Approximation Algorithms for Scheduling with/without Deadline Constraints where Rejection Costs are Proportional to Processing Times

Olivier Beaumont, Rémi Bouzel, Lionel Eyraud-Dubois, Esragul Korkmaz, Laércio Lima Pilla, Alexandre van Kempen

Approximation Algorithms for Scheduling with/without Deadline Constraints where Rejection Costs are Proportional to Processing Times

Olivier Beaumont, Rémi Bouzel, Lionel Eyraud-Dubois, Esragul Korkmaz,
Laércio Lima Pilla, Alexandre Van Kempen

***Abstract*—We address two offline job scheduling problems, where jobs can either be processed on a limited supply of energy-efficient machines on the edge, or offloaded to an unlimited supply of energy-inefficient machines on the cloud (called *rejected* in our context). The goal is to minimize the total energy consumed in processing all tasks. We consider a first scheduling problem with no due date (or deadline) constraints, and we formulate it as a scheduling problem with rejection, where the cost of rejecting a job is directly proportional to its processing time. We introduce a novel $\frac{5}{4}(1+\epsilon)$ approximation algorithm $\mathcal{BEKP}$ by associating it with a Multiple Subset Sum problem for this version. Our algorithm is an improvement over the existing literature, which provides a $(\frac{3}{2}-\frac{1}{2m})$ approximation for scenarios with arbitrary rejection costs. In the second scheduling problem, jobs have due date (or deadline) constraints, and the goal is to minimize the weighted number of late jobs. In our context, if a job is late, it is offloaded (rejected) to an energy-inefficient machine on the cloud, which incurs a cost directly proportional to its processing time of the job. We position this problem in the literature, and introduce a novel $(1-\frac{(m-1)^m}{m^m})$-approximation algorithm $\mathcal{MDP}$ for this version, where we got our inspiration from an algorithm for the interval selection problem with a $(1-\frac{m^m}{(m+1)^m})$ approximation ratio for arbitrary rejection costs. We evaluate and discuss the effectiveness of our approaches through a series of experiments, comparing them to existing algorithms.**

Keywords: Scheduling with rejection, computing continuum, approximation algorithm, energy minimization

I. Introduction

CLOUD computing has been a central topic of research for decades, with studies revealing the trade-offs involved in offloading computational tasks to machines hosted in distant datacenters [1]. To address these challenges, fog and edge computing infrastructures have been introduced, placing computing and storage resources closer to end-users [2]. By positioning resources at the network edge, these infrastructures help reduce latency and alleviate network congestion, as data can be processed locally before being sent to the cloud. Harnessing cloud, edge, and fog computing resources enables the creation of a unified infrastructure that optimizes these resources usage across all layers. So-called computing continuum systems [3] can provide optimal balance between resource utilization and performance by dynamically allocating computations to the most appropriate resource within this continuum.

Beyond improving efficiency, such computing continuum systems offer opportunities for better energy management by exploiting their geo-distributed nature. Computing resources located at the network edge allow for the reuse of energy, particularly in the form of heat, in areas where it is needed. For example, some providers, like Qarnot Computing[1] [4], deploy computing units in water boilers, generating hot water for heating networks, swimming pools or facilities while running computations. This approach significantly reduces energy waste by eliminating the need for cooling infrastructure and lowering carbon emissions. However, there might not be enough of these energy-efficient edge resources for large computations, necessitating the offloading of excess workloads to more conventional, less energy-efficient cloud providers when necessary.

This paper draws inspiration from such architectures, where computational tasks can either be sent to a limited set of energy-reusing edge resources or to a more abundant pool of high-carbon-footprint resources in traditional datacenters. The challenge lies in job scheduling, specifically in deciding which jobs to reject from the energy-efficient resources and how to minimize the energy impact of processing these rejected jobs on traditional cloud systems. We explore scheduling problems where job rejection is an option, and the cost of rejection is directly proportional to the job's processing time. This concept of *scheduling with rejection* has practical applications in a variety of real-world scenarios [5], [6], and this paper investigates it within the context of energy-efficient computing continuum systems.

In this paper, we propose scheduling algorithms for two scenarios where the rejection cost of a job is proportional to its processing time. This is an extension of our previous work [7] that focused only on the scenario where jobs have no deadlines. There, we presented a $\frac{5}{4}(1+\epsilon)$-approximation algorithm for any positive ϵ, whereas the best-known algorithm for this problem (with arbitrary rejection costs) had an approximation factor of $\frac{3}{2}$ [8]. In this extension, we also consider the scenario where jobs have deadlines, and the objective is to maximize the occupation of edge boilers while respecting the jobs' deadlines (i.e., minimize the total rejection cost). We present a new $(1-\frac{(m-1)^m}{m^m})$-approximation algorithm based on iteratively

O. Beaumont, L. Eyraud-Dubois, E. Korkmaz and L. Lima Pilla are with Univ. Bordeaux, CNRS, Bordeaux INP, Inria, LaBRI, UMR 5800, F-33400 Talence, France. R. Bouzel and A. Van Kempen are with Qarnot Computing, Montrouge, France.
E-mail: esragul.korkmaz@inria.fr

[1] https://qarnot.com/

scheduling jobs to each individual machine optimally, which we compare to the $(1 - \frac{(m)^m}{(m+1)^m})$-approximation algorithm proposed by Berman and DasGupta [9]. We can summarize our main contributions as follows:

- we present a $\frac{5}{4}(1 + \epsilon)$ approximation algorithm for scheduling with rejection without deadlines [7] (Section III);
- we propose a $(1 - \frac{(m-1)^m}{m^m})$-approximation algorithm for scheduling with rejection with deadlines (Section IV);
- we evaluate the quality of their schedules against state-of-the-art solutions using simulation (Section V); and
- we discuss ideas on how to improve the current approximation ratios of our algorithms (Section VI).

The remainder of the paper includes a review of related work in Section II, and concluding remarks and perspectives in Section VII.

II. Related Work

For a comprehensive review of the literature on scheduling with rejection problems, we refer the reader to the survey papers by Slotnick [10], Shabtay et al. [5] and Adamu and Adewumi [11].

Bartal et al. [12] introduced the problem of scheduling with job rejection, with arbitrary rejection costs (penalties). Their objective is to minimize the makespan of accepted jobs plus the sum of penalties associated with rejected jobs. For the online setting, they present a $(1 + \phi)$-competitive algorithm, where ϕ is the golden ratio. For the offline setting, they provide a fully polynomial approximation algorithm for fixed m and a polynomial approximation algorithm for arbitrary m. In particular, they propose a $(2 - \frac{1}{m})$ approximation algorithm for the offline problem with $\mathcal{O}(n \log n)$ complexity.

Ou et al. [13] improve the approximation of Bartal et al. with a heuristic that achieves a worst-case bound of $\frac{3}{2} + \epsilon$ with a complexity of $\mathcal{O}(n \log n + \frac{n}{\epsilon})$. Liu and Lu [8] provide a $(\frac{3}{2} - \frac{1}{2m})$ approximation algorithm with $\mathcal{O}(n^3 log n)$ complexity, improving on the work of Ou et al. [13]. In addition to this solution for the identical release date problem, Liu and Lu also present solutions for the single machine and parallel machine problems in the presence of release dates.

In this paper, we present a heuristic to improve on the $(\frac{3}{2} - \frac{1}{2m})$ approximation provided by Liu and Lu [8]. In our setting, the rejection cost is proportional to the processing time of the jobs. We relate the problem to a Multiple Subset Sum Problem (MSSP) and build on ideas of Caprara et al. [14] to obtain a $\frac{5}{4}(1 + \epsilon)$ approximation algorithm.

The problem of scheduling with rejection has also been studied, with the goal of optimizing the sum of the weighted completion times of scheduled jobs plus the sum of the penalties for rejected jobs. Engels et al. [15] propose general techniques for solving offline scheduling with rejection problems with this objective. Epstein et al. [16] focus on the single-machine online problem, where jobs have unit processing times and the weight of each job's completion time is equal to 1. Liu [17] considers the single-machine problem with partial rejection and provides both polynomial-time and pseudopolynomial-time optimal algorithms.

In their work, Mor and Shabtay [18] explore two different objectives in scheduling with rejection for single-machine problems. One is to minimize the sum of the total late work and the total rejection cost, while the other is to focus only on minimizing the total rejection cost, providing an upper bound on the total late work.

In the context where jobs have deadlines, the corresponding problem is similar to scheduling to minimize the weighted number of late jobs (where weights can be seen as penalties), where rejected jobs are processed at the end of the schedule rather than on additional cloud resources. A survey by Adamu and Adewumi [11] provides a comprehensive literature review on this topic. Another related topic is Interval Scheduling (see [19] for a survey), where each job is given one (or more) possible start times, and the problem is to select a subset of jobs that can be scheduled together. Most of the literature assumes that jobs also have release dates.

The problem of minimizing the number of late jobs (a special case where all jobs have the same penalty cost) has been studied from a practical perspective by Ho et al. [20], who proposed a number of heuristics, and from a theoretical perspective by Briskorn et al. [21], who proved several approximation ratios for these heuristics.

For arbitrary weights, most of the literature focuses on complex, exponential-time exact algorithms. Chen et al. [22] formulate the problem as an Integer Linear Programming problem and rely on Dantzig-Wolfe reformulation and branch-and-bound techniques to solve it optimally. Some approximation algorithms have also been proposed: Bar-Noy et al. [23] developed two algorithms with approximation ratio $1 + \left(\frac{m}{m+1}\right)^m$, one for unitary weights based on a greedy algorithm and the other for arbitrary weights based on rounding a Linear Programming relaxation. Berman and DasGupta [9] independently proposed an algorithm with the same approximation ratio based on a greedy algorithm for the interval scheduling problem on a single machine.

In this paper, we consider a special case of the problem where all jobs have the same release date and where the penalty cost of each job is proportional to its processing time. We obtain an improved approximation ratio $1 + \left(\frac{m-1}{m}\right)^m$, which holds for arbitrary job weights.

III. Scheduling with Rejection without Deadlines

In this section, we consider the problem where the jobs have *no* due dates (deadlines). The problem formulation and the necessary notations are provided in Section III-A. In Sections III-B and III-C we describe our solution $\mathcal{BEKP}$, provide a proof of its approximation ratio, and discuss its computational complexity.

A. Problem Formulation

We consider a scheduling problem where a set of non-preemptive jobs J are to be scheduled on m identical machines. Each job i is characterized by its processing time p_i and can either be processed on one of the energy-efficient machines of the edge boilers or rejected at a cost $\rho \cdot p_i$. This rejection cost represents the cost of offloading the job to

TABLE I: Notation employed for the scheduling problem without deadlines.

m	Number of machines
n	Number of jobs
J	Set of jobs
p_i	Processing time of job i for $i \in \{1, 2, ..., n\}$
W	Area of all jobs in J ($W = \sum_{i \in \{1,2,...,n\}} p_i$)
ρ	Rejection cost coefficient
$C^{\mathcal{S}}$	Makespan of the accepted jobs in schedule $\mathcal{S}$
$A^{\mathcal{S}}$	Area of the accepted jobs in schedule $\mathcal{S}$
$R^{\mathcal{S}}$	Area of the rejected jobs in schedule $\mathcal{S}$
$Z^{\mathcal{S}}$	Cost of the schedule $\mathcal{S}$: $Z^{\mathcal{S}} = mC^{\mathcal{S}} + \rho R^{\mathcal{S}}$
OPT	An optimal schedule which minimizes the cost
$R^*(T)$	Minimum possible area of rejected jobs within makespan T ($min_{\mathcal{S},\, C^{\mathcal{S}} \leq T} R^{\mathcal{S}}$)

another computing resource on the cloud, which are assumed to be in unlimited supply. Table I provides a summary of the notation used in this section.

A solution $\mathcal{S}$ specifies (i) whether each job is accepted or rejected, and (ii) assigns each accepted job i to a machine $j \leq m$. The makespan $C^{\mathcal{S}}$ of a solution is the maximum load on any energy-efficient machine, $C^{\mathcal{S}} = \max_{j \leq m} \sum_{i \text{ assigned to } j} p_i$. Since jobs are independent and available from the start, knowing the set of jobs running on each machine is enough to determine the makespan, since their relative execution order does not affect it. We denote by $R^{\mathcal{S}}$ the total processing time (or *area*) of the rejected jobs in $\mathcal{S}$: $R^{\mathcal{S}} = \sum_{i \text{ rejected}} p_i$. Our objective is then to minimize the cost $Z^{\mathcal{S}}$, defined as the sum of the utilization of any energy-efficient machine plus the overall rejection cost:

$$Z^{\mathcal{S}} = m \cdot C^{\mathcal{S}} + \rho \cdot R^{\mathcal{S}}. \tag{1}$$

Given a target makespan T, we denote with $R^*(T)$ the smallest possible area of rejected jobs among the solutions of makespan at most T. More formally, $R^*(T) = \min_{\mathcal{S}, C^{\mathcal{S}} \leq T} R^{\mathcal{S}}$. This definition leads to the following result:

Lemma 1. *For two values T_1 and T_2, if $T_1 \leq T_2$, then $R^*(T_2) \leq R^*(T_1)$.*

B. Scheduling with a Bound on Makespan

In this section, we present an algorithm called $FillMaxArea$ which, given a set of jobs J, a number of machines m and a makespan bound T, outputs a solution $\mathcal{S}$ with $C^{\mathcal{S}} \leq \frac{5}{4}T$ and $R^{\mathcal{S}} \leq R^*(T)$.

1) Job Types: The problem of scheduling with rejection is related to the Multiple Subset Sum Problem (MSSP), where the goal is to allocate a set of n items with weights w_i into m identical bins, each with a positive capacity c, so as to maximize the total allocated weight. Caprara et al. [14] proposed an algorithm for MSSP that guarantees to achieve at least a fraction of $\frac{3}{4}$ of the maximum possible weight. This algorithm works by excluding small items from the instance, and by classifying the other items in different categories depending on their weights.

For scheduling with rejection, when the weight of a job is proportional to its processing time, the problem of assigning jobs within a given makespan T is actually equivalent to MSSP. However, we are interested in providing an approximate solution in a different way: instead of selecting a set of jobs that fits within T with a $\frac{3}{4}$ guarantee with respect to the accepted weight, we aim to select a set of jobs that fits within $\frac{5}{4}T$ and whose total weight is at least as much as the best possible weight that can be accepted within time T.

For this purpose, we group jobs from J according to their processing time, with cutoff values (with different cutoff values compared to the solution by Caprara et al. [14]), as described below:

$$G = \{i \mid \frac{3}{4}T < p_i \leq T\} \qquad N_1 = \{i \mid \frac{1}{2}T < p_i \leq \frac{3}{4}T\}$$
$$N_2 = \{i \mid \frac{3}{8}T < p_i \leq \frac{1}{2}T\} \quad N_3 = \{i \mid \frac{1}{4}T < p_i \leq \frac{3}{8}T\}$$
$$P = \{i \mid p_i \leq \frac{1}{4}T\}$$

Jobs in G, N_1, N_2, and N_3 are called *long* jobs, while jobs in P are called *short* jobs. We define a *combination* $(Set_1, Set_2, ...)$ as a mapping of jobs to a machine, where exactly one job from each set (a set can occur multiple times) is scheduled onto the same machine. For example, (N_3, N_3, N_2) represents two jobs from N_3 and one job from N_2 assigned to a machine in any order.

Lemma 2. *In a schedule with maximum makespan of T, only the following combinations of long jobs are valid:*

$$(G), (N_1), (N_2), (N_3)$$
$$(N_2, N_1), (N_3, N_1), (N_2, N_2), (N_3, N_2), (N_3, N_3)$$
$$(N_3, N_3, N_2), (N_3, N_3, N_3)$$

Proof. Consider one machine in a schedule with makespan at most T. We split the proof depending on the number l of long jobs this machine handles.

For $l = 1$, any long job guarantees that the makespan bound T is respected. Thus, singleton possibilities are (G), (N_1), (N_2) and (N_3).

For $l = 2$, the combination (N_1, N_1) can not be assigned to that machine: indeed, jobs in N_1 are such that $p_i > \frac{1}{2}T$, so that processing any two of them is not feasible within makespan T. All other combinations of length 2 are valid. Thus, the possible pairs are (N_2, N_1), (N_3, N_1), (N_2, N_2), (N_3, N_2), (N_3, N_3).

For $l = 3$, if two jobs from N_2 are assigned to a machine, even assigning one extra N_3 job is not feasible since the total processing time would exceed $(\frac{3}{8}+\frac{3}{8}+\frac{1}{4})T = T$. Thus, triplets with longer jobs are not valid either. Therefore, the only possible triplets are (N_3, N_3, N_2) and (N_3, N_3, N_3). Finally, $l \geq 4$ is not feasible, since any long job has $p_i > \frac{1}{4}T$. □

In addition, it is possible to bound the maximum total processing time of any of these combinations.

Lemma 3. *The overall processing time of any valid combination described in Lemma 2 is at most $\frac{5}{4}T$.*

Proof. The proof is trivial, by enumerating all valid combinations and summing the upper bounds for each element, depending on its specific subset of long jobs. □

2) Algorithm: $FillMaxArea$ is based on these two lemmas. By guaranteeing that long jobs assigned to each machine obey one of the combinations in Lemma 2, we can guarantee that the resulting solution $\mathcal{S}$ satisfies $C^{\mathcal{S}} \leq \frac{5}{4}T$.

Our long job assignment algorithm is based on the $AssignFrom$ routine, whose pseudocode is given in Algorithm 1. Given the list of combinations and the number l of machines, $AssignFrom$ creates l machine assignments by successively picking the jobs with the largest processing times from the first combination of available jobs. For example, $AssignFrom(\{(N_2, N_1), (N_3, N_1), (N_2, N_2)\}, l)$ selects the largest job from N_2 and the largest job from N_1 until one of them is empty, and then proceeds with the combination (N_3, N_1), and so on.

Algorithm 1 $AssignFrom(combs, l)$

$Result \leftarrow \emptyset$
Remove all combinations from $combs$ where at least one set within the combination is empty
while $|Result| \leq l$ and $combs$ is not empty **do**
  Denote by $(K_1, K_2, ..., K_k)$ the first combination in $combs$
  $j_1 \leftarrow$ the largest job from K_1
  $j_2 \leftarrow$ the largest remaining job from K_2
  Continue until $j_k \leftarrow$ the largest remaining job from K_k
  $Result = Result \cup (j_1, j_2, ..., j_k)$
  Remove all combinations from $combs$ where at least one set within the combination is empty
return $Result$

$FillMaxArea$, whose pseudocode is given in Algorithm 2, starts by scheduling the long jobs first, and then completes the schedule with a greedy assignment of the short jobs without exceeding the makespan bound $\frac{5}{4}T$. To decide which long jobs to accept, we set values for l_0, l_1, l_2, and l_3 that represent the number of machines running no long job, one long job, two long jobs, and three long jobs, respectively. For each of these quadruplet $l_0, l_1, l_2, l_3)$, we use the $AssignFrom$ routine with a careful ordering of the combinations identified in Lemma 2. If we run out of jobs in this process, we discard the current solution with quadruplet $j = (l_0, l_1, l_2, l_3)$ and move on to the next possible quadruplet solution. Once an assignment has been computed for all possible quadruplets, the result of $FillMaxArea$ is the one that maximizes the total processing time of assigned jobs.

3) Proof: We now prove a guarantee on the solution produced by $FillMaxArea$: its makespan is at most $\frac{5}{4}T$, and it rejects not more work (in terms of total processing time) than any solution with makespan at most T.

Lemma 4. *For any T, let $\mathcal{S}$ be the solution obtained by $FillMaxArea(J, m, T)$. Then, $C^{\mathcal{S}} \leq \frac{5}{4}T$ and $R^{\mathcal{S}} \leq R^*(T)$.*

Proof. $C^{\mathcal{S}} \leq \frac{5}{4}T$ is a direct consequence of Lemma 3. We focus on proving $R^{\mathcal{S}} \leq R^*(T)$. Let us denote by $\mathcal{S}_0$ any solution with makespan at most T: we aim to prove that $R^{\mathcal{S}} \leq R^{\mathcal{S}_0}$, or equivalently $A^{\mathcal{S}} \geq A^{\mathcal{S}_0}$.

Algorithm 2 $FillMaxArea(J, m, T)$

Generate G, N_1, N_2, N_3 and P subsets of J
for each $j = (l_0, l_1, l_2, l_3)$ such that $l_0 + l_1 + l_2 + l_3 = m$ and $l_1 + 2l_2 + 3l_3 \leq n$ **do**
  $X_j \leftarrow \emptyset$
  $X_j \leftarrow X_j \cup AssignFrom(\{(G), (N_1), (N_2), (N_3)\}, l_1)$
  $X_j \leftarrow X_j \cup AssignFrom(\{(N_2, N_1), (N_3, N_1), (N_2, N_2), (N_3, N_2), (N_3, N_3)\}, l_2)$
  $X_j \leftarrow X_j \cup AssignFrom(\{(N_3, N_3, N_2), (N_3, N_3, N_3)\}, l_3)$
  if $l_0 + |X_j| < m$ **then**
    Discard X_j and continue
  Add jobs from P greedily (in any order) to X_j, keeping makespan $\leq \frac{5}{4}T$
$X^* = \{X_j | \max_j A^{X_j}\}$
return X^*

Lemma 2 defines the list of valid combinations for long jobs in $\mathcal{S}_0$. Let $j = (l_0, l_1, l_2, l_3)$ denote the number of machines with zero, one, two, and three long jobs in $\mathcal{S}_0$, respectively, and consider the solution X_j constructed by $FillMaxArea$ for this particular quadruplet. By construction, $A^{\mathcal{S}} \geq A^{X_j}$. Let us now prove that $A^{X_j} \geq A^{\mathcal{S}_0}$.

Let us consider the small jobs first, and distinguish between two possibilities.

- If at least one small job in P is rejected in X_j, since a small job is only rejected if it cannot be scheduled to finish before $\frac{5}{4}T$, and since the processing time of any short job is at most $\frac{1}{4}T$, this ensures that all machines have a workload of at least T. Thus, the total number of accepted jobs satisfies $A^{X_j} \geq m \cdot T \geq A^{\mathcal{S}_0}$, since $\mathcal{S}_0$ has a makespan of at most T.
- If all small jobs are accepted in X_j, then we can ignore the small jobs and we prove that $A^{X_j} \geq A^{\mathcal{S}_0}$ when restricted to long jobs. Indeed, since $\mathcal{S}_0$ cannot accept more small jobs than X_j, this will imply $A^{X_j} \geq A^{\mathcal{S}_0}$ for all jobs.

In the following, we denote as *singleton*, *pair*, and *triplet* a machine that processes one, two, and three long jobs, respectively. Both X_j and $\mathcal{S}_0$ have l_1 singletons, l_2 pairs, and l_3 triplets. In the rest of the proof, we show that $\mathcal{S}_0$ can be transformed into a solution that uses the same number of each *type* of jobs as X_j, without decreasing the total accepted area, where the type of a job refers to the specific long job subset to which it belongs. We will use two possible transformations: *replace*, where an accepted job is exchanged for a rejected job with a longer processing time, and *swap*, where two accepted jobs assigned to different machines are swapped. The first operation increases the total accepted area, while the second does not modify it. Along with the transformations, we will make sure to use only *valid* combinations from the list of Lemma 2.

We start the transformation by considering the l_1 singletons. In X_j, the jobs assigned to these machines are the l_1 longest jobs from J. We build $\mathcal{S}_1$ from $\mathcal{S}_0$ by applying a transformation for each of these longest jobs:

1) If it is rejected in $\mathcal{S}_0$, we *replace* it with the smallest job in a singleton of $\mathcal{S}_0$. This increases the total accepted area of $\mathcal{S}_0$.
2) If it is scheduled either in a pair or triplet in $\mathcal{S}_0$, we *swap* this large job with the smallest job in a singleton of $\mathcal{S}_0$.

The resulting schedule is denoted $\mathcal{S}_1$, and satisfies $(\mathcal{P}_1)$: its singletons process the same set of jobs as the singletons of X_j. In particular, the number of each type of job processed by the singletons is the same.

Based on $(\mathcal{P}_1)$, and given that $FillMaxArea$ schedules as many N_1 jobs as possible in the pairs, the number of N_1 jobs present in a pair is not greater in $\mathcal{S}_1$ than in X_j. If the number of N_1 jobs processed on a pair is greater in X_j, then $\mathcal{S}_1$ must contain more (N_2, N_2), (N_3, N_2), or (N_3, N_3) combinations than X_j, and therefore rejects more N_1 jobs (since N_1 jobs cannot be processed on a triplet). We can *replace* any job in such a combination with a rejected N_1 job until the number of N_1 jobs in pairs is the same as in X_j. This results in either (N_2, N_1) or (N_3, N_1) combinations, both of which are valid. The resulting solution is denoted $\mathcal{S}_2$ and satisfies $(\mathcal{P}_1)$ and $(\mathcal{P}_2)$: it processes the same number of N_1 jobs on pairs as X_j.

X_j cannot use less N_2 jobs than $\mathcal{S}_2$ for the pairs, because $FillMaxArea$ prioritizes N_2 jobs over N_3 jobs. Let us assume that the number of N_2 jobs processed on a pair is greater in X_j than in $\mathcal{S}_2$. Then the missing N_2 jobs in $\mathcal{S}_2$ can either be rejected or scheduled in a (N_3, N_3, N_2) triplet. We can *swap* all N_2 jobs in a (N_3, N_3, N_2) combination with N_3 jobs from (N_3, N_2) or (N_3, N_3). Here, the possible set combinations we get are either (N_2, N_2) or (N_3, N_2) and (N_3, N_3, N_3), which are all valid. If X_j still uses more N_2 jobs in pairs, then there are rejected N_2 jobs in $\mathcal{S}_2$. We can *replace* one N_3 job from a (N_3, N_2) or (N_3, N_3) combination with each of these rejected N_2 jobs. This results in (N_2, N_2) or (N_3, N_2) valid combinations. The resulting solution is denoted $\mathcal{S}_3$ and satisfies $(\mathcal{P}_1)$, $(\mathcal{P}_2)$ and $(\mathcal{P}_3)$: it processes the same number of N_2 and N_3 jobs on pairs as X_j.

Finally, if X_j schedules more N_2 jobs on triplets than $\mathcal{S}_3$, this implies that there are rejected N_2 jobs in $\mathcal{S}_3$. We can *replace* one N_3 job from a (N_3, N_3, N_3) combination of $\mathcal{S}_3$ with each of these rejected N_2 jobs. This results in a valid (N_3, N_3, N_2) combination. This solution is denoted $\mathcal{S}_4$, and since it satisfies $(\mathcal{P}_1)$, $(\mathcal{P}_2)$, $(\mathcal{P}_3)$ in addition to having the same number of N_2 jobs in triplets, we have shown that $\mathcal{S}_4$ uses the same number of G, N_1, N_2, and N_3 jobs as X_j.

Finally, we use the fact that when choosing a job from a long job set, $FillMaxArea$ always chooses the largest available job. This implies that $A^{X_j} \geq A^{\mathcal{S}_4}$. Since all transformations either increase or do not change the accepted area, we know that $A^{\mathcal{S}_4} \geq A^{\mathcal{S}_0}$, which concludes the proof. □

From this lemma, we can deduce the following bound on the cost of $\mathcal{S}$:

Lemma 5. *For any T, let $\mathcal{S}$ be the solution obtained by $FillMaxArea(J, m, T)$. We can bound its cost by: $Z^{\mathcal{S}} \leq \frac{5}{4}Tm + \rho R^*(T)$.*

Proof. This follows directly from $Z^{\mathcal{S}} = mC^{\mathcal{S}} + \rho R^{\mathcal{S}}$ and Lemma 4. □

C. $\mathcal{BEKP}$ Approximation Algorithm

If we know an optimal solution OPT with respect to the objective function Z, then we can compute the solution $FillMaxArea(J, m, C^{OPT})$. From Lemma 5 we get a $\frac{5}{4}$-approximation. In this section we show how to get an approximation of C^{OPT} (with ϵ as the precision ratio) with controlled complexity. In the end, we obtain $\mathcal{BEKP}$, which is a $\frac{5}{4}(1+\epsilon)$ approximation algorithm for any positive number ϵ.

The idea behind $\mathcal{BEKP}$ is to first compute an upper bound U and a lower bound L for the optimal makespan C^{OPT}, and then build different schedules with the $FillMaxArea$ algorithm for each makespan value C_i such that

$$C_i \in \{L, (1+\epsilon)L, ..., (1+\epsilon)^k L\}. \tag{2}$$

The number of iterations k is the smallest value such that $(1+\epsilon)^k L \geq U$, and can be computed as $k = \lceil \log_{1+\epsilon}(\frac{U}{L}) \rceil$. We now present how to compute U and L so that the value of $\frac{U}{L}$ is bounded, thus providing a bound for k.

1) Computing Bounds on the Optimal Makespan: First, let us define $f(C)$ in Eq. (3), which represents the minimum possible cost for a schedule with makespan of C, since $R^*(C)$ is the minimum possible area of rejected jobs for any schedule with that makespan. We can then provide two lower bounds on $f(C)$. The first one is presented in Eq. (4). For the second one, given the total workload $W = \sum_i p_i$, we know that $Cm + R^*(C) \geq W$, which implies that $R^*(C) \geq W - Cm$. Together with Eq. (3), this yields the lower bound shown in Eq. (5). Let us also define a target value $H = \dfrac{4\rho W}{5}$.

$$f(C) = Cm + \rho R^*(C) \tag{3}$$
$$f(C) \geq Cm \tag{4}$$
$$f(C) \geq \rho W - (\rho - 1)Cm \tag{5}$$

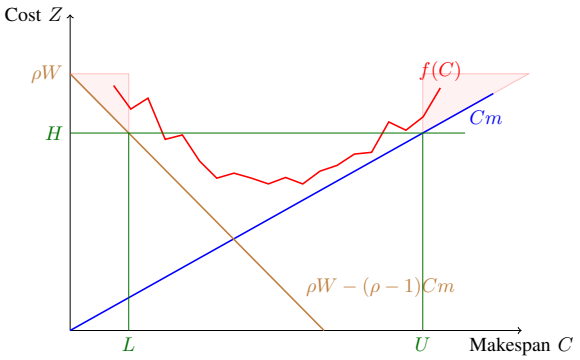

Fig. 1: Sketch of the plot of $f(C)$ (in red), highlighting how the bounds U and L are computed

Let us sketch the possible plot of the cost function $f(C)$ and the two bounds in Fig. 1. The function $f(C)$ is shown in red, the first bound from Eq. (4) is shown with a blue line, while the second bound from Eq. (5) is shown in brown. Finally, the target value H is displayed as a green horizontal line.

We define U and L as the values of C such that the first and second bounds are equal to H. This is shown in Fig. 1, and we get $U = \frac{4\rho W}{5m}$. Similarly, given the second bound and the target, we can compute their intersection as $L = \frac{\rho W}{5m(\rho-1)}$.

These values for U and L yield a ratio $\frac{U}{L} = 4(\rho-1)$, which provides a bound on the number of iterations k. For example, if we assume a rejection cost coefficient $\rho \leq 10$, then $4(\rho-1) \leq 36$. For example, if the precision is set to $1+\epsilon = 1.05$, then Eq. (2) specifies 74 different makespan values, which leads to a practical number of iterations. With these values, $\mathcal{BEKP}$ is a $\frac{5}{4} * 1.05 = 1.3125$ approximation algorithm. The tradeoff between number of iterations and performance guarantee can be adjusted when considering different values for ϵ.

2) $\mathcal{BEKP}$ Algorithm: $\mathcal{BEKP}$ is specified in Algorithm 3, where ϵ is a fixed parameter. The algorithm considers several possible schedules: the solution where all jobs are rejected, denoted by X_0, whose cost is $Z^{X_0} = \rho W$, and the result of $FillMaxArea(J, m, C_i)$ for each value C_i between L and U as in Eq. (2). The result of $\mathcal{BEKP}$ is the lowest cost schedule among all these candidates.

Algorithm 3 $\mathcal{BEKP}(J, m)$

1: $X_0 =$ the solution where all jobs are rejected
2: $U = \frac{4\rho W}{5m}$ and $L = \frac{\rho W}{5m(\rho-1)}$ and $k = \lceil \log_{1+\epsilon} \frac{U}{L} \rceil$
3: **for** each $C_i \in \{L, (1+\epsilon)L, (1+\epsilon)^2 L, ..., (1+\epsilon)^k L\}$ **do**
4: $\quad X_i = FillMaxArea(J, m, C_i)$
5: **return** schedule with the lowest cost among X_0 and all X_i

Theorem 1. *For any positive ϵ, $BEKP$ is a $\frac{5}{4}(1+\epsilon)$ approximation algorithm.*

Proof. Consider an arbitrary set of jobs J to be scheduled on m machines. Let OPT be a schedule for this instance with optimal cost. We compare Z^{OPT} with the cost of one of the X_i schedules considered in $\mathcal{BEKP}$. We analyze two cases, depending on the value of C^{OPT} relative to L and U:

If $C^{OPT} < L$ or $C^{OPT} > U$, we know from $Z^{OPT} \geq f(C^{OPT})$ and our lower bounds (Eqs. (4) and (5)) that $Z^{OPT} \geq H = \frac{4\rho W}{5}$. In Fig. 1 this can be interpreted as $f(C^{OPT})$ being located in one of the red triangle areas. Since $Z^{X_0} = \rho W$, we get $Z^{X_0} \leq \frac{5}{4} Z^{OPT}$.

If $L \leq C^{OPT} \leq U$, then there exists an index i such that $C^{OPT} \leq C_i \leq (1+\epsilon)C^{OPT}$. Let us denote this solution as X_i. By Lemma 5, we know that $Z^{X_i} \leq \frac{5}{4} m C_i + \rho R^*(C_i)$. Since $C^{OPT} \leq C_i$, Lemma 1 states that $R^*(C_i) \leq R^*(C^{OPT})$. Finally, since $C_i \leq (1+\epsilon)C^{OPT}$, we obtain

$$Z^{X_i} \leq \frac{5}{4}(1+\epsilon) m C^{OPT} + \rho R^*(C^{OPT}) \leq \frac{5}{4}(1+\epsilon) Z^{OPT}$$

In both cases, we identify a schedule X considered by $\mathcal{BEKP}$ that satisfies $Z^X \leq \frac{5}{4}(1+\epsilon) Z^{OPT}$. Since the result of $\mathcal{BEKP}$ has a cost not greater than X, this concludes the proof. □

3) Complexity of $\mathcal{BEKP}$: As discussed in Section III-C1, given the ratio $\frac{U}{L} = 4(\rho-1)$, the number of calls to $FillMaxArea$ in Algorithm 3 is $\mathcal{O}(\log_{1+\epsilon} \rho)$. In $FillMaxArea$ (Algorithm 2), the number of quadruplets to test is $\mathcal{O}(m^3)$, and for each of them we call $AssignFrom$ and greedily schedule the jobs in P. For a quadruplet, the complexity of all $AssignFrom$ calls is $\mathcal{O}(m)$ in total. We can assume that the jobs are sorted by increasing processing time at the beginning of $\mathcal{BEKP}$, which induces a one-time $\mathcal{O}(n \log n)$ complexity. Scheduling the jobs greedily can be performed in $\mathcal{O}(n)$.

In total, the complexity of $\mathcal{BEKP}$ is $\mathcal{O}(m^3(m+n)\log_{1+\epsilon} \rho)$. This can be compared to the algorithm proposed by Liu and Lu [8], whose complexity is $\mathcal{O}(n^3 \log n)$. We expect our approach to be significantly faster in scenarios with fewer machines and a larger number of jobs.

IV. Scheduling with Rejection and Deadlines

In this section, we consider the problem where jobs come with due dates. This eliminates the need to find a target value for the makespan as required in Section III-C, but the additional due date constraints make the problem more difficult. The problem formulation and necessary notations are given in Section IV-A. Sections IV-B and IV-C present how to build a solution for the single-machine and multiple-machines cases, respectively. In the multiple-machines case, our solution $\mathcal{MDP}$ also admits an approximation ratio and we establish its complexity. Finally, in Section IV-D, we prove that the approximation ratio of $\mathcal{MDP}$ is tight.

A. Problem Formulation

Using the notations in Table II, we consider a scheduling problem where a set of non-preemptive jobs J must be scheduled onto m identical machines. Each job i is characterized by its processing time p_i and its deadline d_i. The objective is to minimize the weighted number of late jobs in the schedule, where the weight of a late job is proportional to its processing times ($w_i = \rho p_i$, $\forall i \in J$). This problem is equivalent to maximizing the total weight of jobs that complete on time, which can be written in Graham's three-field notation as $P|d_j|\sum p_j(1-U_j)$.

In the context of Qarnot, the late jobs represent the *rejected* jobs on the edge boilers that must be offloaded to the unlimited supply of energy-inefficient machines on the cloud to satisfy QoS constraints. As a scheduling problem with rejection, we will denote the late jobs as **rejected** and the dues dates as **deadlines**, without loss of generality. Our goal is then to maximize the utilization of the edge boilers while meeting the deadlines of the jobs. A given solution $\mathcal{S}$ specifies (i) whether each job is *accepted* or *rejected* and (ii) assigns each accepted job to a machine while respecting its deadline. The profit associated to such a solution is given by $W^{\mathcal{S}} = \sum_{i \in accepted} p_i$.

B. Solution for a single machine ($\mathcal{DP}$)

In order to build a solution in the multiplemachines case, we start by providing an optimal solution for the single-machine case. This problem is similar to the job sequencing problem [24], and a dynamic programming based algorithm, denoted as $\mathcal{DP}$, has been proposed by Lawler and Moore [25] to solve it. Consider a set of jobs J, where jobs are sorted

TABLE II: Notation employed for the scheduling problem with deadlines.

m	Number of machines
n	Number of jobs
J	Set of jobs
p_i	Processing time (and profit) of job $i \in J$
d_i	Due date (deadline) of job $i \in J$
T	$\max\ d_i, i \in J$
$\mathcal{DP}$	Optimal solution on single machine
$\mathcal{MDP}$	Our solution on multiple machines
J_j	Set of remaining jobs to be scheduled in iteration j of $\mathcal{MDP}$
$W^\star_{Jj}$	Profit of an optimal solution of J_j on m machines
$W^\star$	Profit of an optimal solution for J on m machines
M^j	Profit of $\mathcal{MDP}$ for J on j machines

in non-decreasing order of their deadlines, and denote $T = \max_{i \in J} d_i$. We define $W(i,t)$ as the sum of the processing times of the on-time jobs among the jobs with indices $\{1,...,i\}$ that can complete before time t. Considering $t \leq T$, the recursive solution for a single machine (of time complexity $\mathcal{O}(nT)$) is as follows

$$\forall t \geq 0, W(0,t) = 0 \quad \text{and} \quad \forall i \geq 0, W(i,0) = 0$$

$$W(i,t) = \max \begin{cases} W(i-1, t-p_i) + p_i, & \text{if } p_i \leq t \leq d_i \\ W(i-1,t), & \text{otherwise} \end{cases} \tag{6}$$

Lemma 6. *Given a job set J, $\mathcal{DP}$ provides an optimal solution for the single machine problem.*

C. Solution on multiple machines ($\mathcal{MDP}$)

Our solution for the multiple machine problem, called $\mathcal{MDP}$ and described in Algorithm 4, is built by iteratively calling $\mathcal{DP}$ for each machine with the set of jobs that have not yet been assigned in the previous iterations of the algorithm. This same kind of logic has been employed by Berman and DasGupta [9] for a throughput maximization problem for k identical machines. Nevrtheless, they rely on a 2-approximation algorithm for the interval selection problem in a single machine, whereas our solution relies on an optimal algorithm at each iteration.

$\mathcal{MDP}$ has time complexity $\mathcal{O}(n \log n + mnT)$ due to the m calls to $\mathcal{DP}$ and to the initial sorting of jobs by non-decreasing deadlines.

Algorithm 4 $\mathcal{MDP}(J, m)$

1: $J_1 =$ sort(J, non-decreasing deadlines)
2: $MDP_0 = \emptyset$
3: **for** $j \in [1, ..., m]$ **do**
4: $\quad DP_j = \mathcal{DP}(J_j)$
5: $\quad J_{j+1} = J_j \setminus \{job | job \in DP_j\}$
6: $\quad MDP_j = MDP_{j-1} \cup DP_j$
7: **return** MDP_m

1) $\mathcal{MDP}$ Approximation Ratio: To prove the approximation ratio of $\mathcal{MDP}$ in Theorem 2, we use the same kind of argument used by Berman and DasGupta [9]. Consider a particular iteration j in $\mathcal{MDP}$, where the remaining jobs J_j are to be scheduled (see Line 4 in Algorithm 4), and let us denote by $W^\star_{Jj}$ the profit of an optimal solution with J_j on m machines. $W^\star$ represents the optimal profit for J on m machines, while M^j represents the profit of $\mathcal{MDP}$ for J on j machines.

Lemma 7. $M^j - M^{j-1} \geq \frac{1}{m} W^\star_{Jj}$.

Proof. $M^j - M^{j-1}$ is equal to the profit of $\mathcal{DP}$ when assigning J_j to a single machine. Consider an optimal solution with the same jobs on m machines: the highest profit assigned to a single machine in this solution is at least $\frac{1}{m} W^\star_{Jj}$. Since $\mathcal{DP}$ is optimal for a single machine (Lemma 6), the profit of the $\mathcal{DP}$ solution is larger than or equal to the profit of this single machine assignment. □

Lemma 8. $W^\star_{Jj} \geq W^\star - M^{j-1}$.

Proof. Consider $A \subseteq J$ the set of accepted jobs of an optimal solution for J on m machines, with profit $W^\star$. Consider also B, the set of jobs accepted by $\mathcal{MDP}$ on the first $j-1$ machines, by definition $B = J \setminus J_j$, with profit M^{j-1}. Removing B from A yields a solution whose profit is at least $W^\star - M^{j-1}$, with jobs in $A \cap J_j$. Since this solution uses only jobs in J_j, its profit is at most $W^\star_{Jj}$. □

Theorem 2. *$\mathcal{MDP}$ is an $(1 - \frac{(m-1)^m}{m^m})$-approximation algorithm.*

Proof. Using Lemma 7 and Lemma 8, we get the inequality in Eq. (7), which can be transformed to Eq. (9).

$$M^j - M^{j-1} \geq \frac{1}{m}(W^\star - M^{j-1}) \tag{7}$$

$$(W^\star - M^{j-1}) - (W^\star - M^j) \geq \frac{1}{m}(W^\star - M^{j-1}) \tag{8}$$

$$\frac{(m-1)}{m}(W^\star - M^{j-1}) \geq W^\star - M^j \tag{9}$$

We get Eq. (10) by recursively replacing M^{j-1} and, since $M^0 = 0$, we get the inequality in Eq. (11). The theorem is proved by replacing j by m, where M^m is the profit of the solution returned by $\mathcal{MDP}$.

$$\frac{(m-1)^j}{m^j}(W^\star - M^0) \geq (W^\star - M^j) \tag{10}$$

$$M^j \geq (1 - \frac{(m-1)^j}{m^j}) W^\star \tag{11}$$

□

D. Tightness of the Approximation Ratio

We now demonstrate the tightness of the $(1 - \frac{(m-1)^m}{m^m})$ approximation ratio. We first define a problem instance, then show that it accepts a valid schedule with profit m^m. We finally exhibit a valid $\mathcal{MDP}$ solution with profit $m^m - (m-1)^m$. These three steps are presented in the following sections.

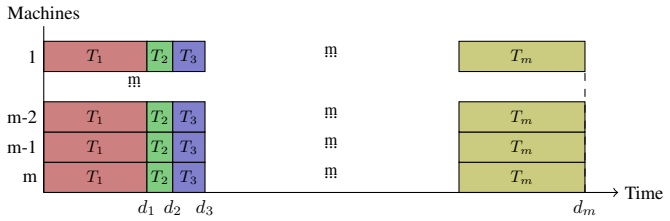

Fig. 2: An optimal schedule that accepts all jobs.

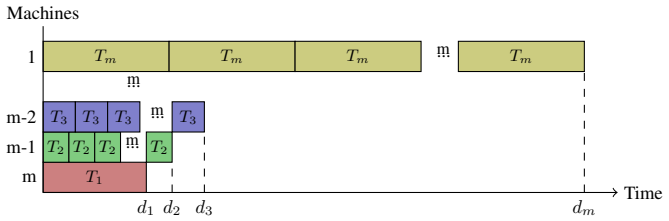

Fig. 3: A schedule that focuses on providing an optimal solution for each machine (in order) using the remaining jobs.

1) Problem Instance: For any m, we consider the problem instance containing the m^2 jobs enumerated in Table III, to be scheduled on m machines. Each row represents m jobs of a given type that share the same processing times and deadlines.

TABLE III: Jobs used in the tightness demonstration.

Count	Type	Processing Time	Deadline
m	T_1	$\mathbf{p_1 = (m-1)^{m-1}}$	$\mathbf{d_1 = p_1}$
m	T_2	$p_2 = (m-1)^{m-2}m^0$	$d_2 = mp_2$
m	T_3	$p_3 = (m-1)^{m-3}m^1$	$d_3 = mp_3$
...	...	...	...
m	T_i	$p_i = (m-1)^{m-i}m^{i-2}$	$d_i = mp_i$
...	...	...	...
m	T_m	$p_m = (m-1)^0 m^{m-2}$	$d_m = mp_m$

Remark 1. *The latest deadline within a job set is an upper bound on the optimal profit for the single machine problem.*

Remark 2. *By construction, all deadlines and processing times (except for jobs of Type T_1) satisfy $d_{i-1} = d_i - p_i$ and $p_i = \frac{d_i}{m}$, respectively.*

Remark 3. *The smaller the index i of type T_i, the earlier the deadline d_i.*

2) Optimal Schedule with Profit $\mathbf{m^m}$: Consider a schedule that assigns exactly one job of each type to each machine in the order (T_1, T_2, ..., T_i, ..., T_m). This means that each job finishes exactly on its deadline (Remark 2). This schedule is illustrated in Fig. 2.

a) Schedule Validity: We can establish the validity of the schedule by focusing on a single machine. The first job (of type T_1) starts at time 0 and finishes at time $d_1 = p_1$. The start time for the job of type T_i for $i > 1$ is $d_i - p_i$ (which is equal to d_{i-1}, as by Remark 2) and its finish time is exactly d_i.) In other words, each job of type T_i starts as soon as the job of type T_{i-1} finishes, and each job finishes on its deadline, with no idle time in between. Since there are no overlaps and all deadlines are met, the schedule is valid.

b) Profit of the Optimal Schedule: The last job assigned on each machine finishes at time d_m. By definition $d_m = m^{m-2} \cdot m = m^{m-1}$. Thus, the sum of all processing times on all machines is $\sum_{j=1}^{m} d_m = m \cdot m^{m-1} = m^m$.

3) Valid $\mathcal{MDP}$ Solution with Profit $\mathbf{m^m - (m-1)^m}$: As $\mathcal{MDP}$ greedily builds in order an optimal schedule for each machine with the remaining jobs, let us consider a schedule that assigns m jobs of type $T_{m-(j-1)}$ to machine $1 \leq j < m$. For the last machine ($j = m$), only one job of type T_1 is assigned. This schedule is illustrated in Fig. 3.

a) Schedule Validity: Let us first focus on a machine index $1 \leq j < m$. We claim that scheduling m jobs of type $T_{m-(j-1)}$ on machine j is a valid and optimal solution with the jobs that remain after the decisions made for the previous machines.

For the first machine assignment ($j = 1$), assigning all the jobs of type T_m (m jobs in total) is valid: the last job of T_m on this machine completes at time $p_m \cdot m = m^{m-2} \cdot m = m^{m-1} = d_m$. Since d_m is the largest deadline among all jobs, this is an optimal solution for this machine.

For machine $j < m$, assigning m jobs of $T_{m-(j-1)}$ is valid: $m \cdot p_{m-(j-1)} = d_{m-(j-1)}$. Moreover, since the largest deadline among remaining jobs is $d_{m-(j-1)}$ (i.e., all other jobs with later deadlines have already been scheduled in the previous machines), this schedule with profit $m \cdot p_{m-(j-1)}$ is an optimal solution for machine j.

Finally, for machine m, only m jobs of type T_1 remain. Since $p_1 = d_1$, only one of the jobs is scheduled on this machine, and this is obviously optimal. This shows that the schedule depicted on Figure 3 is a possible output of $\mathcal{MDP}$.

b) Profit of $\mathcal{MDP}$ Schedule: In total, all jobs are accepted, except for $m - 1$ jobs of type T_1. Since accepting all jobs, as seen in the optimal schedule, provides a profit of m^m, the resulting profit for this schedule is $m^m - (m-1) \cdot p_1 = m^m - (m-1) \cdot (m-1)^{m-1} = m^m - (m-1)^m$. Thus, the ratio between the $\mathcal{MDP}$ schedule and the optimal schedule for this problem is $1 - \frac{(m-1)^m}{m^m}$, proving that the approximation ratio of $\mathcal{MDP}$ presented in Section IV-C1 is tight.

V. Experiments

In this section, we evaluate the quality of the solutions obtained by our algorithms for scheduling with rejection without deadlines (Section III) and with deadlines (Section IV) in comparison to other state-of-the-art approaches.

All experiments were run sequentially on the Miriel nodes (each consisting of two Intel Xeon E5-2680v3 12-core 2.50 GHz processors with 128 GB of memory) of the PlaFRIM platform[2]. We used Python3 for programming and testing the various algorithms, and Gurobi for computing bounds with linear programming.

We generated random instances in which job processing times follow a lognormal distribution with mean 3. We use three different values for the standard deviation σ: 0.5, 0.7, and 1.0. As σ increases from smaller to larger values, the

[2]https://www.plafrim.fr

variance in processing times between jobs also increases. The deadlines in Section V-B are tied to the processing times: the deadline of a job with processing time p_i follows a uniform distribution with values between p_i and $3p_i$. For each case, we generate 30 different random sets of jobs. The results for each method over the 30 random instances are shown with a boxplot showing the median, first and third quartiles, with whiskers extending to the lowest and highest values, and small black dots representing outliers.

A. Experiments with $\mathcal{BEKP}$ (without deadlines)

We evaluate $\mathcal{BEKP}$ in terms of the total solution cost (Eq. (1)). To provide reference points, we consider two existing solutions: a naive solution $\mathcal{LPT}$ that accepts all jobs and schedules them using the Longest Processing Time-first method, and the algorithm proposed by Liu and Lu [8], denoted $\mathcal{LIULU}$. In addition, we also compute a lower bound on the solution cost with an Integer Linear Programming formulation that evaluates the makespan of a set of jobs with the standard lower bounds $\max_i p_i$ and $\frac{\sum_i p_i}{m}$, and optimally decides which jobs to accept. This can be formulated with a boolean variable x_i equal to 1 if the job is accepted and 0 otherwise, as follows

$$\text{minimize } Cm + \sum_{i \in J} \rho(1 - x_i)p_i$$
$$\text{subject to } \forall i \in J, C \geq x_i p_i$$
$$C \geq \sum_{i \in J} (x_i p_i)/m$$

For these experiments, we set the number of machines m to 20 and use two values of ρ: 1.5 and 4. Larger values of m give similar results. The results are shown in Fig. 4, where each grid column corresponds to a different rejection cost coefficient, while each row corresponds to a different number of jobs. The horizontal axis represents the different standard deviations used to generate the processing times, and the vertical axis represents the relative cost of each method compared to the lower bound (where a limit is set for better visualization).

We can observe in Figure 4 that the naive solution $\mathcal{LPT}$ tends to lead to solutions with higher costs than the other two methods, especially when there is a small number of jobs and a large variance in their processing times. This result illustrates the importance of sometimes rejecting jobs (i.e., running them on the cloud instead of on the edge boilers), since accepting all jobs can lead to high occupation costs ($m \cdot C^{\mathcal{S}}$). In our experiments, the high cost of some of the solutions obtained by $\mathcal{LPT}$ reached up to four times the value of the lower bound.

On the other hand, both $\mathcal{BEKP}$ and $\mathcal{LIULU}$ provide low-cost solutions thanks to their rejection capabilities, with neither exceeding the lower bound by more than a factor of 1.2. With the exception of the 40 jobs scenario, $\mathcal{BEKP}$ consistently achieved results with similar or better cost than $\mathcal{LIULU}$. These results show that $\mathcal{BEKP}$ not only provides a better approximation ratio, but also better schedules in practice.

B. Experiments with $\mathcal{MDP}$ (with deadlines)

We evaluate $\mathcal{MDP}$ in terms of the total solution profit (sum of the processing times of the on-time jobs). As a benchmark,

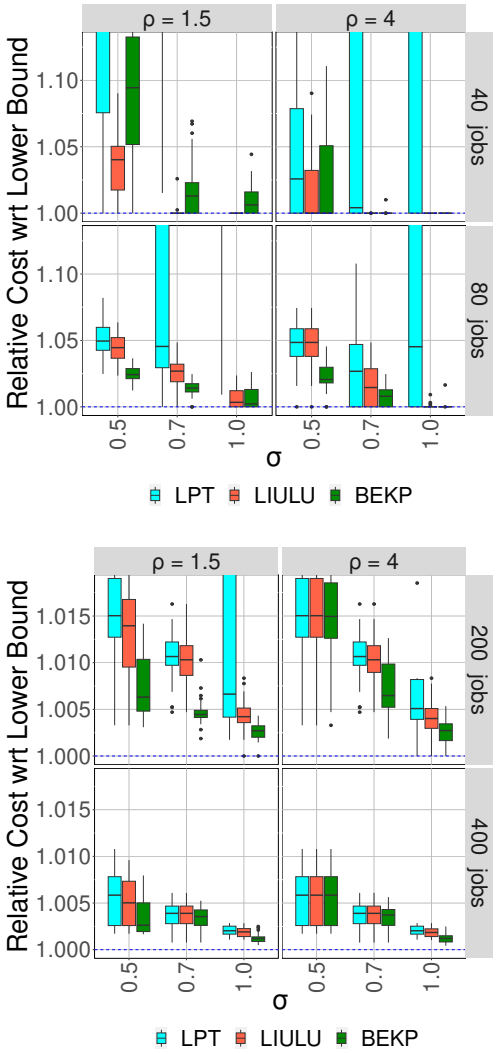

Fig. 4: Comparing $\mathcal{LPT}$, $\mathcal{LIULU}$, and $\mathcal{BEKP}$ using $m = 20$.

we compare it to the algorithm proposed by Berman and DasGupta [9], denoted $\mathcal{BERMAN}$. We compute an upper bound on the solution profit with a Linear Programming formulation that allows fractional and parallel assignment of jobs at different time intervals. For this purpose, we first define each interval j as the time interval between consecutive values of 0 and unique deadlines in the given job set (assuming that the deadlines are sorted in non-decreasing order). The length of the interval j is denoted by l_j. The decision variable $x_{i,j}$ is defined as the processing time of job i during interval j and is only valid if deadline d_i is after the end of interval j. The upper bound is given by the following formulation

$$\text{maximize } \sum_{i,j} x_{i,j}$$
$$\text{subject to } \forall j, \sum_{i} x_{i,j} \leq m \cdot l_j$$
$$\forall i, \sum_{j} x_{i,j} \leq p_i$$

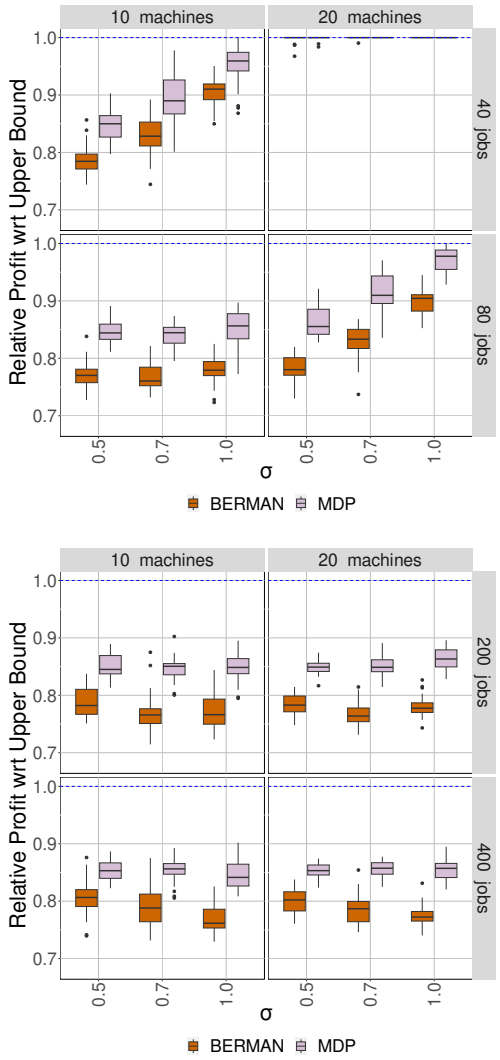

Fig. 5: Comparison of $\mathcal{BERMAN}$ and $\mathcal{MDP}$.

For these experiments, we vary the number of machines m between 10 and 20. Fig. 5 illustrates these results, where each grid column corresponds to a different number of machines and each row corresponds to a different number of jobs. The horizontal axis represents the different standard deviations used to generate the processing times, and the vertical axis represents the relative profit of each method compared to the upper bound (where a limit is set for better visualization).

We can note in Fig. 5 that $\mathcal{MDP}$ produces generally higher profits than $\mathcal{BERMAN}$ in all scenario variations. While $\mathcal{MDP}$ produces median profits relative to the upper bound of about 0.85 or above, $\mathcal{BERMAN}$ often produces median profits below 0.8. In other words, $\mathcal{MDP}$ is able to keep more jobs (more specifically, more processing time) on the edge boilers instead of pushing them to the cloud.

In further analysis, when we directly compare the profits of both algorithms for each problem instance, we find that $\mathcal{MDP}$ provides higher profits in the vast majority of cases (and has slightly lower profits than $\mathcal{BERMAN}$ in the other rare cases). Since both algorithms follow the same logic of scheduling jobs sequentially on each machine, these results illustrate the value of using an optimal algorithm ($\mathcal{DP}$) to make local scheduling decisions.

VI. Extensions to the Algorithms

During the development of this work, we have identified possibilities to improve the approximation ratio of the proposed algorithms at the expense of longer execution times, or to extend their applicability. In this section, we discuss some of these possibilities.

A. Improving the Approximation Ratio of $\mathcal{BEKP}$

Besides the ϵ parameter, the approximation ratio of $\mathcal{BEKP}$ depends on the thresholds used to group long jobs and the combination of long job assignments in $FillMaxArea$ (Section III-B1). In $FillMaxArea$ (Algorithm 2), we were able to limit the complexity of exploring all quadruplets of machines to $\mathcal{O}(m^3)$ by enforcing a strict order of singletons, pairs, and triplets. Lemma 4 shows that this strict ordering ensures that we get the best possible choice of combinations without testing them all. Without this ordering, we would have to iterate over all possible counts of the combinations listed in Lemma 2, resulting in a complexity of $\mathcal{O}(m^{11})$.

If we were to define more sets of jobs with different thresholds and their proper combinations, we could improve the approximation ratio of $\mathcal{BEKP}$. For example, if we define small jobs to be are those with $p_i \leq \frac{1}{5}T$, and list all valid combinations whose total processing times are at most $\frac{6}{5}T$, we would get a $\frac{6}{5}$ approximation for the resulting algorithm. However, it is not clear that an equivalent of Lemma 4 can be established, and this would lead to more set combinations and thus to higher complexity. We have decided to limit $FillMaxArea$ to its current expression in order to keep it practical both in terms of computational complexity and implementation feasibility, but the study of better approximations is left as an open problem.

B. Improving the Approximation Ratio of $\mathcal{MDP}$

It is possible to generalize $\mathcal{DP}$ so that it provides an optimal solution on κ machines (instead of a single machine) for a given set of jobs. Define $W(i, t_1, ..., t_\kappa)$ as the sum of the processing times of the on-time jobs with indices $\{1, ..., i\}$ that can finish before time t_j on machine $j \in \{1, ..., \kappa\}$. Then $\forall i, t_1, ..., t_\kappa > 0$, Eq. 12 returns the value of $W(i, t_1, ..., t_\kappa)$. This generalized $\mathcal{DP}$ solution has a complexity of $\mathcal{O}(nT^\kappa)$.

$$\max \begin{cases} W(i-1, t_1 - p_i, ..., t_\kappa) + p_i, & \text{if } p_i \leq t_1 \leq d_i \\ ... & \\ W(i-1, t_1, ..., t_\kappa - p_i) + p_i, & \text{if } p_i \leq t_\kappa \leq d_i \\ W(i-1, t_1, ..., t_\kappa), & \text{otherwise.} \end{cases} \tag{12}$$

By using this new version of $\mathcal{DP}$, $\mathcal{MDP}$ can iteratively schedule simultaneously the remaining jobs optimally on κ machines (over $\frac{m}{\kappa}$ iterations). This approach leads to a $(1 - (\frac{\frac{m}{\kappa}-1}{\frac{m}{\kappa}})^{\frac{m}{\kappa}})$-approximation of the optimal solution with a complexity of $\mathcal{O}(nlogn + mnT^\kappa)$.

C. Extending $\mathcal{MDP}$ to Arbitrary Profits

Our objective so far has been to minimize the weighted number of late jobs for scenarios in the context of Qarnot, where the weights (profits) are proportional to the processing times (section IV-A). However, both $\mathcal{DP}$ and $\mathcal{MDP}$ can still be used in scenarios where the weights are arbitrary. $\mathcal{DP}$ can find schedules with optimal profits for these scenarios simply by changing the way profits are computed in Eq. (6) to Eq. (13).

$$W(i,t) = \max \begin{cases} W(i-1, t-p_i) + w_i, & \text{if } p_i \leq t \leq d_i \\ W(i-1,t), & \text{otherwise} \end{cases} \tag{13}$$

This change does not affect the approximation ratio of $\mathcal{MDP}$ presented in Theorem 2, and its tightness, explored in Section IV-D, still holds.

VII. Concluding Remarks

In this paper, we consider two offline job scheduling problems where jobs are to be assigned to a limited set of energy-efficient machines at the edge, with the option of offloading them to less energy-efficient machines on the cloud when necessary. These problems can be viewed as scheduling problems with rejection, where rejection means using the less energy-efficient machines with an energy overhead proportional to the job processing time.

For the scenario where jobs have no deadlines, we introduce $\mathcal{BEKP}$, a $\frac{5}{4}(1+\epsilon)$-approximation algorithm with time complexity $\mathcal{O}(m^3(m+n)\log_{1+\epsilon}\rho)$, where the objective is to minimize the cost $Z^{\mathcal{S}}$, defined as the sum of the utilization of energy-efficient machines plus the rejection cost. For the scenario where jobs have deadlines, we propose $\mathcal{MDP}$, a $(1-\frac{(m-1)^m}{m^m})$-approximation algorithm with time complexity $\mathcal{O}(nlogn + mnT)$, where the objective is to maximize the profit of accepted jobs while meeting their deadlines.

In both scenarios, the proposed algorithms provide better approximation ratios than the state of the art with equivalent or better time complexity. Our experimental evaluation also shows that our algorithms produce good quality solutions in practice. $\mathcal{BEKP}$ yielded similar or lower costs than Liu and Lu's approach [8], while $\mathcal{MDP}$ yielded higher gains than Berman and DasGupta's algorithm [9].

We envision some challenges to be addressed as future work. With respect to the jobs, there are issues related to imprecise or probabilistic processing times that should be addressed. With respect to machines, there is the question of potential heterogeneity between cloud, fog, and edge resources, as well as between edge resources themselves, which would require new approaches. There is also the issue of data locality, where input data may be closer to some resources than to others. Finally, there is the challenge of making online job scheduling decisions, which remains to be explored.

References

[1] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica, and M. Zaharia, "A view of cloud computing," *Commun. ACM*, vol. 53, no. 4, p. 50–58, Apr. 2010. [Online]. Available: https://doi.org/10.1145/1721654.1721672

[2] F. Bonomi, R. Milito, J. Zhu, and S. Addepalli, "Fog computing and its role in the internet of things," in *Proceedings of the First Edition of the MCC Workshop on Mobile Cloud Computing*, ser. MCC '12. New York, NY, USA: Association for Computing Machinery, 2012, p. 13–16. [Online]. Available: https://doi.org/10.1145/2342509.2342513

[3] S. Dustdar, V. C. Pujol, and P. K. Donta, "On distributed computing continuum systems," *IEEE Trans. on Knowl. and Data Eng.*, vol. 35, no. 4, p. 4092–4105, Apr. 2023. [Online]. Available: https://doi.org/10.1109/TKDE.2022.3142856

[4] R. Bouzel, Y. Ngoko, P. Benoit, and N. Sainthérant, "Distributed grid computing manager covering waste heat reuse constraints," in *2021 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2021, pp. 294–299.

[5] D. Shabtay, N. Gaspar, and M. Kaspi, "A survey on offline scheduling with rejection," *Journal of scheduling*, vol. 16, pp. 3–28, 2013.

[6] B. Cesaret, C. Oğuz, and F. Sibel Salman, "A tabu search algorithm for order acceptance and scheduling," *Computers & Operations Research*, vol. 39, no. 6, pp. 1197–1205, 2012, special Issue on Scheduling in Manufacturing Systems.

[7] O. Beaumont, R. Bouzel, L. Eyraud-Dubois, E. Korkmaz, L. Pilla, and A. Van Kempen, "A $1.25(1+\epsilon)$-approximation algorithm for scheduling with rejection costs proportional to processing times," in *Euro-Par 2024: Parallel Processing*, J. Carretero, S. Shende, J. Garcia-Blas, I. Brandic, K. Olcoz, and M. Schreiber, Eds. Cham: Springer Nature Switzerland, 2024, pp. 225–238.

[8] P. Liu and X. Lu, "New approximation algorithms for machine scheduling with rejection on single and parallel machine," *Journal of Combinatorial Optimization*, vol. 40, no. 4, pp. 929–952, 2020.

[9] P. Berman and B. DasGupta, "Improvements in throughput maximization for real-time scheduling," in *Proceedings of the 32nd Annual ACM Symposium on Theory of Computing (STOC)*. Portland, OR, USA: ACM, May 2000, pp. 680–687.

[10] S. A. Slotnick, "Order acceptance and scheduling: A taxonomy and review," *European Journal of Operational Research*, vol. 212, no. 1, pp. 1–11, 2011.

[11] M. O. Adamu and A. O. Adewumi, "Minimizing the weighted number of tardy jobs on multiple machines: A review," *Journal of Industrial and Management Optimization*, vol. 12, no. 4, pp. 1465–1493, 2016. [Online]. Available: https://www.aimsciences.org/article/id/e886db39-7b9d-494c-ab28-4537fd2a1057

[12] Y. Bartal, S. Leonardi, A. Marchetti-Spaccamela, J. Sgall, and L. Stougie, "Multiprocessor scheduling with rejection," *SIAM Journal on Discrete Mathematics*, vol. 13, no. 1, pp. 64–78, 2000.

[13] J. Ou, X. Zhong, and G. Wang, "An improved heuristic for parallel machine scheduling with rejection," *European Journal of Operational Research*, vol. 241, no. 3, pp. 653–661, 2015.

[14] A. Caprara, H. Kellerer, and U. Pferschy, "A 3/4-approximation algorithm for multiple subset sum," *Journal of Heuristics*, vol. 9, no. 2, pp. 99–111, 03 2003.

[15] D. W. Engels, D. R. Karger, S. G. Kolliopoulos, S. Sengupta, R. N. Uma, and J. Wein, "Techniques for scheduling with rejection," in *Algorithms — ESA' 98*, G. Bilardi, G. F. Italiano, A. Pietracaprina, and G. Pucci, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 1998, pp. 490–501.

[16] L. Epstein, J. Noga, and G. J. Woeginger, "On-line scheduling of unit time jobs with rejection: minimizing the total completion time," *Operations Research Letters*, vol. 30, no. 6, pp. 415–420, 2002.

[17] Z. Liu, "Scheduling with partial rejection," *Operations Research Letters*, vol. 48, no. 4, pp. 524–529, 2020.

[18] B. Mor and D. Shabtay, "Single-machine scheduling with total late work and job rejection," *Computers & Industrial Engineering*, vol. 169, p. 108168, 2022.

[19] A. W. Kolen, J. K. Lenstra, C. H. Papadimitriou, and F. C. Spieksma, "Interval scheduling: A survey," *Naval Research Logistics (NRL)*, vol. 54, no. 5, pp. 530–543, 2007. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1002/nav.20231

[20] J. C. Ho and Y.-L. Chang, "Minimizing the number of tardy jobs for m parallel machines," *European Journal of Operational Research*, vol. 84, no. 2, pp. 343–355, 1995.

[21] D. Briskorn and S. Waldherr, "Anarchy in the uj: Coordination mechanisms for minimizing the number of late jobs," *European Journal of Operational Research*, vol. 301, no. 3, pp. 815–827, 2022. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S037722172100998X

[22] Z.-L. Chen and W. B. Powell, "Solving parallel machine scheduling problems by column generation," *INFORMS Journal on Computing*, vol. 11, no. 1, pp. 78–94, 1999.

[23] A. Bar-Noy, R. Bar-Yehuda, A. Freund, J. Naor, and B. Schieber, "A unified approach to approximating resource allocation and scheduling," *Journal of the ACM (JACM)*, vol. 48, no. 5, pp. 1069–1090, 2001.

[24] R. M. Karp, *Reducibility Among Combinatorial Problems*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 219–241. [Online]. Available: https://doi.org/10.1007/978-3-540-68279-0_8

[25] E. L. Lawler and J. M. Moore, "A functional equation and its application to resource allocation and sequencing problems," *Management science*, vol. 16, no. 1, pp. 77–84, 1969.