resulting from application of the above rules are simple and natural. Rule (i) gives high priority to jobs associated with high profits; Rule (ii) takes into account the amount of resource required to achieve the profit; Rule (iii) gives high priority to jobs with modest resource demand, and Rule (iv) corresponds to an on-line system that admits jobs greedy upon arrival.

In the first experiment we compared the performance of the algorithms, as a function of the *density* of the input. Recall that the resource is available during the time interval $[0,T]$. The density of an input is defined as the ratio between the total resource capacity (area) of all jobs and the total availability of the resource. Formally, $\frac{1}{B \cdot T} \cdot \sum_{j=1}^{n} \sum_{k=1}^{b_j} p_j^k \cdot c_j^k$.

Since instances can be of variable densities, even if the number of jobs is small, in this experiment, we were able to calculate and compare the heuristics also with an optimal solution. For each value of density in $\{1/2, 4/3, 3/2, 5/2\}$, we used the same set of jobs – with the same values of $\langle (p_j^1, c_j^1), \ldots, (p_j^{b_j}, c_j^{b_j}) \rangle$. However, the values of T, as well as the values of job release-times and deadlines, were *scaled* to fit the required density. Specifically, the values of T in the different runs were $6, 10, 20$ and 30. The resource availability was set to $B = 20$ in all runs.

The generation of a single job consists of the following steps: first, the number of stages is drawn randomly (in this experiment, b_j was a random integer between 1 and 6), then, b_j values in the range $[0, B/2]$ were drawn uniformly and independently. These numbers were sorted in a descending order - to get the dwindling values of $c_j^1, \ldots, c_j^{b_j}$. Then, we randomly set each stage length p_j^k to be either 1 or 2. The release-time and deadlines were set to random points in $[0, T - p_j]$ and $[r_j + p_j, T]$, respectively. Finally, jobs' weight was drawn from the range $1, \ldots, 50$. All values were chosen uniformly and independently.

We run the experiment with each density value 10 times, each time with a new set of 10 jobs. The results (average of all runs) are presented in Fig. 2. In each chart, the rightmost column corresponds to the performance of an optimal solution. We scaled the results such that $OPT = 1$, and the performance of each of the other algorithms is given as a fraction smaller than 1. As can be seen in the figures, the LP-based approximation algorithm performs close to optimal for all density values. It outpreforms most of the heuristics and preforms much better than its

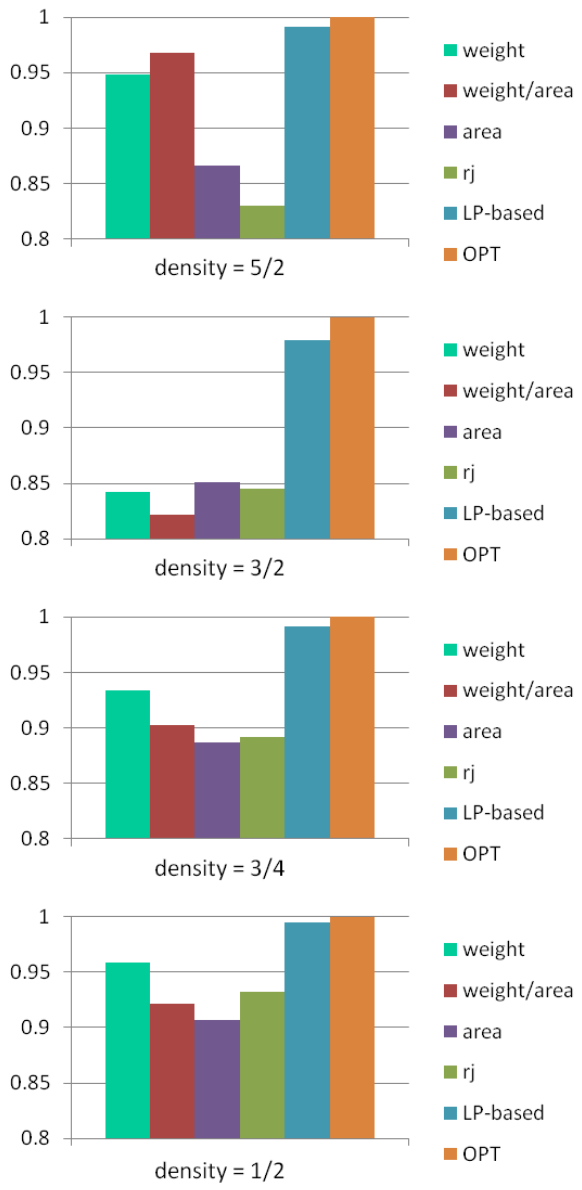

Fig. 2. The performance of all algorithms compared to OPT, as a function of the input's density

theoretical approximation-ratio. It can also be seen that the most challenging density for all four heuristics is $3/2$. In sparser inputs, the resource is relatively available, thus, it is possible to schedule more jobs and even simple heuristics perform well. High-density instances are on one hand more challenging, because every inclusion of a job affects the availability of the resource along a significant portion of its active time, but on the other hand, rejection of jobs can be compensated by inclusion of other jobs that require the resource in an overlapping time window. Another interesting conclusion is that the weight/area heuristic improves its performance as the density increases, and the r_j heuristic reduces its performance significantly when the density increases.

In the second experiment we compared the performance of the heuristics on variable-size inputs. For each $n \in \{50, 200, 500, 1000\}$ we run the four heuristics with the following parameters: The machine was available during the time interval $[0, 50]$ and its capacity was set to $B = 50$. The jobs were created randomly as described in the first experiment with $b_j \in \{1, \ldots, 8\}$. The input for any $n > 50$ was created by adding jobs to the previous input. For example, the input for $n = 500$ consists of the input for $n = 200$ and 300 newly created jobs. Each experiment was run 50 times. Fig. 3 presents the results of this experiment (average of all runs).

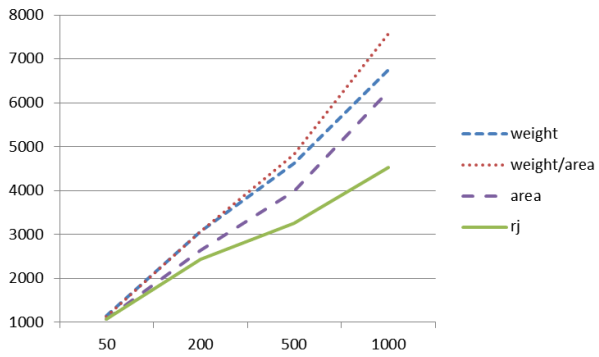

Fig. 3. Total profit as a function of the input size.

As illustrated in Fig. 3, the *weight/area*-heuristic outpreforms the other heuristics, and its advantage is more significant as the number of jobs increases. On the other hand, the r_j-heuristic - which is the most natural on-line algorithm, achieves only half the profit of *weigth/area* on large instances. This result hints that the online version of the problem, in which jobs must be rejected or accepted upon arrival, is much more difficult than the offline setting, in which it is possible to select the subset of processed jobs only after all requests are known.

In the third experiment we measured the affect of keeping the system's density while increasing the number of jobs and the jobs' processing-interval length $(d_j - r_j)$. For each $x = \{25, 100, 250, 500\}$, we run the four heuristics with $n = 2x$ jobs, where the machine is available during the time interval $[0, x]$ and its capacity is $B = 50$. We note that since the ratio between n and T remains the same ($n = 2T$), the expected density of the input remains the same; however, since the release-time of a job is a random point in $[0, T - p_j]$, and the deadline is a random point in $[r_j + p_j, T]$, when x increases, also the gap between the release-time and deadline is increased. Thus, the system has more flexibility in assigning the jobs.

The jobs were created as described in the first experiment. The input for any $n > 50$ was created by expanding the previous input - adjust it to the new value of T, and adding new jobs. For example, the input for $n = 500$ consists of the input for $n = 200$ (same values of $\langle (p_j^1, c_j^1), \ldots, (p_j^{b_j}, c_j^{b_j}) \rangle$ with new random r_j and d_j values), and 300 newly created jobs. Each experiment was run 50 times. Fig. 4 presents the results (average of all runs). The y-axis gives the average profit per time slot, that is, the total weight of processed jobs divided by T.

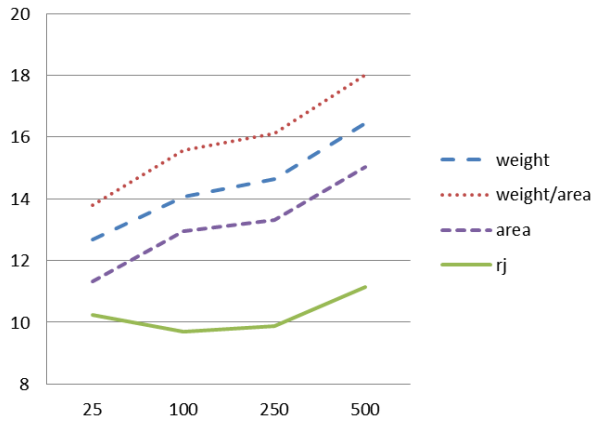

Fig. 4. Profit per time-slot as a function of resource active time.

As illustrated in Fig. 4, all heuristics, except for the r_j-heuristic benefit from extending the active time of the machine and the flexibility in the assignment. This flexibility is not necessarily positive for the r_j-heuristic since it enables accepting early-released jobs that consume the machine's resource and avoid accepting later, possibly more profitable, jobs.

In the last experiment, we measured the influence of the number of stages on the performance of the heuristics. We fixed $T = 600$ and $B = 20$. For each of the $n = 200$ jobs, the job's area was fixed for all the experiments. The expected area of the jobs was set such that the expected density of the inputs is $3/2$. In the first set of runs, with $b_j = 2$, we generated the jobs as in the first experiment. For the other sets of runs, with $b_j = 3, 6, 10$, jobs with the same area were used, that is, the number of stages increases while the length and capacity of each stage decrease. Thus, the heuristics had to cope with 'shallower and longer' stairs. Each experiment was run 50 times. The results are presented in Fig. 5.

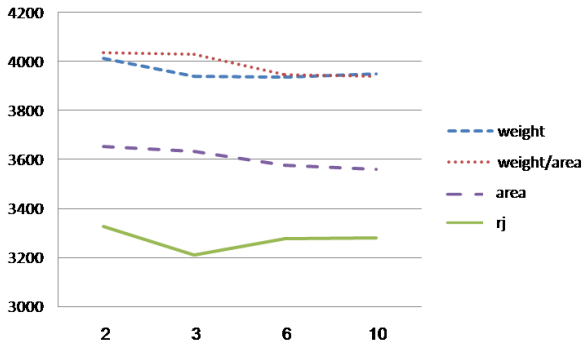

Fig. 5. Total profit as a function of the number of stages in a job.

As illustrated in Fig. 5, the heuristics that prefer jobs with high weight and high weight/area perform better than the other two heuristics. When examining each heuristic separately we see that it achieves almost the same profit (a difference of $2 - 3\%$) independent of the number of jobs' stages, with a slight better utilization when jobs consist of only two stages.

IV. Conclusions and Open Problems

We presented theoretical and experimental results for the *Dwindling-Jobs Scheduling* problem. Our LP-based approximation algorithm yields a constant factor approximation ratio. An empirical study shows that this algorithm outperforms several natural greedy heuristics, and for realistic inputs it is nearly optimal.

Although the studied heuristics preform well on general arbitrary inputs, none of them guarantees a constant-factor approximation to the optimal profit. The following simple example shows that the heuristics

can fail already for inputs where each job consists of a single stage. Consider an input having one job with a single 'long and thin' stage, i.e., with high processing time and small capacity, and r additional jobs, each consisting of a single 'short and fat' stage. One can set the job parameters such that all four heuristics select the first job, and therefore will not be able to schedule any of the other jobs. On the other hand, an optimal solution selects all but the first job. By setting $w_1 = 1$ and $w_j = 1 - \varepsilon$ for $2 \leq j \leq r + 1$, we get that $OPT = r(1-\varepsilon)$, while all the heuristics yield a profit of 1.

In general, our experiments reveal that simple heuristics perform very well on relatively 'uniform' instances – when jobs have similar 'shapes' – similar number of stages, similar dwindling pace, and similar areas. Systems that are expected to provide service to a sequence of homogeneous requests can achieve close to optimal utilization when jobs are selected by simple greedy rules, that are based on job weights or weight/area ratios. On the other hand, when the goal is to maximize resource utilization in a system servicing heterogeneous requests, the performance of the heuristics may be significantly worse than the optimal, and a more sophisticated scheme, such as the LP-based algorithm, should be used.

We also conclude that increasing the average number of stages in a job while keeping its total resource demand does not make the instance harder to solve. That is, all heuristics perform almost the same (with up to $2 - 3\%$ change in the resource utilization) on instances with fixed area and variable number of stages for the jobs.

Our LP-based approximation algorithm assumes that job stages must be performed contiguously and that job resource demands dwindle as they make progress towards completion. An interesting avenue for future work is to study whether a constant-factor approximation algorithm exists if one of these assumptions is relaxed. Namely, (i) the sequence of resource-demands of a job is not necessarily dwindling. We note that even the case of monotone-request jobs (a combination of dwindling-request or expanding-request jobs) is non-trivial and requires new tools. (ii) jobs can be preempted – in general or only at end-of-stage time points.

Another direction to explore is the online version of the problem, in which jobs must be admitted or rejected upon arrival. Using an instance similar to the one described above, it is easy to see that no algorithm achieves a constant competitive ratio for this problem; this calls for a study of online algorithm on restricted inputs, or with resource augmentation.

REFERENCES

[1] A. Bar-Noy, R. Bar-Yehuda, A. Freund, J. Naor, and B. Schieber. A unified approach to approximating scheduling and allocation of resources. *J. of the ACM*, 48(5):1069–1090, 2001.

[2] A. Bar-Noy, S. Guha, J. Naor, B. Schieber. Approximating the throughput of multiple machines in real-time scheduling. *SIAM J. Comput.* 31(2): 331–352 (2001).

[3] A. Bar-Noy, R.E. Ladner, T. Tamir. Optimal delay for media-on-demand with pre-loading and pre-buffering. *Theoretical Computer Science* 399(1), 3–11, 2008.

[4] R. Bar-Yehuda, D. Rawitz. Using fractional primal-dual to schedule split intervals with demands. *Discrete Optimization* 3(4): 275–287 (2006).

[5] P. Brucker. Scheduling Algorithms, 4th ed. Springer, Heidelberg, 2004.

[6] N. Jain, I. Menache, J. Naor, J. Yaniv. A truthful mechanism for value-based scheduling in cloud computing. *SAGT*, 2011.

[7] H. S. Malvar. Fast progressive image coding without wavelets. *IEEE Data Compression Conference*, Utah, 2000.

[8] C. A. Phillips. Personal communication.

[9] C. A. Phillips, R. N. Uma, J. Wein: Off-line admission control for general scheduling problems. In *SODA*, pp. 879–888, 2000.

[10] F. C. R. Spieksma, On the approximability of an interval scheduling problem, *J. of Scheduling*, vol. 2 pp. 215–227, (1999).

[11] A. Zambelli. IIS Smooth streaming technical overview. Microsoft Corporation, March 2009.