Two-Agent Scheduling with Resource Augmentation on Multiple Machines

Vincent Fagnon, Giorgio Lucarelli, Clément Mommessin, Denis Trystram

Two-Agent Scheduling with Resource Augmentation on Multiple Machines

Vincent Fagnon[1], Giorgio Lucarelli[2], Clément Mommessin[3] and Denis Trystram[1]

Abstract

We are interested in this paper in studying how to schedule tasks in an extreme edge setting, where some sensors produce data and subsequent tasks are executed locally. They are interacting with some external tasks submitted by a superior authority in the cloud. We first model such a system as a problem of scheduling two sets of tasks, each associated with its own objective. The tasks of the first set are released on-line, they can be preempted and the target objective is the minimization of the mean flow-time. The tasks of the second set are known beforehand and cannot be preempted. The objective is to execute them before a common deadline.

Due to strong lower bounds on the competitive ratio of this problem, we use the technique of resource augmentation to cope with these limitations. Specifically, our analysis is based on both speed augmentation and rejection. First, we give a general lower bound for the problem, even in the case of speed augmentation and rejection. Then, we propose a competitive algorithm and analyze its performance using dual fitting.

Keywords— Online Algorithm Competitive Analysis Edge Computing Preemptive Scheduling Resource Augmentation Dual Fitting

[1] Univ. Grenoble Alpes, CNRS, Inria, Grenoble INP, LIG, Grenoble, France
{vincent.fagnon,denis.trystram}@inria.fr
[2] LCOMS, University of Lorraine, Metz, France
giorgio.lucarelli@univ-lorraine.fr@univ-lorraine.fr
[3] University of Leeds, Leeds, UK
c.mommessin@leeds.ac.uk

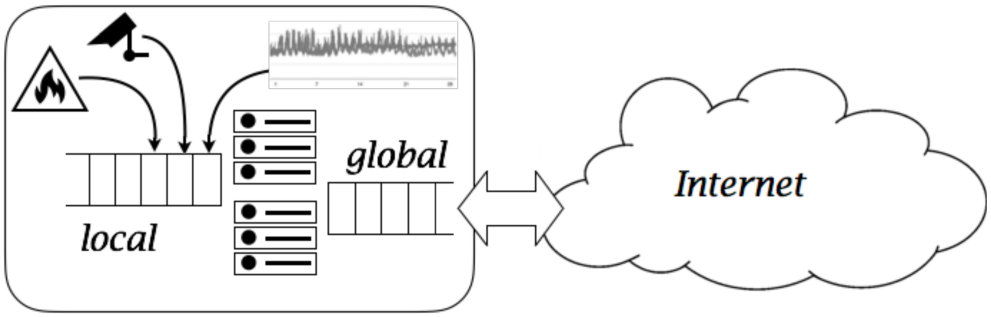

Figure 1: Schematic view of a smart building with multiple sensors (fire detector, motion sensor, video surveillance, sound analysis, etc.) and a link to the Internet (Cloud services, weather forecast data, information collected around in the smart city, etc.). The tasks associated to the sensors are collected in a local queue and the external tasks are put in a global queue. Both are managed by the computing units of the smart building.

1 Introduction

Today, the computing systems are evolving in a continuum of cloud/fog/edge. These systems are always more complex, since they are composed of various types of computing devices. The diversity of the digital components that compose such systems creates new problems in the perspective of managing efficiently the execution of tasks. Most data produced by sensors at the extreme edge should be processed immediately and the analysis of data should be done locally, close to the sensors [21]. Most of these data only have an interest in the local environment where they are produced, and their lifespan is short. Thus, they should be analyzed immediately. When no analysis is performed, the available computing units can be utilized for external or offloaded computing, such as volunteer computing [5].
In this work, we target a computing system composed of multiple computing units connected to some sensors. For instance, think of a smart building with a centralized control and processing units like the one described in Figure 1. A classical edge infrastructure is composed of several of such computing systems, but we will restrict our focus on a single one. Informally, there are two types of tasks to execute in this example, and each type is associated to a distinct objective as it will be described in more details in the next section.

1.1 Problem Description

The characteristics of the two types of tasks, each type associated with an agent, are the following:

- The first agent manages the tasks that are generated locally by the sensors. We call these tasks *local* or *dynamic* as they arrive on-line, and we should process them as soon as possible. The release time of a local task is not known in advance, but its processing time is known at the time it is released. The target objective of the first agent is to minimize the total flow-time of these tasks. Due to locality and their usually small processing time and memory consumption, the preemption (without migration) of these tasks is allowed.

- The second type of tasks corresponds to the external tasks submitted per batch. We call them *global* or *static* as they are submitted off-line. The processing of such tasks is known, and the objective is to complete this set of tasks in a reasonable amount of time, typically before a common deadline that is fixed, regarding the total computational load during the batch. From their nature and possibly large memory consumption compared to local tasks, a global task cannot be preempted. However, we allow the rejection of a global task during its execution, if this is needed to keep a good overall performance of the system. Intuitively, a rejected global task will be re-submitted in a subsequent batch in the same or in another targeted computing system.

The tasks of both agents are sequential. The overall problem is thus to interleave both sets of tasks on the set of computing machines in the *best* possible way with respect to the target objectives. To summarize, the inputs of the corresponding scheduling problem are as follows:

- m identical machines,
- $n^{\mathcal{L}}$ local tasks and $n^{\mathcal{G}}$ global tasks,
- p_j the processing time of task j.

The objective is to minimize the sum flow-time of the local tasks, i.e., the total time that a local task remains in the system, such that a global deadline is respected for the global tasks.

1.2 Contributions and Organization

In the single agent case, the on-line flow-time minimization problem can be solved to optimality on a single machine if preemptions are allowed, but it is hard to approximate with $m \geq 2$ machines [14], or in the off-line setting if preemptions are not allowed [13, 7]. These inapproximability results led to analyze the problem in the context of *resource augmentation*, where more power is given to the algorithm, as for instance allowing the algorithm to use more processors or higher speed than the optimal. In the case of speed augmentation, we assume the algorithm is using machines with a speed of $1 + \epsilon$, while the optimal solution is based on a machine speed of 1, where $\epsilon \geq 0$ is a constant. Using these resource augmentation models, we can achieve performance guarantees that depend on the value of ϵ [22, 12, 9].

With the introduction of tasks from the global agent, we show in this paper that it is hard to approximate the flow-time objective for the local tasks, even though their preemption is allowed and resource augmentation is used. Specifically, any on-line algorithm which uses $(1+\epsilon_s)$-speed machines should reject at most k global tasks to achieve a competitive ratio in $O\left((1-\frac{2k}{n^{\mathcal{G}}})\mathcal{W}\right)$, where $\mathcal{W}$ is the ratio between the total work load of the global tasks and the total work load of the local tasks.

On the positive side, we propose an algorithm to solve the addressed two-agent scheduling problem under the resource augmentation model, with both rejection and speed augmentation. Then, we analyze the competitive ratio of the algorithm using the dual fitting approach [4]. In particular we prove that our algorithm is $(1+\epsilon_s)$-speed and $\max\left\{\frac{\mathcal{W}}{\epsilon_s\epsilon_r} + \frac{1+\epsilon_s}{2\epsilon_s}, \frac{1+\epsilon_s}{\epsilon_s}\right\}$-competitive by rejecting a fraction of global tasks depending on a parameter ϵ_r, where $\epsilon_s > 0$ and $0 < \epsilon_r < 1$.

The organization of the paper is as follows: We start by presenting a brief state of the art about the two-agent scheduling problem, the flow-time minimization problem and resource augmentation models in Section 2. Section 3 gives a formal definition of the problem and the notations used throughout the paper, and Section 4 presents our lower bound. In Section 5, we introduce the competitive algorithm for solving the two-agent scheduling problem and analyze its competitive ratio with the dual fitting approach in Section 6. Section 7 ends the paper, with concluding remarks and discussions.

2 Related Work

The work presented in this paper can be seen as an extension of the two-agent scheduling problem by considering that the tasks of one agent arrive on-line.

The problem of two-agent scheduling has been introduced by Agnetis et al. [2]. Their seminal paper was dedicated to scheduling two sets of non-preemptive tasks competing to execute on a single machine, where each agent aims at minimizing its own objective function based on the completion time of its tasks (maximum, sum, due dates, etc.). The authors considered two ways for solving the problem: First, by minimizing an objective function while keeping the second objective under a threshold and, second, by finding the set of non-dominated pairs of objective values on the Pareto front. Later, Agnetis et al. [1] extended this work to other multi-agent scheduling problems including the execution on parallel machines.

One can find in the literature many variants of the two-agent scheduling problem, for example considering a chain of tasks for each agent [3], tasks with due dates [8, 24], or with an additional setup time when a task of one agent is processed directly after a task of the second agent [15].

In a similar setting, Baker and Smith [6] studied the problem of two or three agents on a single machine, where each agent has its own objective function to minimize. However, they considered the single objective approach by minimizing a linear combination of the objective of each agent. Liu et al. [16] also considered the problem with tasks arriving off-line with release times, with the objective of minimizing a linear combination of the maximum completion times (makespan) of both agents. Saule and Trystram [23] focused on the problem of an arbitrary number of agents scheduling jobs on parallel machines, where the objective of an agent is either the minimization of the makespan or the sum of completion time of its tasks. They proposed inapproximability bounds and approximation algorithms with performance ratios depending on the number of agents.

Our problem may also be considered as an extension of the problem of total flow-time minimization of on-line tasks on parallel machines, under the additional constraint of the off-line scheduling of tasks from a second agent.

For the flow-time minimization problem, it is well known that the *Shortest Remaining Processing Time* (SRPT) policy gives an optimal schedule for on-line tasks on a single machine. Unfortunately, the optimality of SRPT does

not hold in our context with a second agent, as will be shown in Section 4. When $m \geq 2$ machines are considered, the problem becomes NP-hard [10], and Leonardi and Raz [14] showed that SRPT is $O(\min\{\log(P), \log \frac{n}{m}\})$-competitive, where P is the ratio between the maximum and minimum processing time of the tasks, and n the number of tasks. They also showed that no on-line algorithm could achieve a better competitive ratio.

Such strong results on the inapproximability of scheduling problems motivated the introduction of the *resource augmentation* model, where an on-line algorithm is given more power when comparing to the optimal. Such models can be considered with *speed augmentation* [12], *machine augmentation* [22] or *rejection* [9].

In the context of scheduling on a single machine, Feng et al. [11] studied the common two-agent problem of Agnetis et al. with the additional feature that jobs can be rejected with a given penalty to the objective value of the corresponding agent. In the on-line setting, Lucarelli et al. also introduced the speed augmentation and rejection models to the total (weighted) flow-time minimization problem on unrelated [20, 17, 18] and related machines [19]. The authors proposed several algorithms whose competitiveness was proved using the dual fitting technique. We will use the same approach in the analysis of our algorithm in Section 5.

3 Problem Formulation and Notations

We consider a set $\mathcal{M}$ of m identical machines on which to execute two sets of sequential tasks having different settings. The first set $\mathcal{L}$ is composed of $n^{\mathcal{L}}$ *local tasks* that are submitted on-line, while the second set is composed of $n^{\mathcal{G}}$ *global tasks* that are submitted off-line. Only local tasks can be preempted, but migration between machines is not allowed. For a given task j, we denote by r_j its release time and by p_j its processing time on a machine, only known at the time the task is being released. Note that the release time is 0 for all global tasks, since they are off-line.

Given a schedule, we denote by F_j the flow-time of a local task j, defined as the difference between its completion time and its release time. The objective is to minimize the sum flow-time of all local tasks, under the constraint that all global tasks have completed before a common deadline $d^{\mathcal{G}}$. We also consider that the release time of any local task is bounded above by $d^{\mathcal{G}}$. This

deadline, defined as

$$d^{\mathcal{G}} = \frac{1}{m} \cdot \left(\sum_{j \in \mathcal{G}} p_j \ - \max_{j \in \mathcal{G}} \{p_j\} \right) + \max_{j \in \mathcal{G}} \{p_j\} + \sum_{j \in \mathcal{L}} p_j,$$

guarantees that a schedule interleaving local and global tasks is always feasible. Notice however that, since local tasks are released over time, the value of $d^{\mathcal{G}}$ is not known beforehand by the algorithm.

Under the resource augmentation model, we introduce the coefficients of speed augmentation $\epsilon_s > 0$ and rejection $0 < \epsilon_r < 1$. An algorithm to solve the addressed problem can use machines with speed $1 + \epsilon_s$ times faster than that of an optimal adversary, and can reject a number of global tasks bounded by an ϵ_r-fraction of the number of local tasks.

We define below some additional notations used in the next sections. In a partial schedule, $\mathcal{Q}_i^{\mathcal{L}}(t)$ denotes the set of local tasks assigned to machine i, but not completed at time t. This set includes all the local tasks waiting to be executed on machine i, as well as the task currently being executed at time t, if it is local. Note that, upon arrival of task j at time r_j and assigned to machine i, we assume that it is immediately added to $\mathcal{Q}_i^{\mathcal{L}}(r_j)$. Since preemption of local tasks is allowed, $p_j^{rem}(t)$ denotes the remaining processing time of task j at time t. Finally, for a given instance of the problem, we denote by $\mathcal{W} = \sum_{j \in \mathcal{G}} p_j / \sum_{j \in \mathcal{L}} p_j$ the ratio between the total workload of the global tasks and the total workload of the local tasks.

4 A Lower Bound

Theorem 1. *Let $\epsilon_s \leq \frac{1}{3} \cdot \frac{\mathcal{W}-3}{3\mathcal{W}+3}$ and $k \in \{1, 2, \ldots, \frac{n^{\mathcal{G}}}{2}\}$, where $n^{\mathcal{G}}$ is the number of global tasks. Any online algorithm which uses $(1+\epsilon_s)$-speed machines should reject at least k global tasks to have a competitive ratio in $O\left((1 - \frac{2k}{n^{\mathcal{G}}})\mathcal{W}\right)$.*

Proof. We consider an instance with a single machine, consisting of $2Z$ global tasks that are split evenly into two sets of tasks of respective processing times $X/3$ and $2X/3$. We partition the time into Z phases, each one of length $X+1$: the phase ℓ corresponds to the time interval $[(\ell-1)(X+1), \ell(X+1))$, for each $\ell = 1, 2, \ldots, Z$. Let $b_\ell = (\ell-1)(X+1)$, $1 \leq \ell \leq Z$, denote the beginning of phase ℓ. In order to well distinguish the phases, a burst of Y local tasks

of processing time 0 is released online at every time $\ell(X+1)$, $1 \leq \ell \leq Z$. Moreover, in each phase there is an arrival of a single local task of processing time 1, whose release time will be defined later. We have two cases to consider:

Case 1: If the online algorithm decides to execute one global task of processing time $X/3$ (say G_1) and one global task of processing time $2X/3$ (say G_2) in each phase ℓ, $1 \leq \ell \leq Z$.

If the algorithm decides to execute G_1 before G_2, then the earliest time at which the execution of G_1 can finish in the algorithm's schedule is $b_\ell + \frac{X}{3(1+\epsilon_s)}$, since the algorithm has access to a machine which executes the tasks at speed $1+\epsilon_s$. After the completion of G_1, the algorithm will start G_2 at some time $t \geq b_\ell + \frac{X}{3(1+\epsilon_s)}$. Then, the adversary decides to release the local task L at time $b_\ell + \frac{2X}{3}$ and the flow time of L will be at least $\frac{X-2X\epsilon_s+3}{3(1+\epsilon_s)}$, if G_2 is not rejected; otherwise the flow time of L will be $\frac{1}{1+\epsilon_s}$. The optimal solution executes the tasks in the order G_2, L and G_1, resulting in a flow-time of 1.

If the algorithm decides to execute G_2 before G_1, then let $t \geq b_\ell$ be the starting time of G_2. Then, the adversary decides to release the local task L at time $\frac{X}{3}$ and the flow time of L will be at least $\frac{X-X\epsilon_s+3}{3(1+\epsilon_s)}$, if G_2 is not rejected; otherwise the flow time of L will be $\frac{1}{1+\epsilon_s}$. The optimal solution executes the tasks in the order G_1, L and G_2, resulting in a flow-time of 1.

Assuming that the algorithm decides to reject the corresponding task G_2 in exactly k phases, then its total flow time is at least $\frac{k}{1+\epsilon_s} + (Z-k)\frac{X-2X\epsilon_s+3}{3(1+\epsilon_s)} = (Z-k)\frac{\mathcal{W}(1-2\epsilon_s)}{3(1+\epsilon_s)} + \frac{Z}{1+\epsilon_s}$, since $\mathcal{W} = X$. On the other hand, the total flow time of the optimal schedule is Z. Then, by rejecting k global tasks, the competitive ratio will be at least $\frac{Z-k}{Z}\frac{\mathcal{W}(1-2\epsilon_s)}{3(1+\epsilon_s)} + \frac{1}{1+\epsilon_s} = \Omega\left(\frac{Z-k}{Z}\mathcal{W}\right)$.

Case 2: Otherwise, there is a phase during which two global tasks of processing time $2X/3$ are partially executed. Hence, there exists a phase ℓ, at the beginning of which a global task started in the previous phase $\ell - 1$ will be executed for at least $q = \frac{1}{1+\epsilon_s}\frac{4X}{3} - (X+1)$ time. Then, the burst of local tasks arrived at time $(\ell-1)(X+1)$ will have a total flow-time at least qY. Note that in this case the arrival of local tasks of processing time 1 is not important and we can assume w.l.o.g. that the algorithm will execute all of them just upon their arrival, getting a total flow-time for them equal to Z. On the other hand, the optimal solution will be as in the previous case having a total flow time equal to Z. Therefore, the competitive ratio in this

case will be $\Omega(Y)$, which can be chosen large enough such that the first case dominates. □

5 An Algorithm for the Two-Agent Problem

The algorithm is denoted by $\mathcal{A}$ and described as follows.

Initialization: At start, we consider a queue of global tasks $\mathcal{Q}^{\mathcal{G}}$ initialized with all the global tasks sorted in an arbitrary order. For each machine $i \in \mathcal{M}$, we initialize an empty queue of local tasks $\mathcal{Q}_i^{\mathcal{L}}$.

Local task allocation: Upon submission of a local task j, we allocate it to the machine i that minimizes

$$\lambda_{ij} = \sum_{\substack{l \in \mathcal{Q}_i^{\mathcal{L}}(r_j): \\ p_l^{rem}(r_j) \leq p_j}} p_l^{rem}(r_j) + \sum_{\substack{l \in \mathcal{Q}_i^{\mathcal{L}}(r_j): \\ p_l^{rem}(r_j) > p_j}} p_j \tag{1}$$

and denote by λ_j this quantity. Intuitively, we put the task j on the machine minimizing the increase in the total flow-time induced only by the tasks in $\mathcal{Q}_i^{\mathcal{L}}$.

Task execution: For each machine i, the tasks in $\mathcal{Q}_i^{\mathcal{L}}$ are executed in the *Shortest Remaining Processing Time* order. Note that if a newly arrived task is shorter than the local task currently being executed, then preemption occurs and the remaining part of the current task is put back in $\mathcal{Q}_i^{\mathcal{L}}$.

If $\mathcal{Q}_i^{\mathcal{L}}$ becomes empty, then we remove a task from $\mathcal{Q}^{\mathcal{G}}$ and execute it on i, until no more global tasks are left. By not introducing idle times in the schedule before all global tasks are processed, we ensure that the last global task will complete before the common deadline $d^{\mathcal{G}}$. Note that global tasks are removed from the queue in an arbitrary order.

Rejection policy for global tasks: If at any time there are more than $\frac{1}{\epsilon_r}$ local tasks in the local queue of a machine currently executing a global task, we decide to reject that task. Note that such a rejection can only happen when a local task is released. This way, we ensure that no more than $\epsilon_r \cdot n^{\mathcal{L}}$ global tasks are rejected in total.

6 Analysis by Dual Fitting

We analyze our algorithm by dual fitting [4]. We first provide an expression of the total flow-time achieved by the algorithm in Section 6.1 and give a

linear programming formulation of our problem in Section 6.2, as well as the corresponding relaxed dual program. Then, in Section 6.3 we propose an assignment of the dual variables based on the choices made by our algorithm. In Section 6.4, we prove that this assignment satisfies the dual constraints and we give the competitive ratio of our algorithm.

6.1 Algorithm's Flow-Time

First, let Δ_j be the increase in the total flow-time of the current solution of our algorithm incurred by the dispatch of a local task j on machine i. This value corresponds to the flow-time of the task j, plus the increase in flow-time of all tasks being delayed by the arrival of j. Following our scheduling and rejection policies, and assuming the task k was being executed at time r_j, the detailed definition of Δ_j is expressed as follows:

$$\Delta_j = \begin{cases} \text{If k}(\in \mathcal{G}) \text{ is rejected:} \\ \quad - \dfrac{p_k^{rem}(r_j)}{1+\epsilon_s} \cdot |\mathcal{Q}_i^{\mathcal{L}}(r_j) - 1| + \displaystyle\sum_{\substack{l \in \mathcal{Q}_i^{\mathcal{L}}(r_j): \\ p_l^{rem}(r_j) \le p_j}} \frac{p_l^{rem}(r_j)}{1+\epsilon_s} + \sum_{\substack{l \in \mathcal{Q}_i^{\mathcal{L}}(r_j): \\ p_l^{rem}(r_j) > p_j}} \frac{p_j}{1+\epsilon_s} \\ \text{If k}(\in \mathcal{G}) \text{ is not rejected:} \\ \quad \dfrac{p_k^{rem}(r_j)}{1+\epsilon_s} + \displaystyle\sum_{\substack{l \in \mathcal{Q}_i^{\mathcal{L}}(r_j): \\ p_l^{rem}(r_j) \le p_j}} \frac{p_l^{rem}(r_j)}{1+\epsilon_s} + \sum_{\substack{l \in \mathcal{Q}_i^{\mathcal{L}}(r_j): \\ p_l^{rem}(r_j) > p_j}} \frac{p_j}{1+\epsilon_s} \\ \text{If k}(\in \mathcal{L}) \text{ is preempted:} \\ \quad \dfrac{p_j}{1+\epsilon_s} + \displaystyle\sum_{\substack{l \in \mathcal{Q}^{\mathcal{L}}(r_j): \\ p_l^{rem}(r_j) > p_j}} \frac{p_j}{1+\epsilon_s} \\ \text{If k}(\in \mathcal{L}) \text{ is not preempted:} \\ \quad \displaystyle\sum_{\substack{l \in \mathcal{Q}_i^{\mathcal{L}}(r_j): \\ p_l^{rem}(r_j) \le p_j}} \frac{p_l^{rem}(r_j)}{1+\epsilon_s} + \sum_{\substack{l \in \mathcal{Q}^{\mathcal{L}}(r_j): \\ p_l^{rem}(r_j) > p_j}} \frac{p_j}{1+\epsilon_s} \end{cases}$$

Note that, due to speed augmentation, any value of processing time for our algorithm is divided by $(1+\epsilon_s)$.

By definition, the total flow-time of local tasks achieved by the algorithm is equal to the sum of Δ_j for all local tasks. Due to the rejection policy, the impact of global tasks on the total flow-time of local tasks is limited. There cannot be more than $\frac{1}{\epsilon_r}$ local tasks in the queue during the execution of a global task, so a global task j will contribute at maximum to $\frac{p_j}{\epsilon_r(1+\epsilon_s)}$ on the total flow-time of the algorithm. Therefore, we have the following:

$$\sum_{j\in\mathcal{L}} F_j = \sum_{j\in\mathcal{L}} \Delta_j \leq \sum_{j\in\mathcal{L}} \left(\sum_{\substack{l\in\mathcal{Q}_i^{\mathcal{L}}(r_j):\\ p_l^{rem}(r_j)\leq p_j}} \frac{p_l^{rem}(r_j)}{1+\epsilon_s} + \sum_{\substack{l\in\mathcal{Q}_i^{\mathcal{L}}(r_j):\\ p_l^{rem}(r_j)> p_j}} \frac{p_j}{1+\epsilon_s} \right) + \sum_{j\in\mathcal{G}} \frac{p_j}{\epsilon_r(1+\epsilon_s)} \tag{2}$$

6.2 Linear Programming Formulation

We define a decision variable $x_{ij}(t)$ which is equal to 1 if the task $j \in \mathcal{L}\cup\mathcal{G}$ is running on machine $i \in \mathcal{M}$ at time $t \geq 0$, and 0 otherwise. By convention, we assume that $x_{ij}(t) = 0$ for local tasks when $t < r_j$.

Consider this linear programming formulation, with the constant $\Gamma \geq \frac{1}{2}$:

$$\text{min.} \quad \sum_{i\in\mathcal{M}} \sum_{j\in\mathcal{L}} \int_{r_j}^{\infty} \left(\frac{(t-r_j)}{p_j} + \Gamma \right) x_{ij}(t)dt$$

$$\text{s.t.} \sum_{i\in\mathcal{M}} \int_{r_j}^{\infty} x_{ij}(t)dt \geq p_j \qquad \forall j \in (\mathcal{L}\cup\mathcal{G}) \tag{3a}$$

$$\sum_{j\in(\mathcal{L}\cup\mathcal{G})} x_{ij}(t) \leq 1 \qquad \forall i \in \mathcal{M},\ \forall t \geq 0 \tag{3b}$$

$$\sum_{i\in\mathcal{M}} \int_{0}^{\infty} \left(\frac{t}{p_j} + \frac{1}{2} \right) x_{ij}(t)dt \leq d^{\mathcal{G}} \qquad \forall j \in \mathcal{G} \tag{3c}$$

$$x_{ij}(t) \in \{0;1\} \qquad \forall i \in \mathcal{M},\ \forall j \in (\mathcal{L}\cup\mathcal{G}),\ \forall t \geq 0$$

Constraint (3a) verifies that each task $j \in \mathcal{L}\cup\mathcal{G}$ is executed for at least p_j units of time, Constraint (3b) indicates that a machine can execute at most one task at any moment $t \geq 0$, and the left-hand side of Constraint (3c) computes the completion time of a global task j.

Note that the quantity $\int_{r_j}^{\infty} \frac{(t-r_j)}{p_j} x_{ij}(t)dt$ in the objective function corresponds to the well-known fractional flow-time of the job j [4]. This is a lower

bound to the flow-time of j. Moreover, with $\Gamma = \frac{1}{2}$, the objective value of an optimal solution for the linear program is at most the flow-time of an optimal schedule for our problem [4]. During our analysis, we will use a larger value of Γ and the objective value of the program will be at most $(\Gamma + \frac{1}{2})$ that of the optimal value of the problem.

Note also that the above formulation does not exactly model the addressed problem, as it does not forbid preemption of global task or their migration between machines. Instead, the formulation is a relaxation, but it is sufficient to analyse our algorithm and prove its approximation ratio via dual fitting and resource augmentation.

After relaxing the integrality constraint on the variables $x_{ij}(t)$ in the primal program, its dual program can be expressed as follows.

$$\max. \sum_{j \in \mathcal{L} \cup \mathcal{G}} \alpha_j p_j - \sum_{i \in \mathcal{M}} \int_0^\infty \beta_i(t) dt - \sum_{j \in \mathcal{G}} d^{\mathcal{G}} \gamma_j$$

$$\text{s.t. } \alpha_j - \beta_i(t) - \left(\frac{t - r_j}{p_j} + \Gamma \right) \leq 0 \qquad \forall i \in \mathcal{M},\ \forall j \in \mathcal{L},\ \forall t \geq r_j \tag{4a}$$

$$\alpha_j - \beta_i(t) - \left(\frac{t}{p_j} + \frac{1}{2} \right) \gamma_j \leq 0 \qquad \forall i \in \mathcal{M},\ \forall j \in \mathcal{G},\ \forall t \geq 0 \tag{4b}$$

$$\alpha_j \geq 0 \qquad \forall j \in (\mathcal{L} \cup \mathcal{G})$$

$$\beta_i(t) \geq 0 \qquad \forall t \geq 0,\ \forall i \in \mathcal{M}$$

$$\gamma_j \geq 0 \qquad \forall j \in \mathcal{G}$$

6.3 Dual Variables

For the purpose of the analysis via dual fitting, we define the variables of the dual program according to the decisions taken by the algorithm.

We define the first dual variable for local tasks as $\alpha_j = \frac{\lambda_j}{(1+\epsilon_s)p_j} + \Gamma$. Note that the value of α_j is only set at the arrival of j, and it does not change afterward. For the other variables, we define $\alpha_j = 0$ and $\gamma_j = 0$ for any global task j; and $\beta_i(t) = \frac{|Q_i^{\mathcal{L}}(t)|}{1+\epsilon_s},\ \forall t \geq 0$.

6.4 Competitive Analysis

With the two following lemmas, we show that our definition of the dual variables leads to a feasible solution of the dual program.

Lemma 1. *For every every machine $i \in \mathcal{M}$, every local task $j \in \mathcal{L}$ and every time $t \geq r_j$, the dual constraint (4a) is satisfied, that is:*

$$\alpha_j - \beta_i(t) - \left(\frac{t - r_j}{p_j} + \Gamma \right) \leq 0$$

Proof. First of all, the constant term Γ in the α_j is compensated with the Γ of the constraint. By multiplying all remaining terms by $(1 + \epsilon_s)p_j$, we have to prove:

$$\lambda_j - |\mathcal{Q}_i^{\mathcal{L}}(t)| \cdot p_j - (1 + \epsilon_s)(t - r_j) \leq 0$$

If more local tasks arrive after j, the quantity $|\mathcal{Q}_i^{\mathcal{L}}(t)|$ will increase and the constraint will be easier to satisfy. For this reason, we assume that there is no other task arrival after j. Thus $|\mathcal{Q}_i^{\mathcal{L}}(t)|$ can only decrease over time, when a task finishes its execution. Moreover, due to the dispatching policy of local tasks we have $\lambda_j \leq \lambda_{ij}$.

Assume that a task k is currently being executed at time r_j and finishes at time $t' = r_j + \frac{p_k^{rem}(r_j)}{1+\epsilon_s}$. For any time $t \in [r_j, t')$, $|\mathcal{Q}_i^{\mathcal{L}}(t)|$ will remain constant and it is sufficient to prove the constraint at the beginning of the interval, when $t = r_j$. We have

$$\lambda_{ij} \leq \sum_{\substack{l \in \mathcal{Q}_i^{\mathcal{L}}(r_j): \\ p_l^{rem}(r_j) \leq p_j}} p_j \quad + \sum_{\substack{l \in \mathcal{Q}_i^{\mathcal{L}}(r_j): \\ p_l^{rem}(r_j) > p_j}} p_j \; = |\mathcal{Q}_i^{\mathcal{L}}(r_j)| \cdot p_j$$

and thus the constraint is satisfied for any time t in this interval.

Let now k' be the first task in $\mathcal{Q}_i^{\mathcal{L}}(r_j)$ after k, such that when k finishes its execution at time t' then k' starts executing during the interval $\left[t', t' + \frac{p_{k'}^{rem}(r_j)}{1+\epsilon_s}\right)$. Again, it is sufficient to verify the constraint at the beginning of the interval, at time $t' = r_j + \frac{p_k^{rem}(r_j)}{1+\epsilon_s}$. We have

$$\begin{aligned} \lambda_{ij} &\leq p_k^{rem}(r_j) + \sum_{\substack{l \in \mathcal{Q}_i^{\mathcal{L}}(r_j) \setminus \{k\}: \\ p_l^{rem}(r_j) \leq p_j}} p_j \; + \sum_{\substack{l \in \mathcal{Q}_i^{\mathcal{L}}(r_j): \\ p_l^{rem}(r_j) > p_j}} p_j \\ &= p_k^{rem}(r_j) + |\mathcal{Q}_i^{\mathcal{L}}(t')| \cdot p_j \end{aligned}$$

and

$$p_k^{rem}(r_j) - (1 + \epsilon_s)(r_j + \frac{p_k^{rem}(r_j)}{1 + \epsilon_s} - r_j) \leq 0.$$

Thus, the constraint is satisfied for any t in the interval.

A similar reasoning for every time interval during the execution of any local task in $\mathcal{Q}_i^{\mathcal{L}}(r_j)$ can be made. Note that, if a global task was rejected on the arrival of the local task j, it would not change the above reasoning. Hence, the dual constraint (4a) is satisfied for any time $t \geq 0$. □

Lemma 2. *For every machine $i \in \mathcal{M}$, every global task j and every time $t \geq 0$, the dual constraint (4b) is satisfied, that is:*

$$\alpha_j - \beta_i(t) - (\frac{t}{p_j} + \frac{1}{2})\gamma_j \leq 0$$

Proof. The smallest possible value of $\beta_i(t)$ is when the queue of local tasks is empty. Thus, for any t we have $\beta_i(t) \geq 0$. Provided that $\alpha_j = 0$ and $\gamma_j = 0$ the dual constraint (4b) is satisfied. □

Recall that F_j denotes the flow-time of task j in the schedule produced by algorithm. Then, the following lemma shows the relation between the total flow-time of local tasks achieved by the algorithm and the objective value of the dual program.

Lemma 3. *Given our definitions of α_j, $\beta_i(t)$ and γ_j, the dual objective value verifies:*

$$\sum_{j \in \mathcal{L} \cup \mathcal{G}} \alpha_j p_j - \sum_{i \in \mathcal{M}} \int_0^\infty \beta_i(t) dt - \sum_{j \in \mathcal{G}} d^{\mathcal{G}} \gamma_j \geq \frac{\epsilon_s}{1+\epsilon_s} \sum_{j \in \mathcal{L}} F_j$$

Proof. With our definition of α_j (recall $\alpha_j = 0, \forall j \in \mathcal{G}$) we have

$$\begin{aligned}
\sum_{j \in \mathcal{L} \cup \mathcal{G}} \alpha_j p_j &= \sum_{j \in \mathcal{L}} \left(\sum_{\substack{l \in \mathcal{Q}_{i^*}^{\mathcal{L}}(r_j): \\ p_l^{rem}(r_j) \leq p_j}} \frac{p_l^{rem}(r_j)}{1+\epsilon_s} + \sum_{\substack{l \in \mathcal{Q}_{i^*}^{\mathcal{L}}(r_j): \\ p_l^{rem}(r_j) > p_j}} \frac{p_j}{1+\epsilon_s} + \Gamma \cdot p_j \right) \\
&\quad + \frac{1}{1+\epsilon_s} \left(\sum_{j \in \mathcal{G}} \frac{p_j}{\epsilon_r} - \sum_{j \in \mathcal{G}} \frac{p_j}{\epsilon_r} \right) \\
&\geq \sum_{j \in \mathcal{L}} F_j + \Gamma \cdot \sum_{j \in \mathcal{L}} p_j - \sum_{j \in \mathcal{G}} \frac{p_j}{\epsilon_r(1+\epsilon_s)},
\end{aligned}$$

where the inequality holds due to Equation (2).

With the definition of $\beta_i(t)$, we have:

$$\sum_{i\in\mathcal{M}}\int_0^\infty \beta_i(t)dt = \sum_{i\in\mathcal{M}}\int_0^\infty \frac{|\mathcal{Q}_i^{\mathcal{L}}(t)|}{1+\epsilon_s}dt = \frac{1}{1+\epsilon_s}\sum_{j\in\mathcal{L}} F_j$$

Recall the notation $\mathcal{W} = \frac{\sum_{j\in\mathcal{G}} p_j}{\sum_{j\in\mathcal{L}} pj}$, which is the ratio between the total work load of the global tasks and the total work load of the local tasks. By choosing

$$\Gamma = \max\left\{\frac{\mathcal{W}}{\epsilon_r(1+\epsilon_s)}, \frac{1}{2}\right\},$$

and knowing that $\gamma_j = 0$, $\forall j \in \mathcal{G}$, the objective value of the dual program can be expressed as follows:

$$\begin{aligned}
&\sum_{j\in\mathcal{L}\cup\mathcal{G}} \alpha_j p_j - \sum_{i\in\mathcal{M}}\int_0^\infty \beta_i(t)dt - \sum_{j\in\mathcal{G}} d^{\mathcal{G}}\gamma_j \\
&\quad\geq \sum_{j\in\mathcal{L}} F_j + \Gamma\cdot\sum_{j\in\mathcal{L}} p_j - \sum_{j\in\mathcal{G}}\frac{p_j}{\epsilon_r(1+\epsilon_s)} - \frac{1}{1+\epsilon_s}\sum_{j\in\mathcal{L}} F_j \\
&\quad= \frac{\epsilon_s}{1+\epsilon_s}\sum_{j\in\mathcal{L}} F_j + \Gamma\cdot\sum_{j\in\mathcal{L}} p_j - \sum_{j\in\mathcal{G}}\frac{p_j}{\epsilon_r(1+\epsilon_s)} \\
&\quad\geq \frac{\epsilon_s}{1+\epsilon_s}\sum_{j\in\mathcal{L}} F_j
\end{aligned}$$

□

We now have enough to conclude on the competitive ratio of our algorithm.

Theorem 2. *For any $0 < \epsilon_r < 1$ and $\epsilon_s > 0$, the algorithm $\mathcal{A}$ is $(1+\epsilon_s)$-speed, $\max\left\{\frac{\mathcal{W}}{\epsilon_s\epsilon_r} + \frac{1+\epsilon_s}{2\epsilon_s}, \frac{1+\epsilon_s}{\epsilon_s}\right\}$-competitive and rejects at most $(\epsilon_r\cdot n^{\mathcal{L}})$ global tasks.*

Proof. From the three previous lemmas we know that the dual variables as we defined them form a feasible solution of the dual program, and that the objective value of the dual program is an upper bound of the total flow-time achieved by the algorithm multiplied by a constant factor. From the rejection policy, a global task is rejected the first time when there are more than $\frac{1}{\epsilon_r}$ local tasks waiting in the queue. Thus, no more than $\epsilon_r\cdot n^{\mathcal{L}}$ global tasks

are rejected. By definition, the algorithm uses a machine with speed $(1+\epsilon_s)$ times the machine speed of an optimal solution.

It remains to express the relation between the total flow-time of our algorithm with that of an optimal solution. As $\Gamma \geq \frac{1}{2}$, the objective value of the primal program is at most $\Gamma + \frac{1}{2}$ times the flow-time of an optimal solution, denoted by OPT. Finally, using the duality theorem, we can write the relation:

$$\frac{\epsilon_s}{1+\epsilon_s} \cdot \sum_{j \in \mathcal{L}} F_j \leq (\Gamma + \frac{1}{2}) \cdot OPT$$

$$\frac{\sum_{j \in \mathcal{L}} F_j}{OPT} \leq \max \left\{ \frac{\mathcal{W}}{\epsilon_s \epsilon_r} + \frac{1+\epsilon_s}{2\epsilon_s}, \frac{1+\epsilon_s}{\epsilon_s} \right\}$$

and the theorem follows. □

7 Concluding Remarks

We introduced in this paper a new scheduling problem with two agents targeting each a different objective. As far as we know, it is the first time a mixed off-line/on-line setting was studied for this problem. We provided a lower bound on the competitive ratio of any algorithm for solving the problem, and proposed an algorithm with a proof of its competitive ratio via dual fitting under a resource augmentation framework (speed and rejection).

The objective of the global tasks is taken as a constraint, while its value is specified by the scheduling policy of global tasks. Using List Scheduling in our case, the common deadline was taken as the classical Graham's bound plus the total work load of the local tasks. It is possible to refine the value of the deadline in accordance with a change in the scheduling policy of global tasks in the algorithm $\mathcal{A}$: Using any algorithm $\mathcal{X}$ for $P||C_{max}$ with known approximation ratio ρ to construct a fixed allocation of global tasks to the machines prior to the submission of any local task, it is guaranteed that a value of the deadline $d^{\mathcal{G}} = C^{\mathcal{X}}_{max} + \sum_{j \in \mathcal{L}} p_j$ will be respected. Doing so, the dynamic allocation of global tasks is lost, but it gives a "bi-objective" vision to the problem: the minimization of both the total flow-time of local tasks and the global deadline.

Going further with the model, we believe it is possible to remove the speed augmentation, which is only used in the algorithm's analysis, as it was

successfully done in another work on a single agent problem with on-line tasks [17]. Another interesting perspective is to extend the model to the weighted case, or with related/unrelated machines.

Acknowledgment

This work was partly supported by the french ANR Energumen project 18-CE25-0008, by the research program on Edge Intelligence of the MIAI@Grenoble Alpes (ANR-19-P3IA-0003), and by the EPSRC funded project EP/T01461X/1 "Algorithmic Support for Massive Scale Distributed Systems".

References

[1] Alessandro Agnetis, Jean-Charles Billaut, Stanislaw Gawiejnowicz, Dario Pacciarelli, and Ameur Soukhal. *Multiagent Scheduling - Models and Algorithms*. Springer, 2014.

[2] Alessandro Agnetis, Pitu B. Mirchandani, Dario Pacciarelli, and Andrea Pacifici. Scheduling problems with two competing agents. *Operations Research*, 52(2):229–242, 2004.

[3] Alessandro Agnetis, Gaia Nicosia, Andrea Pacifici, and Ulrich Pferschy. Scheduling two agent task chains with a central selection mechanism. *J. Scheduling*, 18(3):243–261, 2015.

[4] S Anand, Naveen Garg, and Amit Kumar. Resource augmentation for weighted flow-time explained by dual fitting. In *Proceedings of the 23rd ACM-SIAM symposium on Discrete Algorithms*, pages 1228–1241. SIAM, 2012.

[5] David P Anderson. Boinc: A system for public-resource computing and storage. In *proceedings of the 5th IEEE/ACM International Workshop on Grid Computing*, pages 4–10. IEEE Computer Society, 2004.

[6] Kenneth R. Baker and J. Cole Smith. A Multiple-Criterion Model for Machine Scheduling. *J. Scheduling*, 6(1):7–16, 2003.

[7] Chandra Chekuri, Sanjeev Khanna, and An Zhu. Algorithms for minimizing weighted flow time. In *Proceedings of the 33rd ACM symposium on Theory of Computing*, pages 84–93, 2001.

[8] C. Cheng, S. Li, K. Ying, and Y. Liu. Scheduling Jobs of Two Competing Agents on a Single Machine. *IEEE Access*, 7, 2019.

[9] Anamitra Roy Choudhury, Syamantak Das, Naveen Garg, and Amit Kumar. Rejecting jobs to minimize load and maximum flow-time. *J. of Computer and System Sciences*, 91:42–68, 2018.

[10] Jianzhong Du, Joseph Y-T Leung, and Gilbert H Young. Minimizing mean flow time with release time constraint. *Theoretical Computer Science*, 75(3):347–355, 1990.

[11] Qi Feng, Baoqiang Fan, Shisheng Li, and Weiping Shang. Two-agent scheduling with rejection on a single machine. *Applied Mathematical Modelling*, 39(3):1183–1193, 2015.

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

[13] Hans Kellerer, Thomas Tautenhahn, and Gerhard Woeginger. Approximability and nonapproximability results for minimizing total flow time on a single machine. *SIAM Journal on Computing*, 28(4):1155–1166, 1999.

[14] Stefano Leonardi and Danny Raz. Approximating total flow time on parallel machines. *J. of Computer and System Sciences*, 73(6):875–891, 2007.

[15] Shi-Sheng Li, Ren-Xia Chen, and Qi Feng. Scheduling two job families on a single machine with two competitive agents. *J. Comb. Optim.*, 32(3):784–799, 2016.

[16] Peihai Liu, Manzhan Gu, and Ganggang Li. Two-agent scheduling on a single machine with release dates. *Computers and Operations Research*, 111:35–42, 2019.

[17] Giorgio Lucarelli, Benjamin Moseley, Nguyen Kim Thang, Abhinav Srivastav, and Denis Trystram. Online non-preemptive scheduling on unrelated machines with rejections. In *Proceedings of the 30th on Symposium on Parallelism in Algorithms and Architectures, SPAA*, pages 291–300. ACM, 2018.

[18] Giorgio Lucarelli, Benjamin Moseley, Nguyen Kim Thang, Abhinav Srivastav, and Denis Trystram. Online non-preemptive scheduling to minimize weighted flow-time on unrelated machines. In *26th Annual European Symposium on Algorithms, ESA*, volume 112 of *LIPIcs*, pages 59:1–59:12, 2018.

[19] Giorgio Lucarelli, Benjamin Moseley, Nguyen Kim Thang, Abhinav Srivastav, and Denis Trystram. Online non-preemptive scheduling to minimize maximum weighted flow-time on related machines. In *39th IARCS Annual Conference on FSTTCS*, volume 150 of *LIPIcs*, pages 24:1–24:12, 2019.

[20] Giorgio Lucarelli, Nguyen Kim Thang, Abhinav Srivastav, and Denis Trystram. Online non-preemptive scheduling in a resource augmentation model based on duality. In *24th Annual European Symposium on Algorithms, ESA*, volume 57 of *LIPIcs*, pages 63:1–63:17, 2016.

[21] Dinh C Nguyen, Ming Ding, Pubudu N Pathirana, Aruna Seneviratne, Jun Li, and H Vincent Poor. Federated learning for internet of things: A comprehensive survey. *IEEE Communications Surveys & Tutorials*, 2021.

[22] Cynthia A Phillips, Cliff Stein, Eric Torng, and Joel Wein. Optimal time-critical scheduling via resource augmentation. In *Proceedings of the 29th ACM symposium on Theory of Computing*, pages 140–149, 1997.

[23] Erik Saule and Denis Trystram. Multi-users scheduling in parallel systems. In *23rd IEEE International Symposium on Parallel and Distributed Processing, (IPDPS)*, pages 1–9, 2009.

[24] Yunqiang Yin, Dujuan Wang, and T.C.Edwin Cheng. *Due Date-Related Scheduling with Two Agents - Models and Algorithms*. Springer, 2020.