Bounding the Flow Time in Online Scheduling with Structured Processing Sets

Abstract

Replication in distributed key-value stores makes scheduling more challenging, as it introduces processing set restrictions, which limits the number of machines that can process a given task. We focus on the online minimization of the maximum response time in such systems, that is, we aim at bounding the latency of each task. When processing sets have no structure, Anand et al. (*Algorithmica*, 2017) derive a strong lower bound on the competitiveness of the problem: no online scheduling algorithm can have a competitive ratio smaller than $\Omega(m)$, where m is the number of machines. In practice, data replication schemes are regular, and structured processing sets may make the problem easier to solve. We derive new lower bounds for various common structures, including *inclusive*, *nested* or *interval* structures. In particular, we consider fixed sized intervals of machines, which mimic the standard replication strategy of key-value stores. We prove that EFT scheduling is $(3 - \frac{2}{k})$-competitive when optimizing max-flow on *disjoint* intervals of size k. However, we show that the competitive ratio of EFT is at least $m - k + 1$ when these intervals overlap, even when unit tasks are considered. We compare these two replication strategies in simulations and assess their efficiency when popularity biases are introduced, i.e., when some machines are accessed more frequently than others because they hold popular data.

1 Introduction

Since more than a decade, a variety of applications increasingly relies on key-value stores to record user data [1], monitoring information in scientific projects [2], activity logs, metadata, statistics, etc. Such systems deal with a heavy load and while they succeed to process most requests with reasonable performance, they are prone to high delays for a few tasks (also known as the tail latency problem [3, 4]), which motivates the design of efficient processing strategies.

The large amount of stored data most commonly requires the replication of the key-value tuples on distributed resources. This mechanism ensures high availability in the case of a large number of requests. For instance, Dynamo [5] replicates data on nodes organized as a ring in a clock-wise fashion. This approach inspired other implementations such as Cassandra [6], Riak KV and Project Voldemort [7]. However, this eligibility constraint of each task to specific machines prevents achieving optimal performance in current systems. Moreover, loads between machines tends to be heterogeneous [8, 9] due to varying popularities between the keys, which constitutes an additional challenge. Finally, requests vary in size and the moment they are performed cannot be predicted precisely, leading to a difficult problem.

In this paper, we focus on the scheduling problems that appear in key-value stores and other distributed systems using data replication. We consider requests for data in the key-value store as *tasks* to be processed on a server (or *machine* in the scheduling terminology). In key-value stores, the most common objective is to minimize the response time, which is the time between the submission of a request (the *release* of a task) and the moment a server answers this request (*completion time* of the task). In the scheduling literature, this is called the *flow time*. Given the dynamic nature of the problem, we focus on simple practical algorithms with competitive guarantees: we say that an online algorithm (without knowledge of future tasks) is ρ-competitive if it provides a solution that is always at most ρ times worst than an optimal offline solution. Using Graham's notation [10], we consider the problem $P|\text{online}-r_i|F_{\max}$: minimize the maximum flow time ($F_{\max}$) on identical machines (P), with tasks released over time (r_i) without prior knowledge of tasks before their released times (online). For this problem, FIFO is known to be a good solution: it is $(3 - 2/m)$-competitive on m parallel machines [11].

A major difficulty that we need to take into account is that data are not replicated everywhere in key-value stores: only a subset of servers holds the data for a specific request. In the scheduling literature, processing set restrictions are used to model the fact that only a subset $\mathcal{M}_i$ of machines may process some task T_i. This constraint makes the problem a lot more difficult: Anand et al. [12] prove a lower bound of $\Omega(m)$ on the competitive ratio of any online algorithm.

However, processing set restrictions often exhibit particular structures such as the clockwise ring used by Dynamo. In this case, data are replicated on direct neighbors forming an interval of consecutive machines, and it is unknown if this enables better results. In particular, we show that EFT, which is equivalent to FIFO for the problem $P|\text{online}-r_i|F_{\max}$ (Section 4), is a good strategy in some cases, but suffers from inefficient worst case performance with such realistic processing set restrictions (Section 6). Moreover, we establish the challenge of this problem, even with specific processing set restrictions, by proving lower bounds on the competitive ratio of any simple algorithm. Section 7 provides the last contribution by assessing the interaction of the popularity bias, or load imbalance, with the replication scheme in key-value stores. The rest of this paper starts by covering related works (Section 2) and presenting the model (Section 3).

2 Related Work

Max-flow minimization. Bender et al. were the first to propose the max-flow objective $F_{\max} = \max_i(C_i - r_i)$ [11, 13], in which C_i and r_i denote the completion and release times of the i-th task, respectively. They show that the well-known FIFO strategy is a $(3 - 2/m)$-competitive algorithm for minimizing max-flow on m parallel machines (note that this ratio is tight [14]), and they give a lower bound of $\frac{3}{2}$ on the online problem's competitiveness. The offline minimization of max-flow is strongly NP-hard since it is a generalization of the parallel makespan problem; Mastrolilli gives an FPTAS in unrelated setting that runs in time $O(nm(n^2/\varepsilon)^m)$ [15], where n is the number of tasks. When preemption is allowed, the problem becomes solvable on unrelated machines, as $F_{\max}$ is a special case of $L_{\max}$, in which a task's deadline is set to the value of its release time (i.e., $d_i = r_i$) [16–18]. FIFO has also been shown to be $(3 - 2/m)$-competitive for the preemptive problem [15]. Ambühl et al. refine the lower bound for both the preemptive and non-preemptive versions, proving that no online algorithm can achieve a ratio better than $2 - 1/m$ [19]. They provide an optimal algorithm for the preemptive case (i.e., matching the lower bound) and a lower bound of 2 for the non-preemptive problem when $m = 2$, implying that FIFO is also optimal on two parallel machines. In related setting, Bansal et al. derive lower bounds of $\Omega(m)$ and $\Omega(\log m)$ on the competitive ratio of Slow-Fit and Greedy [20]. They develop a new online algorithm, Double-Fit, that is 13.5-competitive by combining these two strategies. They also present a PTAS

Env.	Preemption	Algorithm	Type	Approximation	Ref.
P	Non-preemptive	FIFO	Online	$3 - \frac{2}{m}$	[11]
		any	Online	$\geq 2 - \frac{1}{m}$	[19]
	Preemptive	FIFO	Online	$3 - \frac{2}{m}$	[15]
				$2 - \frac{1}{m}$	[19]
		any	Online	$\geq 2 - \frac{1}{m}$	[19]
$P, \mathcal{M}_i$	Non-preemptive	*any*	Online	$\geq \Omega(m)$	[12]
Q	Non-preemptive	Double-Fit	Online	13.5	[20]
		Slow-Fit	Online	$\geq \Omega(m)$	[20]
		Greedy	Online	$\geq \Omega(\log m)$	[20]
R	Non-preemptive		Offline	$O(\log n)$	[22]
			Offline, PTAS	$1 + \varepsilon$ in $n^{O(m/\varepsilon)}$	[21]
			Offline, FPTAS	$1 + \varepsilon$ in $O(nm(n^2/\varepsilon)^m)$	[15]
	Preemptive		Offline	Optimal	[16–18]

Table 1: Existing results on max-flow optimization, with and without preemption. P, $P, \mathcal{M}_i$, Q and R denote parallel machines, parallel machines with processing set restrictions, related machines, and unrelated machines, respectively. We have $P \to Q \to R$ and $P \to P, \mathcal{M}_i \to R$, where $A \to B$ means that A is a special case of B. (F)PTAS stands for (Fully) Polynomial-Time Approximation Scheme. *any* indicates that the approximation ratio applies to any algorithm of the corresponding type. The approximation ratio of an online algorithm should be seen as a competitive ratio.

in unrelated environment, running in time $n^{O(m/\varepsilon)}$ [21], and an offline $O(\log n)$-approximation [22].

Processing set restrictions. Various surveys have been conducted on scheduling problems involving processing set restrictions. The majority of such problems concern makespan minimization in a wide range of situations, including preemption, structured sets, release times, and so on [23–26]. To the best of our knowledge, the only result on online max-flow minimization under (unstructured) processing set restrictions is due to Anand et al., who derive a lower bound of $\Omega(m)$ on the competitive ratio of any online algorithm [12].

Table 1 summarizes existing results on online max-flow minimization.

3 Model

Even though our problem originates from key-value stores, we formally formulate it using classical scheduling terms. In particular, we want to schedule n tasks $T_1, \ldots, T_n$ on m homogeneous machines $M_1, \ldots, M_m$ (or n requests on m servers/processors). Each task T_i has a release time $r_i \geq 0$ and a processing time $p_i > 0$. Any machine cannot process several tasks simultaneously and preemption is not allowed. Tasks arrive in the system over time and no information (release or processing time) on task T_i is available to the scheduler before time r_i, which is noted online$-r_i$. Without loss of generality, we assume tasks are numbered such that $i < j \implies r_i \leq r_j$.

Processing set restrictions (or eligibility constraints) prevent tasks to be processed on any machine. Formally, a task T_i can only be processed by a subset of machines $\mathcal{M}_i \subseteq M$ and we say that $\mathcal{M}_i$ is the processing set of T_i. Let us consider the following special structures for these processing sets:

$\boldsymbol{\mathcal{M}_i}$**(interval).** Interval processing sets are such that for all T_i, $\mathcal{M}_i = \{M_j \text{ s.t. } a_i \leq j \leq b_i\}$ or $\mathcal{M}_i = \{M_j \text{ s.t. } j \leq a_i \text{ or } b_i \leq j\}$, for some $a_i \leq b_i$.

$\boldsymbol{\mathcal{M}_i}$**(nested).** Nested processing sets are such that for all T_i, T_j (with $i \neq j$), either $\mathcal{M}_i \subseteq \mathcal{M}_j$, $\mathcal{M}_j \subseteq \mathcal{M}_i$ or $\mathcal{M}_i \cap \mathcal{M}_j = \emptyset$.

$\boldsymbol{\mathcal{M}_i}$**(inclusive).** Inclusive processing sets are such that for all T_i, T_j (with $i \neq j$), either $\mathcal{M}_i \subseteq \mathcal{M}_j$ or $\mathcal{M}_j \subseteq \mathcal{M}_i$.

$\boldsymbol{\mathcal{M}_i}$**(disjoint).** Disjoint processing sets are such that for all T_i, T_j (with $i \neq j$), either $\mathcal{M}_i = \mathcal{M}_j$ or $\mathcal{M}_i \cap \mathcal{M}_j = \emptyset$.

The *nested*, *inclusive* and *disjoint* processing set restrictions can be seen as special cases of the *interval* processing set restriction because it is always possible to reorder the machines in each subset $\mathcal{M}_i$ so that one obtains contiguous intervals of machines. Furthermore, the *inclusive* and *disjoint* processing set restrictions are special cases of the *nested* processing set restriction. Figure 1 summarizes the relations between the different structures in processing set restrictions.

In key-value stores, requests indicate which file to retrieve based on a key that can be used multiple times. This implies that multiple tasks may share the same processing time and processing set.

We can now define the desired output and objective function. For any scheduling algorithm $\mathcal{S}$, we note $\rho_i^{\mathcal{S}}$ the time at which T_i is scheduled by $\mathcal{S}$, $\mu_i^{\mathcal{S}}$ the index of the machine on which T_i is scheduled by $\mathcal{S}$, and $\sigma_i^{\mathcal{S}}$ the starting time of T_i under $\mathcal{S}$. In other words, $\mathcal{S}$ gives a schedule $\Pi^{\mathcal{S}}$ such that $\Pi^{\mathcal{S}}(i) = (\mu_i^{\mathcal{S}}, \sigma_i^{\mathcal{S}})$ for all task T_i. We want to minimize the maximum flow time $F_{\max}^{\mathcal{S}} = \max F_i^{\mathcal{S}}$, where $F_i^{\mathcal{S}} = C_i^{\mathcal{S}} - r_i$ ($C_i^{\mathcal{S}}$ denotes the completion time of T_i in $\Pi^{\mathcal{S}}$: $C_i^{\mathcal{S}} = \sigma_i^{\mathcal{S}} + p_i$). The superscript $\mathcal{S}$ is omitted when the considered algorithm is obvious from context.

We say that an online algorithm $\mathcal{D}$ has the *Immediate Dispatch* property if all tasks are scheduled as soon as they arrive in the system, i.e., for all T_i, we have $r_i \leq \rho_i^{\mathcal{D}} < r_i + \varepsilon$, where $0 < \varepsilon \ll 1$, and we call $\mathcal{D}$ an immediate dispatch algorithm. This property is of particular importance in systems that need to scale and cannot handle large waiting queues; the scheduling phase should be as fast as possible. It is often the case in online distributed systems such as load balancers or replicated key-value stores.

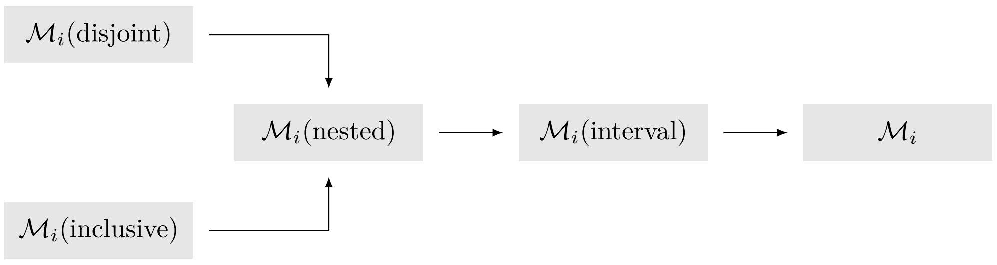

$\mathcal{M}_i$(disjoint)
$\mathcal{M}_i$(nested)
$\mathcal{M}_i$(interval)
$\mathcal{M}_i$
$\mathcal{M}_i$(inclusive)

Figure 1: Reduction graph of structures in processing set restrictions. $A \to B$ means that A is a special case of B.

4 Equivalence of FIFO and EFT strategies

FIFO scheduling has been extensively studied in previous work. It consists of a single queue of tasks, located on a central scheduler, that are pulled whenever some machine is available (see Algorithm 1). It is known to be $(3 - \frac{2}{m})$-competitive when minimizing maximum flow time on parallel machines [11, 13, 15], which makes it optimal on a single machine. In the present paper, we move our focus to the EFT scheduler (see Algorithm 2), which pushes each released task on the machine that finishes the earliest. We show here that both schedulers are equivalent on any instance of the scheduling problem $P|\text{online}{-}r_i|F_{\max}$. However, EFT has two main advantages over FIFO, which motivates our choice:

1. FIFO relies on a centralized queue, whereas EFT allocates tasks to machines as soon as they arrive (it is an immediate dispatch algorithm). Hence, it does not require a centralized scheduler with a potentially large queue of jobs, which is impractical in most existing online systems with critical scalability needs.

2. EFT can easily be extended to scenarios with processing set restrictions, whereas transforming FIFO to allow such constraints would be cumbersome.

For each machine $M_j \in M$ and for any $1 \leq i \leq n$, let $H_{j,i}^{\mathcal{S}}$ denote the subset of tasks $T_1, \ldots, T_i$ being assigned to M_j in a schedule $\Pi^{\mathcal{S}}$:

$$H_{j,i}^{\mathcal{S}} = \{T_{i'} \in T \text{ s.t. } 1 \leq i' \leq i \text{ and } \mu_{i'}^{\mathcal{S}} = j\}.$$

Then we define $\mathcal{C}_{j,i}^{\mathcal{S}}$ as the time at which M_j completes its assigned tasks among the first i tasks in $\Pi^{\mathcal{S}}$, i.e.,

$$\mathcal{C}_{j,i}^{\mathcal{S}} = \max_{T_{i'} \in H_{j,i}^{\mathcal{S}}} \left\{ C_{i'}^{\mathcal{S}} \right\},$$

where $C_{i'}^{\mathcal{S}} = \sigma_{i'}^{\mathcal{S}} + p_{i'}$ is the completion time of $T_{i'}$ in $\Pi^{\mathcal{S}}$, with the convention $\mathcal{C}_{j,0}^{\mathcal{S}} = 0$. Finally, we define $U_i^{\mathcal{S}}$ as the set of machines that may start the i-th task at the earliest possible time $t_{\min,i}^{\mathcal{S}} = \max\left(r_i, \min_{M_j \in M}\left\{\mathcal{C}_{j,i-1}^{\mathcal{S}}\right\}\right)$, i.e., $U_i^{\mathcal{S}}$ is the set of machines that are in a tie for T_i:

$$U_i^{\mathcal{S}} = \left\{ M_j \in M \text{ s.t. } \mathcal{C}_{j,i-1}^{\mathcal{S}} \leq t_{\min,i}^{\mathcal{S}} \right\}. \tag{1}$$

Note that EFT needs to know the set U_i for each released task T_i, which implies that one must know the processing time of arriving tasks with precision, in order to compute the completion times of machines at each step (we are in a clairvoyant setting). In this way, EFT can be readily modified to account for processing set restrictions by changing Equation (1) to

$$U_i' = \left\{ M_j \in \mathcal{M}_i \text{ s.t. } \mathcal{C}_{j,i-1} \leq t_{\min,i}' \right\}, \tag{2}$$

where $t_{\min,i}' = \max\left(r_i, \min_{M_j \in \mathcal{M}_i}\{\mathcal{C}_{j,i-1}\}\right)$.

For both EFT and FIFO strategies, a tie-break policy decides which machine will process T_i. We consider that ties are broken according to the same policy BreakTie in FIFO and EFT (in FIFO, ties are broken when at least 2 machines are idle at the same time; we assume the selected machine runs first).

Algorithm 1 FIFO

```
Require: Global FIFO queue $Q$
1: when a new task $T_i$ is released do
2:     $enqueue(i, Q)$

1: when some machines $U$ are idle at time $t$ do
2:     $i \leftarrow dequeue(Q)$
3:     if $i \neq NIL$ then
4:         $u \leftarrow$ BreakTie($U$)
5:         $\mu_i \leftarrow u$
6:         $\sigma_i \leftarrow t$
```

Algorithm 2 EFT

```
1: when a new task $T_i$ is released do
2:     Get $U_i$ according to completion times of
       machines $M$ (Equation (1))
3:     $u \leftarrow$ BreakTie($U_i$)
4:     $\mu_i \leftarrow u$
5:     $\sigma_i \leftarrow \max(r_i, \mathcal{C}_{u,i-1})$
6:     Update the completion time of $M_u$
```

Proposition 1. *For any instance $\mathcal{I}$ of the problem $P|\text{online}-r_i|F_{\max}$, we have* $\text{FIFO}(\mathcal{I}) = \text{EFT}(\mathcal{I})$*, i.e.,* $\Pi^{\text{FIFO}}(i) = \Pi^{\text{EFT}}(i)$ *for all $T_i \in T$ in the instance $\mathcal{I}$.*

Proof: Let $\mathcal{I}$ denote an arbitrary instance of the problem $P|\text{online}-r_i|F_{\max}$. We prove the following statement by induction: for any k such that $1 \leq k \leq n$, $\Pi^{\text{FIFO}}(i) = \Pi^{\text{EFT}}(i)$ for all $1 \leq i \leq k$, where $T_i \in T$ in the instance $\mathcal{I}$.

Base case ($k = 1$). All machines are idle (thus $U_1^{\text{FIFO}} = U_1^{\text{EFT}} = M$). As FIFO and EFT have the same tie-break policy and it is called on the same machine subset, they will choose the same machine and execute T_1 as soon as it is released.

Induction step. Suppose that for a given $k < n$, $\Pi^{\text{FIFO}}(i) = \Pi^{\text{EFT}}(i)$ for all $1 \leq i \leq k$. We show that $\Pi^{\text{FIFO}}(k+1) = \Pi^{\text{EFT}}(k+1)$.

On the one hand, at time r_{k+1}, EFT will schedule the task T_{k+1} on one machine M_u in the subset U_{k+1}^{EFT} according to the tie-break policy. Thus, we have $\mu_{k+1}^{\text{EFT}} = u$ and $\sigma_{k+1}^{\text{EFT}} = \max(r_{k+1}, \mathcal{C}_{u,k}^{\text{EFT}})$.

On the other hand, at time $\max(r_{k+1}, \min_j \mathcal{C}_{j,k}^{\text{FIFO}})$, one of the machine in the subset U_{k+1}^{FIFO} will wake up first according to the tie-break policy. Let $M_{u'}$ denote this machine. $M_{u'}$ will pull the next task to process from the shared queue Q, which is necessarily T_{k+1}. Therefore, $\mu_{k+1}^{\text{FIFO}} = u'$ and $\sigma_{k+1}^{\text{FIFO}} = \max(r_{k+1}, \mathcal{C}_{u',k}^{\text{FIFO}})$.

As $\Pi^{\text{FIFO}}(i) = \Pi^{\text{EFT}}(i)$ for all $1 \leq i \leq k$, we deduce that all machines complete at the same time in Π^{FIFO} and Π^{EFT} when the first k tasks are considered, i.e., for all j, $\mathcal{C}_{j,k}^{\text{FIFO}} = \mathcal{C}_{j,k}^{\text{EFT}}$.

This implies that $U_{k+1}^{\text{FIFO}} = U_{k+1}^{\text{EFT}}$. Thus, as FIFO and EFT break ties the same way, we have $u = u'$, and then $\mathcal{C}_{u,k}^{\text{FIFO}} = \mathcal{C}_{u',k}^{\text{EFT}}$. Therefore, $\mu_{k+1}^{\text{EFT}} = \mu_{k+1}^{\text{FIFO}}$ and $\sigma_{k+1}^{\text{EFT}} = \sigma_{k+1}^{\text{FIFO}}$. ■

The equivalence between EFT and FIFO implies that all existing results for FIFO also apply to EFT in the context of max-flow minimization on parallel machines without processing set restrictions.

5 Online Minimization of Max-Flow

In this section, we recall and give some results about the minimization of maximum flow time when there are no processing set restrictions, that is, when tasks can be scheduled on any machine. The offline problem $P|r_i|F_{\max}$ is clearly strongly NP-hard, as it is a generalization of $P||C_{\max}$.

For the online version $P|\text{online}-r_i|F_{\max}$, FIFO has been proven to be a $(3-2/m)$-competitive online algorithm. We describe the proof here, as the existing proof of Bender et al. is not entirely correct.

Theorem 1 (Bender et al. [11,13]). FIFO *is* $(3-\frac{2}{m})$*-competitive for the problem* $P|\text{online}-r_i|F_{\max}$.

Let $\mathcal{T}_t^{\mathcal{S}}$ denote the set of tasks released before time t and not yet started in schedule $\Pi^{\mathcal{S}}$, i.e.,

$$\mathcal{T}_t^{\mathcal{S}} = \{T_i \in T \text{ such that } r_i \leq t \text{ and } \sigma_i^{\mathcal{S}} > t\},$$

and let $\delta_{t,j}^{\mathcal{S}}$ be the remaining processing time of the task T_i being executed by machine M_j at time t in $\Pi^{\mathcal{S}}$, i.e., $\delta_{t,j}^{\mathcal{S}} = C_i^{\mathcal{S}} - t$, where $\mu_i^{\mathcal{S}} = j$ and $\sigma_i^{\mathcal{S}} \leq t \leq C_i^{\mathcal{S}}$. Obviously, if no task is being processed on M_j at time t, $\delta_{t,j}^{\mathcal{S}}$ is set to 0. Then, the total work waiting to be processed at time t in $\Pi^{\mathcal{S}}$ is defined as

$$W_t^{\mathcal{S}} = \sum_j \delta_{t,j}^{\mathcal{S}} + \sum_{T_i \in \mathcal{T}_t^{\mathcal{S}}} p_i.$$

We also define the maximum processing time among $T_1, \dots, T_i$ as $p_{\max,i}$. The maximum flow time among $T_1, \dots, T_i$ in schedule $\Pi^{\mathcal{S}}$ is noted as $F_{\max,i}^{\mathcal{S}}$.

Lemma 1. *For any task* T_i, $W_{r_i}^{\text{FIFO}} \leq W_{r_i}^{OPT} + (m-1)\, p_{\max,i}$*, where* OPT *is an optimal offline strategy.*

Proof: Let us proceed by induction.

Base case ($i = 1$). At time r_1, all machines are idle and we have $W_{r_1}^{\text{FIFO}} = W_{r_1}^{OPT} = p_1$.

Induction step. Suppose that $W_{r_i}^{\text{FIFO}} \leq W_{r_i}^{OPT} + (m-1)\, p_{\max,i}$ for a given i. We consider two cases:

(i) All machines are busy between r_i and r_{i+1} in Π^{FIFO}. We have

$$W_{r_{i+1}}^{\text{FIFO}} = W_{r_i}^{\text{FIFO}} - m(r_{i+1} - r_i) + p_{i+1}.$$

Moreover, $W_{r_{i+1}}^{OPT} \geq W_{r_i}^{OPT} - m\,(r_{i+1} - r_i) + p_{i+1}$ (there may be idle times between r_i and r_{i+1} in OPT). Then, $W_{r_{i+1}}^{OPT} - W_{r_i}^{OPT} \geq W_{r_{i+1}}^{\text{FIFO}} - W_{r_i}^{\text{FIFO}}$.

Hence,

$$\begin{aligned} W_{r_{i+1}}^{\text{FIFO}} &\leq W_{r_{i+1}}^{OPT} + W_{r_i}^{\text{FIFO}} - W_{r_i}^{OPT} \\ &\leq W_{r_{i+1}}^{OPT} + (m-1)\, p_{\max,i} \\ &\leq W_{r_{i+1}}^{OPT} + (m-1)\, p_{\max,i+1}. \end{aligned}$$

(ii) There is at least one idle machine between r_i and r_{i+1} in Π^{FIFO}. At time r_{i+1}, there is thus no waiting tasks except T_{i+1} (otherwise, it would have already started on an idle machine). In the worst case, $m-1$ machines start to process some tasks just before time r_{i+1} for $p_{\max,i}$ time units. Then we have $W^{\text{FIFO}}_{r_{i+1}} \leq p_{i+1} + (m-1)\,p_{\max,i}$.

Furthermore, in the best case T_{i+1} is the only task in the system at time r_{i+1} in Π^{OPT}, thus $W^{OPT}_{r_{i+1}} \geq p_{i+1}$.

Therefore,

$$\begin{aligned} W^{\text{FIFO}}_{r_{i+1}} &\leq p_{i+1} + (m-1)\,p_{\max,i} \\ &\leq W^{OPT}_{r_{i+1}} + (m-1)\,p_{\max,i} \\ &\leq W^{OPT}_{r_{i+1}} + (m-1)\,p_{\max,i+1}. \end{aligned} \qquad \blacksquare$$

Proof of Theorem 1: We consider an online schedule Π^{FIFO} built by FIFO and an optimal offline schedule Π^{OPT}. We start by describing two lower bounds for $F^{OPT}_{\max,i}$:

$$F^{OPT}_{\max,i} \geq p_{\max,i}, \tag{3}$$

$$F^{OPT}_{\max,i} \geq W^{OPT}_{r_i}/m. \tag{4}$$

Lower bound (3) is immediate. Lower bound (4) follows from the fact that there is always a non-finished task $T_{i'}$ such that $r_{i'} \leq r_i$ and that will necessarily complete after time $r_i + W^{OPT}_{r_i}/m$.

Now let T_i be a task in Π^{FIFO}. Then—as it is scheduled by FIFO—it is the last task in $\mathcal{T}^{\text{FIFO}}_{r_i}$, and it will not be able to start before time $r_i + (W^{\text{FIFO}}_{r_i} - p_i)/m$ in the worst case. Hence,

$$F^{\text{FIFO}}_i \leq W^{\text{FIFO}}_{r_i}/m + p_i - \frac{p_i}{m} \leq W^{\text{FIFO}}_{r_i}/m + \left(1 - \frac{1}{m}\right) p_{\max,i}$$

is an upper bound for FIFO. By Lemma 1, we know that $W^{\text{FIFO}}_{r_i} \leq W^{OPT}_{r_i} + (m-1)\,p_{\max,i}$ for each task T_i. Then,

$$\begin{aligned} F^{\text{FIFO}}_i &\leq W^{\text{FIFO}}_{r_i}/m + \left(1 - \frac{1}{m}\right) p_{\max,i} \\ &\leq W^{OPT}_{r_i}/m + 2\left(1 - \frac{1}{m}\right) p_{\max,i} \\ &\leq \left(3 - \frac{2}{m}\right) F^{OPT}_{\max,i} \text{ (by lower bounds (3) and (4)).} \end{aligned} \qquad \blacksquare$$

As a corollary, the problem is polynomial on a single-machine; FIFO is optimal in this case. It is also known that $P|r_i|F_{\max}$ is polynomial on parallel machines for homogeneous tasks [27]. As it solves a more general problem, the proposed algorithm is quite complex; we show that FIFO is sufficient to solve $P|r_i, p_i = p|F_{\max}$.

Theorem 2. FIFO *solves the problem* $P|\text{online}{-}r_i, p_i = p|F_{\max}$ *to optimality.*

Proof: Let OPT be an optimal offline strategy and Π^{OPT} an optimal schedule. If on each machines, tasks are processed by non-decreasing release time, then OPT corresponds to an execution of the FIFO algorithm: two tasks starting simultaneously on two machines may be

Processing set structure	Online	Immediate Dispatch	EFT
inclusive		$\lfloor \log_2(m) + 1 \rfloor$ (Th. 3)	
$\lvert\mathcal{M}_i\rvert = k$		$\lfloor \log_k(m) \rfloor$ (Th. 4)	
nested	$\frac{1}{3} \lfloor \log_2(m) + 2 \rfloor$ (Th. 5)		
interval, $\lvert\mathcal{M}_i\rvert = k$	2 (Th. 7)		$m - k + 1$ (Th. 8, 9, 10)

Table 2: Lower bounds on competitive ratios for the problem $P|\text{online}{-}r_i, p_i = 1, \mathcal{M}_i|F_{\max}$ with various processing set restrictions and depending on the type of algorithm.

allocated on different machines in OPT and FIFO, but it does modify neither their completion time nor the completion times of other tasks. If this is not the case, let T_i and T_j be two tasks in Π^{OPT} such that $r_i \leq r_j$, and where T_i starts after T_j ($\sigma_i \geq \sigma_j$). T_i can be on any machine, as well as T_j. Thus $\sigma_i + p \geq \sigma_j + p$, and then $C_i \geq C_j$ ($p_i = p_j = p$).

Their contribution to the objective is $\mathcal{F} = \max(C_i - r_i, C_j - r_j) = C_i - r_i$ because $r_i \leq r_j$ and $C_i \geq C_j$. Consider what happens if we swap T_i and T_j. Not that this is possible as T_j was originally started first although T_i is released before T_j. Their contribution to the maximum flow becomes $\mathcal{F}' = \max(C'_i - r_i, C'_j - r_j)$. By construction, $C'_i = C_j$ and $C'_j = C_i$. We have $C'_i - r_i = C_j - r_i \leq C_i - r_i$ (because $C_i \geq C_j$), and $C'_j - r_j = C_i - r_j \leq C_i - r_i$ (because $r_i \leq r_j$). Hence, $\mathcal{F}' \leq \mathcal{F}$.

It follows that we can transform Π^{OPT} in another optimal schedule Π^{FIFO} by swapping repeatedly non-sorted tasks. Then, FIFO is optimal. ■

As we proved the equivalence of FIFO and EFT in Section 4, all the results of the current section also apply to EFT.

6 Bounds under Processing Set Restrictions

Obviously, the problem $P|r_i, \mathcal{M}_i|F_{\max}$ is NP-hard in the offline context, that is, when all details on tasks are available beforehand. However, when considering tasks with unit processing times, Brucker et al. show that the problem $P|r_i, p_i = 1, \mathcal{M}_i|\sum w_i T_i$ is solvable in polynomial time [23]. Thus, $P|r_i, p_i = 1, \mathcal{M}_i|L_{\max}$ is also polynomial, and by setting the deadline $d_i = r_i$ for all tasks, it follows that $P|r_i, p_i = 1, \mathcal{M}_i|F_{\max}$ is polynomial.

Anand et al. show that $P|\text{online}{-}r_i, p_i = 1, \mathcal{M}_i|F_{\max}$ has a lower bound of $\Omega(m)$ on the competitive ratio of any online algorithm [12] (even the ones that do not have the Immediate Dispatch property). However, their proof is only valid for the general constraint $\mathcal{M}_i$, and it is unknown if special structures of the processing sets make the problem easier.

We provide here lower bounds on the competitive ratios of scheduling algorithms when considering that the processing sets follow a particular structure. Table 2 gives a summary of the results presented here.

We first study the *inclusive* structure of processing sets. We show in Theorem 3 that restricting to this structure reduces the lower bound on the competitive ratios to $\lfloor \log_2(m) + 1 \rfloor$ for immediate dispatch algorithms. This is also true for the *nested* and *interval* structures, which generalize the *inclusive* structure.

Theorem 3. *The competitive ratio of any immediate dispatch algorithm is at least* $\lfloor \log_2(m) + 1 \rfloor$ *for the problem* $P|\text{online}{-}r_i, p_i = 1, \mathcal{M}_i(\textit{inclusive})|F_{\max}$.

Proof: Let us assume that we work on a number of machines m that is a power of 2, i.e., $m = 2^{\lfloor \log_2(m') \rfloor}$, where m' is the actual number of machines. Let $\mathcal{D}$ be an arbitrary online immediate dispatch algorithm. We build the following adversary. For each ℓ such that $1 \leq \ell \leq \log_2(m)$, let $\mathcal{T}^{(\ell)}$ denote the set of $\frac{m}{2^\ell}$ tasks with $p_i = 1$ and $r_i = (\ell - 1)\varepsilon$ (where $0 < \varepsilon \ll 1$) for all $T_i \in \mathcal{T}^{(\ell)}$.

Then we define $\mathcal{M}^{(1)} = \{M_1, \ldots, M_m\}$ and for all $\ell > 1$, $\mathcal{M}^{(\ell)}$ denotes the subset of machines of $\mathcal{M}^{(\ell-1)}$ of size $\frac{m}{2^{\ell-1}}$ with at least $(\ell - 1)\frac{m}{2^{\ell-1}}$ allocated tasks in total after step $\ell - 1$ (we prove below that such a set exists). Finally, for each ℓ and for all $T_i \in \mathcal{T}^{(\ell)}$, we set $\mathcal{M}_i = \mathcal{M}^{(\ell)}$.

Let us prove by induction that the construction of $\mathcal{M}^{(\ell)}$ is valid, i.e., that such a subset exists for all $\ell > 0$. Note that as $\mathcal{D}$ is an immediate dispatch algorithm, all tasks of $\mathcal{T}^{(\ell)}$ are irremediably scheduled at time $(\ell - 1)\varepsilon$ on some machines of $\mathcal{M}^{(\ell)}$. For the construction of $\mathcal{M}^{(2)}$, we start from $\mathcal{M}^{(1)} = \{M_1, \ldots, M_m\}$ where $\frac{m}{2}$ tasks have been allocated on the first step. We select for $\mathcal{M}^{(2)}$ the subset of machines where these tasks have been allocated, possibly with additional machines to reach the proper size $\frac{m}{2}$.

We now assume that $\mathcal{M}^{(\ell)}$ has been constructed and prove that we can build $\mathcal{M}^{(\ell+1)}$. By induction, $\mathcal{M}^{(\ell)}$ has been allocated $(\ell - 1)\frac{m}{2^{\ell-1}}$ tasks up to step $\ell - 1$, and $\frac{m}{2^\ell}$ new tasks on step ℓ. This makes a total of $(2\ell - 1)\frac{m}{2^\ell}$ tasks. We select for $\mathcal{M}^{(\ell+1)}$ the $\frac{m}{2^\ell}$ machines that are the most loaded in $\mathcal{M}^{(\ell)}$. We consider two cases:

(i) Each of the selected machines has at least ℓ tasks. Then in total, we have at least $\ell\frac{m}{2^\ell}$ tasks, as requested.

(ii) There exists a selected machine with at most $\ell - 1$ tasks. This means that all non-selected machines have at most $\ell - 1$ tasks (otherwise, we would have selected one of them instead), for a total work (on the $\frac{m}{2^\ell}$ non-selected machines) of at most $(\ell - 1)\frac{m}{2^\ell}$ tasks. Thus, on selected machines, the number of tasks is at least

$$(2\ell - 1)\frac{m}{2^\ell} - (\ell - 1)\frac{m}{2^\ell} = \ell\frac{m}{2^\ell}.$$

At step $\log_2(m)$, $\mathcal{M}^{(\log_2(m))}$ is reduced to two machines, with at least $2(\log_2(m) - 1)$ allocated tasks, where a single task is scheduled. This leaves one machine with at least $\log_2(m)$ tasks, where we finally allocate one last task at time $\log_2(m)\varepsilon$, leading to a maximum flow of $\log_2(m) + 1 - \log_2(m)\varepsilon$. Thus, on all m' machines, as $\varepsilon \to 0$, we have a maximum flow of

$$\begin{aligned} \log_2(m) + 1 &= \log_2(2^{\lfloor \log_2(m') \rfloor}) + 1 \\ &= \lfloor \log_2(m') \rfloor + 1 = \lfloor \log_2(m') + 1 \rfloor . \end{aligned}$$

The optimal strategy consists in scheduling each set $\mathcal{T}^{(\ell)}$ on the machines of $\mathcal{M}^{(\ell)} \setminus \mathcal{M}^{(\ell+1)}$, for a max-flow of 1. ■

The previous result may be adapted for processing sets that do not present any particular structure, but have all the same size k.

Theorem 4. *The competitive ratio of any immediate dispatch algorithm is at least* $\lfloor \log_k(m) \rfloor$ *for the problem* $P|\text{online}{-}r_i, p_i = 1, \mathcal{M}_i, |\mathcal{M}_i| = k|F_{\max}$.

Proof: Let us assume that we work on a number of machines m that is a power of k, i.e., $m = k^{\lfloor \log_k(m') \rfloor}$, where m' is the actual number of machines. Let $\mathcal{D}$ be an arbitrary immediate dispatch algorithm. We proceed by building the following adversary. For each ℓ such that $1 \leq \ell \leq$

$\log_k(m)$, let $\mathcal{T}^{(\ell)}$ denote the set of $\frac{m}{k^\ell}$ tasks with $p_i = 1$ and $r_i = (\ell - 1)\varepsilon$ (where $0 < \varepsilon \ll 1$) for all $T_i \in \mathcal{T}^{(\ell)}$.

Note that as $\mathcal{D}$ is an immediate dispatch algorithm, all tasks of $\mathcal{T}^{(\ell)}$ are irremediably scheduled at time $(\ell-1)\varepsilon$. Then we define $\mathcal{M}^{(\ell)}$ as the set of machines on which the tasks of $\mathcal{T}^{(\ell)}$ are scheduled at this specific time, with the particular case $\mathcal{M}^{(0)} = M$. Finally, for each ℓ and for all $T_i \in \mathcal{T}^{(\ell)}$, we set $\mathcal{M}_i \subseteq \mathcal{M}^{(\ell-1)}$, with $|\mathcal{M}_i| = k$. Moreover, all processing sets of tasks that belong to the same subset are mutually disjoint, i.e., $\mathcal{M}_i \cap \mathcal{M}_j = \emptyset$ for all $T_i, T_j \in \mathcal{T}^{(\ell)}$ such that $i \neq j$.

$\mathcal{D}$ will be forced to schedule each set $\mathcal{T}^{(\ell)}$ on the exact same machines that are already busy with the tasks of the previous set $\mathcal{T}^{(\ell-1)}$. As all processing sets are mutually disjoint, we know that the tasks $\mathcal{T}^{(\ell)}$ are scheduled on $\left|\mathcal{T}^{(\ell)}\right| = \frac{m}{k^\ell}$ machines exactly. Moreover, there are exactly $\ell\frac{m}{k^\ell}$ waiting tasks on these machines at step ℓ. Thus, at the last step $\ell = \log_k(m)$, the completion time is $\log_k(m)$. Therefore, the maximum flow time is $\log_k(m) - (\log_k(m) - 1)\varepsilon$. Thus, on all m' machines, as $\varepsilon \to 0$, we have a maximum flow of

$$\begin{aligned}\log_k(m) &= \log_k(k^{\lfloor \log_k(m') \rfloor}) \\ &= \lfloor \log_k(m') \rfloor .\end{aligned}$$

The optimal strategy consists in scheduling each set $\mathcal{T}^{(l)}$ on the machines $\mathcal{M}^{(l-1)} \setminus \mathcal{M}^{(l)}$, for a max-flow of 1. ■

When considering online algorithms that do not have the Immediate Dispatch property (and thus may allocate tasks only when machines are available for computation), we can still prove a similar lower bound on the competitive ratio, as long as the processing sets are *nested.* The proof is an adaptation of Anand et al. [12], which did not consider any structure.

Theorem 5. *The competitive ratio of any online algorithm is at least* $\frac{1}{3}\lfloor \log_2(m) + 2 \rfloor$ *for the problem* $P|\text{online}-r_i, p_i = 1, \mathcal{M}_i\,\textit{(nested)}|F_{\max}$.

Proof: Let us assume that we work on a number of machines m that is a power of 2, i.e., $m = 2^{\lfloor \log_2(m') \rfloor}$, where m' is the actual number of machines. Let $\mathcal{N}$ be an arbitrary online scheduling algorithm. Machines are numbered from 1 to m, and let F be a number such that $F \geq \log_2(m) + 2$. We construct the following instance. At time $t_0 = 0$, we consider the interval of machines of size s_0 and starting from u_0 (that is, $\{M_{u_0}, M_{u_0+1}, \ldots, M_{u_0+s_0-1}\}$), denoted by $I(u_0, s_0)$, where $u_0 = 1$ and $s_0 = m$. We submit s_0 unit tasks at time t_0, with the processing set restriction $\mathcal{M}_i = I(u_0, s_0)$. Let $\mathcal{G}_{1,0}$ denote this set of tasks. For each machine $M_j \in I(u_0, s_0)$, we release one unit task at each time $t_0, t_0 + 1, \ldots, t_0 + F - 1$ and feasible only on the machine M_j. Let $\mathcal{G}_{2,0}$ denote this set. Note that at time $t_0 + F - 1$, algorithm $\mathcal{N}$ should have completed the tasks of $\mathcal{G}_{1,0}$, otherwise the maximum flow time would be greater than $\log_2(m) + 2$.

Now, for all $k > 0$, we set $t_k = t_{k-1} + F$ and $s_k = \frac{1}{2}s_{k-1}$. We choose u_k such that $u_{k-1} \leq u_k \leq u_{k-1} + s_{k-1} - s_k = u_{k-1} + s_k$ (in other words, $I(u_k, s_k)$ is a subinterval of $I(u_{k-1}, s_{k-1})$), and such that $|\mathcal{G}_{0,k}|$ is maximized, where $\mathcal{G}_{0,k} \subset \mathcal{G}_{2,k-1}$ is the set of tasks that are submitted before t_k but not completed at this time, and that can be executed on one machine only in the interval $I(u_k, s_k)$. Then we submit task sets $\mathcal{G}_{1,k}$ and $\mathcal{G}_{2,k}$ as previously: $\mathcal{G}_{1,k}$ is made of s_k tasks with processing set $I(u_k, s_k)$ released at time t_k, and $\mathcal{G}_{2,k}$ contains F tasks for each machine $M_j \in I(u_k, s_k)$ submitted at times $t_k, t_k + 1, \ldots, t_k + F - 1$ and that must be processed on M_j.

Figure 2 illustrates a schedule of the described instance.

We prove the following statements by induction: for all $k \geq 0$, (i) $s_k = m/2^k$ and (ii) there are at least ks_k uncompleted tasks on $I(u_k, s_k)$ at time t_k before sending $\mathcal{G}_{1,k}$ and $\mathcal{G}_{2,k}$, i.e., $|\mathcal{G}_{0,k}| \geq ks_k$.

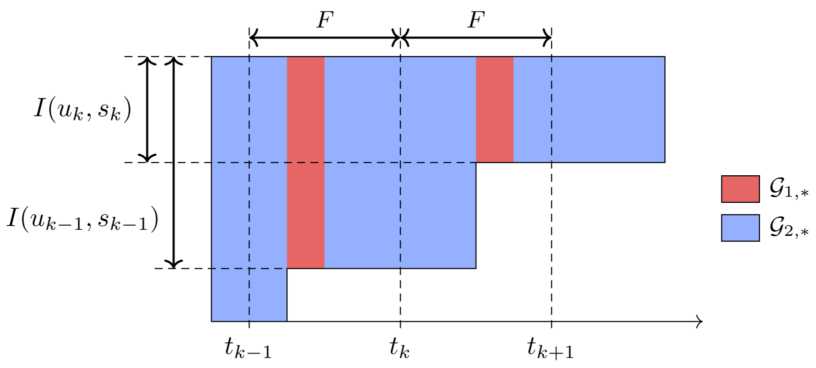


Figure 2: Example of scheduling for the described adversary.

For the base case ($k = 0$), we have $s_0 = m/2^0 = m$, and $\mathcal{G}_{0,k} = \emptyset$, so there is no completed task on $I(1, m)$ at time 0 before sending $\mathcal{G}_{1,0}$ and $\mathcal{G}_{2,0}$.

Now assume that $s_k = m/2^k$ is true at a certain step k. At step $k + 1$, we have $s_{k+1} = \frac{1}{2}s_k$ by definition, so $s_{k+1} = \frac{1}{2}(m/2^k) = m/2^{k+1}$, which proves the statement (i).

Suppose that there are at least ks_k uncompleted tasks on $I(u_k, s_k)$ at time t_k, i.e., $|\mathcal{G}_{0,k}| \geq ks_k$. Then we send $\mathcal{G}_{1,k}$ and $\mathcal{G}_{2,k}$, which means that there are at least

$$ks_k + s_k + Fs_k - Fs_k = (k + 1)s_k$$

uncompleted tasks on $I(u_k, s_k)$ at time $t_{k+1} = t_k + F$.

Now we choose the subinterval $I(u_{k+1}, s_{k+1}) \subset I(u_k, s_k)$ maximizing $|\mathcal{G}_{0,k+1}|$ at time t_{k+1}. Let us divide $I(u_k, s_k)$ into 2 disjoint subintervals of size $\frac{1}{2}s_k$ and by contradiction, assume that no such subinterval contains $(k + 1)\frac{1}{2}s_k$ uncompleted tasks, i.e., there are at most $(k + 1)\frac{1}{2}s_k - 1$ uncompleted tasks on each of these subintervals. Thus, there are at most $2\left((k + 1)\frac{1}{2}s_k - 1\right) = (k + 1)s_k - 2$ uncompleted tasks on $I(u_k, s_k)$, which contradicts the fact that $I(u_k, s_k)$ holds at least $(k + 1)s_k$ uncompleted tasks. Then, the chosen subinterval $I(u_{k+1}, s_{k+1})$ contains at least $(k+1)\frac{1}{2}s_k = (k+1)s_{k+1}$ uncompleted tasks at time t_{k+1} before sending $\mathcal{G}_{1,k+1}$ and $\mathcal{G}_{2,k+1}$ (that is, $|\mathcal{G}_{0,k+1}| \geq (k + 1)s_{k+1}$), which proves the statement (ii).

We stop when we reach the step k such that $s_k = 1$. This means that $m/2^k = 1$, i.e., $k = \log_2(m)$. Therefore, there remains at least $ks_k = \log_2(m)$ uncompleted tasks on an interval of size 1 at time t_k, plus 1 task of $\mathcal{G}_{1,k}$ and 1 task of $\mathcal{G}_{2,k}$, which gives a maximum flow time of at least $\log_2(m) + 2$. Thus, on all m' machines, we have a maximum flow of

$$\begin{aligned}\log_2(m) + 2 &= \log_2(2^{\lfloor \log_2(m') \rfloor}) + 2 \\ &= \lfloor \log_2(m') \rfloor + 2 = \lfloor \log_2(m') + 2 \rfloor .\end{aligned}$$

The optimal strategy consists, at each step $0 \leq k < \log_2(m)$, in executing all tasks of $\mathcal{G}_{1,k}$ on the subinterval $I(u_k, s_k) \setminus I(u_{k+1}, s_{k+1})$, for a max-flow of 3: tasks of $\mathcal{G}_{1,k}$ are scheduled first (with flow 2), followed by tasks of $\mathcal{G}_{2,k}$, which have a flow at most 3. ■

The case of disjoint processing sets is particular: we may apply a competitive algorithm independently on each set, which leads to an algorithm with adapted competitive ratio.

Theorem 6. *From any $f(m)$-competitive algorithm for the problem $P|\text{online}{-}r_i|F_{\max}$, we can design an algorithm with a competitive ratio of* $\max_i \{f(|\mathcal{M}_i|)\}$ *for the problem* $P|\text{online}{-}r_i, \mathcal{M}_i\,\textit{(disjoint)}|F_{\max}$.

Proof: Let $\mathcal{I}$ be an arbitrary instance of the problem $P|\text{online}-r_i, \mathcal{M}_i(\text{disjoint})|F_{\max}$, and let $\mathcal{N}$ be an $f(m)$-competitive algorithm for $P|\text{online}-r_i|F_{\max}$. By definition of the disjoint processing set restriction, we have $\mathcal{M}_i \cap \mathcal{M}_j = \emptyset$ or $\mathcal{M}_i = \mathcal{M}_j$ for all tasks T_i, T_j (with $i \neq j$) of the instance $\mathcal{I}$. Let $\mathcal{M}$ denote the set of all subsets $\mathcal{M}_i$.

Then, for all $\mathcal{M}_u \in \mathcal{M}$, we construct the set of tasks $\mathcal{T}_u = \{T_i \in T \text{ s.t. } \mathcal{M}_i = \mathcal{M}_u\}$. As $\mathcal{M}_u \cap \mathcal{M}_v = \emptyset$ for all $\mathcal{M}_u, \mathcal{M}_v \in \mathcal{M}$ such that $u \neq v$, we clearly have $\mathcal{T}_u \cap \mathcal{T}_v = \emptyset$. Moreover,

$$\bigcup_{\mathcal{M}_u \in \mathcal{M}} \mathcal{T}_u = T.$$

Hence, for all $\mathcal{M}_u \in \mathcal{M}$, $\mathcal{T}_u$ and $\mathcal{M}_u$ can clearly constitute an instance $\mathcal{I}_u$ of the problem $P|\text{online}-r_i|F_{\max}$. We design an online algorithm $\mathcal{N}'$ for the original problem by applying $\mathcal{N}$ in parallel to each instance $\mathcal{I}_u$.

By definition of the competitive ratio of $\mathcal{N}$, we have $F_{\max}^{\mathcal{N}}(\mathcal{I}_u) \leq f(|\mathcal{M}_u|)F_{\max}^{OPT}(\mathcal{I}_u)$, where OPT is an optimal offline strategy. As $\mathcal{I}_u$ is a subproblem of $\mathcal{I}$, we also have

$$F_{\max}^{OPT}(\mathcal{I}_u) \leq F_{\max}^{OPT'}(\mathcal{I})$$

for all $\mathcal{I}_u$, where OPT' is an optimal offline strategy built by applying OPT in parallel on each instance $\mathcal{I}_u$. Then, $F_{\max}^{\mathcal{N}}(\mathcal{I}_u) \leq f(|\mathcal{M}_u|)F_{\max}^{OPT'}(\mathcal{I})$, and

$$\begin{aligned} F_{\max}^{\mathcal{N}'}(\mathcal{I}) &= \max_u \left\{F_{\max}^{\mathcal{N}}(\mathcal{I}_u)\right\} \\ &\leq \max_u \left\{f(|\mathcal{M}_u|)\right\} F_{\max}^{OPT'}(\mathcal{I}). \end{aligned}$$ ■

This result has an important corollary for EFT on disjoint processing sets.

Corollary 1. EFT *is* $(3-2/\max|\mathcal{M}_i|)$*-competitive for the problem* $P|\text{online}-r_i, \mathcal{M}_i(\textit{disjoint})|F_{\max}$ *and* $(3-\frac{2}{k})$*-competitive when* $|\mathcal{M}_i| = k$ *for all* $\mathcal{M}_i$*.*

We now move to the study of processing sets that are intervals of fixed size, which we outlined in the introduction as being representative of the replication scheme used in key-value stores. We show that the competitive ratio of any algorithm (even without the Immediate Dispatch property) is not smaller than 2.

Theorem 7. *The competitive ratio of any online algorithm is at least 2 for the problem* $P|\text{online}-r_i, p_i = 1, \mathcal{M}_i(\textit{interval}), |\mathcal{M}_i| = k|F_{\max}$*.*

Proof: Let $\mathcal{N}$ be an arbitrary online algorithm. At time 0, the adversary sends one unit task T_1 with $\mathcal{M}_1 = \{M_2, M_3\}$. Now there are two cases: $\mathcal{N}$ executes this task (i) on M_2 or (ii) on M_3, and we denote its starting time by σ_1.

Let us assume $\mathcal{N}$ executes T_1 on M_2 (i). Then the adversary sends two unit tasks T_2 and T_3 at time $\sigma_1 + \varepsilon$ (where $0 < \varepsilon \ll 1$) with $\mathcal{M}_2 = \mathcal{M}_3 = \{M_1, M_2\}$. $\mathcal{N}$ will be forced to schedule at least one task at time $\sigma_1 + \varepsilon + 1$, and this task will complete at time $\sigma_1 + \varepsilon + 2$, for a max-flow of 2. The optimal schedule consists in executing T_1 on M_3 at time 0, to let the next two tasks execute on M_1 and M_2 at time ε, for a max-flow of 1. The case (ii) is proved analogously by sending two tasks on interval $\{M_3, M_4\}$. ■

The lower bound on the competitive ratio can be largely increased when considering immediate dispatch algorithms, and in particular EFT, as defined in Algorithm 2 in Section 4. Note that among immediate dispatch algorithms, EFT is a very reasonable candidate: when a new task is submitted, it is allocated to the machine that will finish it the earliest. Without processing set restrictions, this is known to produce a very good load balancing, as well as good performance for the max-flow [11]. Surprisingly, it turns out that this is not the case when adding processing set restrictions. We prove in Theorems 8, 9 and 10 that the competitive ratio of EFT is larger than $m-k+1$ in a variety of settings.

To exhibit this result, we need to focus on a specific tie-break function. We start by studying the Min tie-break function: in the set U_i of candidate machines that may finish task T_i at the earliest, we choose the machine with smallest index. The obtained algorithm is called EFT-Min (Algorithm 3) and its competitive ratio is bounded in Theorem 8.

Algorithm 3 EFT-Min

```
1: when a new task $T_i$ is released do
2:     Get $U_i'$ according to completion times of
       machines $\mathcal{M}_i$ (Equation (2))
3:     $u \leftarrow \textsc{Min}(U_i')$
4:     $\mu_i \leftarrow u$
5:     $\sigma_i \leftarrow \max(r_i, \mathcal{C}_{u,i-1})$
6:     Update the completion time of $M_u$
```

Theorem 8. *The competitive ratio of* EFT-Min *is at least* $m-k+1$ *for the problem* $P|\text{online}-r_i, p_i = 1, \mathcal{M}_i\textit{(interval)}, |\mathcal{M}_i| = k|F_{\max}$, *where* $1 < k < m$.

We show that the competitive ratio of EFT-Min is at least $m-k+1$ for the problem of minimizing max-flow when the processing set is an interval of size k, with $1 < k < m$, even when tasks are unitary. For ease of reading, we say that a given task T_i is of type λ if its processing interval restriction starts on machine M_λ, i.e., $\mathcal{M}_i = \{M_\lambda, \ldots, M_{\lambda+k-1}\}$, and we say that it is of type $\geq \lambda'$ (resp. $\leq \lambda'$) if $\lambda \geq \lambda'$ (resp. $\lambda \leq \lambda'$).

Let us build the following adversary (we illustrate an EFT-Min schedule of this adversary in Figure 3). At each time t, we send m tasks such that:

(i) for $1 \leq i \leq m-k$, task T_i is of type $m-k-i+2$ (blue task in Figure 3);

(ii) for $m-k < i \leq m$, task T_i is of type 1 (red task in Figure 3).

This adversary relies on the key observation that EFT-Min is naive: when several machines present the minimum load value among all machines, it will choose the first machine that satisfies its load-minimality criterion, i.e., the machine whose index is the lowest.

Note that at time t, just before sending the m next tasks, mt tasks have already been scheduled in Π^{EFT}, and each machine M_j completes at time $\mathcal{C}_{j,mt}$. Let $w_t(j) = \max(0, \mathcal{C}_{j,mt} - t)$ be the work allocated on machine M_j and waiting to be processed, just before the adversary releases the m tasks. We call w_t the schedule profile of EFT at time t.

The proof consists in showing that EFT-Min converges to a stable schedule profile w_τ such that for all j,

$$w_\tau(j) = \min(m-j, m-k).$$

Figure 4 shows an example of a schedule profile w_t, which is behind the stable profile w_τ.

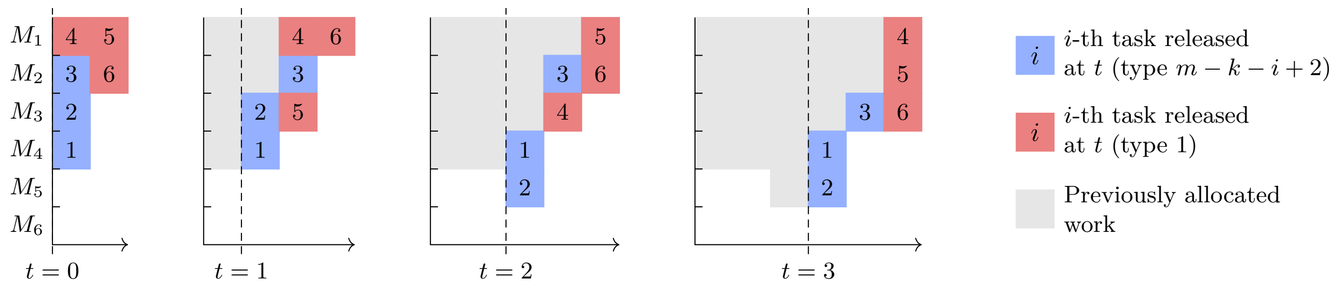

M_1
M_2
M_3
M_4
M_5
M_6
$t = 0$
$t = 1$
$t = 2$
$t = 3$
i-th task released at t (type $m - k - i + 2$)
i-th task released at t (type 1)
Previously allocated work

Figure 3: An EFT-Min schedule of the adversary from time $t = 0$ to $t = 3$, for $m = 6$ and $k = 3$. Colored tasks are released in-order at each time t.

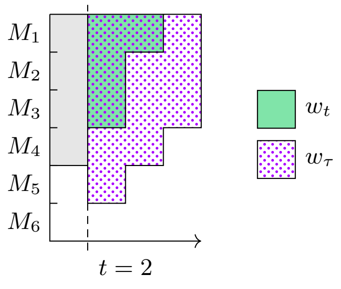


Figure 4: The schedule profile w_t of EFT-Min at time t (in green), just before the adversary sends m new tasks. w_t is strictly behind the stable profile w_τ we want to reach (in purple).

Definition 1. *For any $t \neq t'$, we say that*

(i) $w_t = w_{t'}$ if $w_t(j) = w_{t'}(j)$ for all j;

(ii) $w_t \leq w_{t'}$ if $w_t(j) \leq w_{t'}(j)$ for all j (w_t is behind $w_{t'}$);

(iii) $w_t < w_{t'}$ if $w_t \leq w_{t'}$ and there is at least one machine M_j such that $w_t(j) < w_{t'}(j)$ (w_t is strictly behind $w_{t'}$).

The proof consists in two phases: first, we show that when the schedule profile is strictly behind w_τ, there exists a future time such that the schedule profile is closer to w_τ (Lemma 3); second, we show that at any time, either we can find a past time such that the schedule profile exceeds w_τ, or the current schedule profile is behind w_τ (Lemma 4).

Before we dive into the proof, we start with the following lemma, which proves that for any t, w_t is a non-increasing function. This will be of particular importance when proving Lemma 3.

Lemma 2. *At any time t and for all j such that $1 \leq j < m$, $w_t(j+1) \leq w_t(j)$.*

Proof: Let us proceed by induction.

Base case ($t = 0$). No task arrived yet, so $w_0(j) = 0$ for all machines j.

Induction step. Now we assume that for a given t, $w_t(j+1) \leq w_t(j)$ for all j such that $1 \leq j < m$. By contradiction, suppose there exists j such that $w_{t+1}(j+1) > w_{t+1}(j)$. We begin by showing that, as a consequence, only one task can have been scheduled on machine M_{j+1} at time t, which will lead to a contradiction.

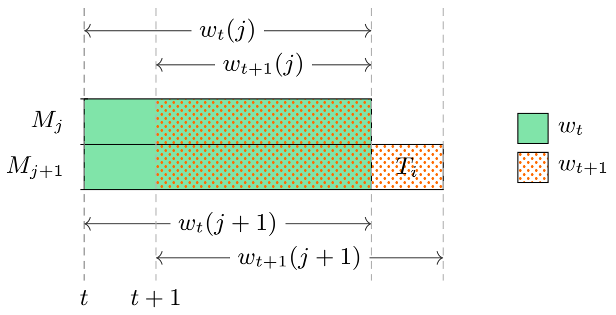


Figure 5: Schedule profiles on M_j and M_{j+1} at times t and $t+1$, under the described hypotheses.

Let T_i be the last allocated task on machine M_{j+1}. By induction hypothesis, we know that $w_t(j+1) \leq w_t(j)$, and we assumed that $w_{t+1}(j+1) > w_{t+1}(j)$, thus T_i has been scheduled at a time comprised between t and $t+1$ (let $t+\varepsilon$ denote this specific time).

If we had $w_{t+\varepsilon}(j) > w_{t+\varepsilon}(j+1)$, then T_i could not be the last allocated task on M_{j+1} at time $t+1$, because we assumed that $w_{t+1}(j+1) > w_{t+1}(j)$, and EFT-Min is an immediate dispatch algorithm. Therefore, we necessarily had $w_{t+\varepsilon}(j) \leq w_{t+\varepsilon}(j+1)$ at time $t+\varepsilon$, just before scheduling T_i. We can deduce that we have $\mathcal{M}_i = \{M_{j+1}, \dots, M_{j+k}\}$, otherwise we would have scheduled T_i on the less-loaded machine M_j (then we say that T_i is of type $j+1$). Furthermore, all machines $M_{j+2}, \dots, M_{j+k}$ were at least as much loaded as M_{j+1} at time $t+\varepsilon$ ($w_{t+\varepsilon}(j') \geq w_{t+\varepsilon}(j+1)$ for all j' such that $j+1 < j' \leq j+k$), otherwise we would not have scheduled T_i on M_{j+1}.

By construction of the adversary, the tasks sent before T_i at time t cannot have been placed on M_{j+1} because their interval restriction starts after M_{j+1} (they are of type $\geq j+2$). Moreover, the tasks sent after T_i at time t cannot have been placed on M_{j+1} as well (otherwise, T_i would not be the last task on M_{j+1}). Hence, T_i is the only task scheduled on M_{j+1} between times t and $t+1$.

We consider two cases:

(i) First, suppose that $w_t(j+1) = 0$. This means that EFT-Min makes T_i starting at time t on M_{j+1}, and then T_i completes at time $t+1$. We proved that T_i is the only task that is scheduled on M_{j+1} at time t, and as it completes at time $t+1$, we can say that there is no remaining work at this exact time, i.e., $w_{t+1}(j+1) = 0$. This contradicts our hypothesis $w_{t+1}(j+1) > w_{t+1}(j) \geq 0$.

(ii) Now suppose that $w_t(j+1) > 0$. Following our two hypothesis $w_t(j+1) \leq w_t(j)$ and $w_{t+1}(j+1) > w_{t+1}(j)$, and from the consequent fact that exactly one task has been scheduled on M_{j+1} between t and $t+1$, the only way to match all these assumptions is when there is as much waiting work on M_j as on M_{j+1} at time t, i.e., $w_t(j) = w_t(j+1)$), and no task is scheduled on M_j between t and $t+1$. Figure 5 helps to visualize the described situation.

    But the last task T_i scheduled on M_{j+1} is of type $j+1$, which means that, by construction, at least one task of type j arrived after T_i at time t. We showed that all machines $M_{j+2}, \dots, M_{j+k}$ were at least as much loaded as M_{j+1} at time $t+\varepsilon$, thus they were at least as much loaded as M_j. As a consequence, at least one task must have been scheduled by EFT-Min on M_j between $t+\varepsilon$ and $t+1$, which is a contradiction. ■

Now we are able to show the first part of our proof: when the schedule profile of EFT-Min is strictly behind the stable profile w_τ, there must exist a future time t' such that the waiting work volume is greater than the current volume, i.e., $\sum_j w_t(j) < \sum_j w_{t'}(j)$.

Lemma 3. *At any time t such that $w_t < w_\tau$, there exists a time $t' > t$ such that:*

(i) for all t_1 such that $t \leq t_1 < t'$, we have $\sum_j w_{t_1}(j) = \sum_j w_t(j)$;

(ii) $\sum_j w_t(j) < \sum_j w_{t'}(j)$.

Proof: **Idleness property.** The first thing to notice is that when w_t is empty for a given machine that is not the last one, i.e., there exists $j < m$ such that $w_t(j) = 0$, we know that all subsequent machines have no remaining work to do as well: $w_t(j') = 0$ for all $j' > j$ (Lemma 2). When it happens, EFT-MIN will not schedule any task on the last machine, because the only eligible task is the first one, which is of type $m - k + 1$; that task will be scheduled on $M_{\max(j,m-k+1)}$ (the first lightly-loaded compatible machine) at time t.

Therefore, m new tasks are released by the adversary and at most $m - 1$ tasks are processed (no work can be done by the last machine), so we have

$$\sum_j w_{t+1}(j) \geq \sum_j w_t(j) + m - (m - 1),$$

and thus $\sum_j w_{t+1}(j) > \sum_j w_t(j)$.

This means that if $w_t(j) = 0$ for some $j < m$, we have $\sum_j w_{t+1}(j) > \sum_j w_t(j)$. The proof is mainly based on this useful property that we call the Idleness Property. If the schedule profile is strictly behind w_τ, we will show that there must exist a plateau on some machines, and this plateau will necessarily propagate on next machines step by step, until we reach a time t such that $w_t(m - 1) = 0$.

Existence of a plateau. Now suppose that the schedule profile w_t is strictly behind w_τ ($w_t < w_\tau$). By Definition 1, this means that $w_t(j) \leq w_\tau(j)$ for all j, and there is at least one machine $M_{j'}$ such that

$$w_t(j') < w_\tau(j'). \tag{5}$$

Let j' be the highest index of such a machine; then we have

$$w_t(j) = w_\tau(j) \text{ for all } j > j', \tag{6}$$

and in particular, $w_t(j' + 1) = w_\tau(j' + 1)$. Let us show that there is a plateau on $M_{j'}$ and $M_{j'+1}$ at time t, i.e., that $w_t(j') = w_t(j' + 1)$. First, note that by definition of w_τ, we have

$$w_\tau(j) = w_\tau(j + 1) + 1 \text{ for all } k \leq j < m. \tag{7}$$

We have $j' \geq k$, because if $w_t(j) < w_\tau(j)$ for some $j < k$, schedule profiles of all machines $M_{j+1}, \ldots, M_k$ are also strictly behind the stable profile w_τ (by Lemma 2 and definition of w_τ), and we defined j' as the highest index; furthermore, $j' < m$, because we assumed that $w_t(j') < w_\tau(j')$ and by definition $w_\tau(m) = 0$ ($w_t(m)$ cannot be lower than 0). Therefore,

$$\begin{aligned} w_t(j' + 1) \leq w_t(j') &< w_\tau(j') && \text{(by Lemma 2 and Eq. (5))} \\ &< w_\tau(j' + 1) + 1 && \text{(by Eq. (7))} \\ &< w_t(j' + 1) + 1 && \text{(by Eq. (6))} \end{aligned}$$

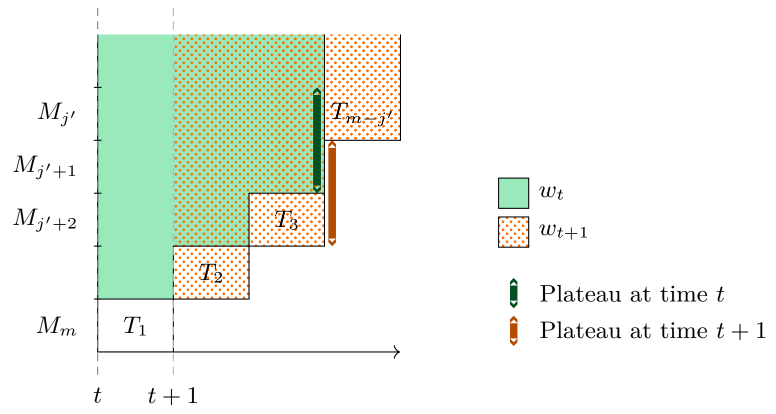


Figure 6: Propagation of the plateau from machines $M_{j'}$ and $M_{j'+1}$ at time t to machines $M_{j'+1}$ and $M_{j'+2}$ at time $t+1$.

which gives

$$w_t(j'+1) \leq w_t(j') \leq w_t(j'+1), \tag{8}$$

and then $w_t(j') = w_t(j'+1)$.

By definition, $w_\tau(m) = 0$, and as $w_t < w_\tau$, $w_t(m) = 0$. If $j' = m-1$, we have $w_t(j') = w_t(m-1) = w_t(m) = 0$, and by the Idleness Property, the conclusion is immediate. Otherwise, $j' < m-1$, so $w_t(m-1) = w_\tau(m-1) = 1$. By Lemma 2, $w_t(j) \geq 1$ for all $j < m$, and then EFT-Min will schedule the first task on the last machine. Overall, m tasks will be processed at time t, and m tasks are sent by the adversary; therefore, $\sum_j w_{t+1}(j) = \sum_j w_t(j) - m + m = \sum_j w_t(j)$.

Propagation of the plateau. Now we show that the plateau propagates on the next machine in the next step, i.e., as $j' < m-1$, $w_t(j') = w_t(j'+1)$ implies $w_{t+1}(j'+1) = w_{t+1}(j'+2)$.

By Eq. (6), $w_t(j) = w_\tau(j)$ for all $j > j'$, which means that the first $m - j' - 1$ tasks will be scheduled on their last machine ($\mu_{mt+i} = m - i + 1$ for each $1 \leq i < m - j'$). The corresponding machines will process one task at time t. Thus, $w_{t+1}(j) = w_t(j) - 1 + 1 = w_t(j)$ for all $j > j'+1$, and in particular,

$$w_{t+1}(j'+2) = w_t(j'+2). \tag{9}$$

As $w_t(j') = w_t(j'+1)$, the $(m-j')$-th task will not be scheduled on $M_{j'+1}$ (the index of the machine it will be placed on is at most j': $\mu_{mt+m-j'} < j'+1$). All remaining tasks will be scheduled on machines $M_1, \ldots, M_{j'}$, because they are of type $\leq j' - k + 1$. Figure 6 shows the propagation process.

Then $M_{j'+1}$ does not receive any additional task at time t, but it still processes one task at this time, so we have:

$$\begin{aligned}
w_{t+1}(j'+1) &= w_t(j'+1) - 1 \\
&= w_\tau(j'+1) - 1 && \text{(by Eq. (6))} \\
&= w_\tau(j'+2) && \text{(by Eq. (7))} \\
&= w_t(j'+2) && \text{(by Eq. (6))} \\
&= w_{t+1}(j'+2). && \text{(by Eq. (9))}
\end{aligned}$$

This shows that the plateau propagates on machines $M_{j'+1}$ and $M_{j'+2}$ at time $t+1$. By repeating the process, we reach a time at which $j'+1 = m-1$ and $j'+2 = m$, thus $w_t(m-1) = w_t(m) = 0$, and the Idleness Property applies. This concludes the proof. ■

The second phase of our proof consists in showing that either there exists a past time such that the schedule profile exceeds the stable profile w_τ, or the current profile is behind w_τ.

Lemma 4. *At any time t, either (i) there exists a time $t' \leq t$ such that $w_{t'}(j) > m-k$ for some j or (ii) $w_t \leq w_\tau$.*

Proof: Let us proceed by induction.

Base case ($t = 0$). Obviously, the base case is true ($w_0 = 0 \leq w_\tau$).

Induction step (case (i)). First suppose there exists a time $t' \leq t$ such that $w_{t'}(j) > m-k$ for some j. This is obviously still true at time $t+1$.

Induction step (case (ii)). Now suppose that $w_t \leq w_\tau$ for some t. By contradiction, let us assume that there exists a machine M_j such that $w_{t+1}(j) > w_\tau(j)$. Combined to the fact that $w_t(j) \leq w_\tau(j)$, we have $w_{t+1}(j) \geq w_t(j) + 1$. Let q denote the number of tasks scheduled on M_j at time t, such that $w_t(j) + q - 1 = w_{t+1}(j)$. Then, $w_t(j) + q - 1 \geq w_t(j) + 1$, i.e., $q \geq 2$.

So at least 2 tasks must have been scheduled on machine M_j at time t. Let j be the highest index of such a machine. Two subcases arise:

(a) $j \leq k$. Then by construction $w_\tau(j) = m-k$, and we have $w_{t+1}(j) > w_\tau(j) = m-k$. This proves the induction.

(b) $j > k$. By induction hypothesis, we know that $w_t(m) = 0$ (because $w_\tau(m) = 0$), and by construction, at most one task can be scheduled on the last machine at time t. Therefore, $w_{t+1}(m) = 0$, so $j < m$. Let T_i be the last allocated task on M_j, with σ_i its starting time:

$$\sigma_i = t + 1 + w_{t+1}(j) - 1 = t + w_{t+1}(j).$$

Let λ_i be the type of T_i, i.e., $\mathcal{M}_i = \{M_{\lambda_i}, \dots, M_{\lambda_i+k-1}\}$. As T_i has been allocated to M_j, we necessarily have $\lambda_i \leq j \leq \lambda_i + k - 1$.

Suppose $\lambda_i = j$. By construction, all tasks sent before T_i at time t cannot have been scheduled on M_j, because their machine interval starts after M_{λ_i}. As T_i is the last task of M_j, no task sent after T_i at time t can have been scheduled on this machine. Then T_i is the only task scheduled on M_j between t and $t+1$, which contradicts the fact that at least 2 tasks must have been scheduled on M_j. Hence, $\lambda_i < j$.

Now, as T_i has been allocated on M_j and not on M_{j-1}, we know there was already a task $T_{i'}$ on M_{j-1} when the scheduling of T_i occurred, with $\sigma_{i'} = \sigma_i = t + w_{t+1}(j)$.

At time t, just before the adversary releases the m tasks, M_{j-1} completes at time $\mathcal{C}_{j-1,mt} = t + w_t(j-1)$. We have $w_t(j-1) \leq w_\tau(j-1)$ (induction hypothesis); we supposed that $w_{t+1}(j) > w_\tau(j)$; finally, $w_\tau(j-1) = w_\tau(j) + 1$ (by construction of w_τ). Therefore,

$$\begin{aligned} t + w_t(j-1) &\leq t + w_\tau(j-1) \\ &\leq t + w_\tau(j) + 1 \\ &< t + w_{t+1}(j) + 1 = \sigma_{i'} + 1, \end{aligned}$$

which means that $\sigma_{i'} \geq t + w_t(j-1)$. In other words, $T_{i'}$ starts after time $\mathcal{C}_{j-1,mt}$. Hence, the scheduling of $T_{i'}$ occurred between t and $t+1$, before the scheduling of T_i ($t \leq \rho_{i'} < \rho_i < t+1$). Let $\lambda_{i'}$ be the type of $T_{i'}$. We necessarily have $\lambda_{i'} > \lambda_i$.

If $k = 2$, this is a contradiction, because $T_{i'}$ cannot have been scheduled on M_{j-1} (we proved that $\lambda_i < j$, so $\mathcal{M}_i = \{M_{j-1}, M_j\}$, and then $\lambda_{i'} > j-1$).

If $k > 2$, we deduce that $T_{i'}$ has been scheduled on M_{j-1} because all the machines $M_j, \dots, M_{\lambda_{i'}+k-1}$ are planned to finish at or after time $\sigma_{i'}$. In particular, we have

$$\mathcal{C}_{j+1,i'} \geq \sigma_{i'} \geq t + w_{t+1}(j) > t + w_\tau(j).$$

Then,

$$\mathcal{C}_{j+1,i'} - 1 > t + w_\tau(j) - 1 = t + w_\tau(j+1).$$

Moreover, $\mathcal{C}_{j+1,i'} \leq \mathcal{C}_{j+1,m(t+1)} = t + 1 + w_{t+1}(j+1)$. Therefore,

$$t + w_{t+1}(j+1) > t + w_\tau(j+1),$$

i.e., $w_{t+1}(j+1) > w_\tau(j+1)$. This is a contradiction, because we had chosen j to be the highest index such that $w_{t+1}(j) > w_\tau(j)$. This concludes the proof. ■

Proof of Theorem 8: To exhibit the lower bound of $m-k+1$ on the competitive ratio of EFT-Min, we first show there exists a time t such that $w_t = w_\tau$ or $w_t(j) > m-k$ for some j.

For a given time t, we know by Lemma 4 that either (i) there exists a time $t' \leq t$ such that $w_{t'}(j) > m-k$ for some j or (ii) $w_t \leq w_\tau$. If case (i) is true, then we found a time t such that $w_t(j) > m-k$ for some j. If case (ii) is true, either $w_t < w_\tau$ or $w_t = w_\tau$. Suppose that $w_t < w_\tau$. Then by Lemma 3, we can find a future time t' such that $\sum w_{t'}(j) > \sum w_t(j)$. Therefore, while we have $w_t < w_\tau$, we can always find a future time such that the schedule profile is closer to w_τ. If we proceed step by step, we necessarily reach a time t' such that $w_{t'} = w_\tau$. This proves our initial claim.

Now, if $w_t(j) > m-k$ for some t, j, there exists a task T_i such that $F_i \geq m-k+1$. If $w_t = w_\tau$ for some t, EFT-Min will schedule one task on each machine (by definition of the adversary and w_τ). Hence, the k last tasks will be allocated on the k first machines, and they will have a flow time of $m-k+1$. In any case, we have $F_{\max} \geq m-k+1$.

On the described instance, at each time step, the optimal strategy consists in scheduling each task of type $\geq k+1$ on the compatible machine of highest index. This allows reserving the k first machines to the k last tasks, and avoid any delay accumulation. Therefore, for all tasks T_i, we have $F_i^{OPT} = 1$, and then $F_{\max}^{OPT} = 1$. ■

The previous bound on the competitive ratio of EFT-Min can be extended to the case where EFT uses a random tie-break function Rand, and we call this algorithm EFT-Rand (Algorithm 4). The only condition for Theorem 9 to hold is that among a set of candidate machines, the random tie-break function chooses each machine with positive probability, i.e., no machine is systematically discarded when it is a possible candidate.

Before starting the proof, we begin with the definition of a set of events, and we describe the class of random tie-break functions that we consider.

Definition 2 (Random events.). *We define the following set of events:*

Algorithm 4 EFT-Rand

```
1: when a new task $T_i$ is released do
2:     Get $U_i'$ according to completion times of
       machines $\mathcal{M}_i$ (Equation (2))
3:     $u \leftarrow$ Rand$(U_i')$
4:     $\mu_i \leftarrow u$
5:     $\sigma_i \leftarrow \max(r_i, \mathcal{C}_{u,i-1})$
6:     Update the completion time of $M_u$
```

(i) $A_{t,i}$: *the event for which the task* T_i*, released at time* t*, is scheduled on its last machine, i.e.,* $\mu_{mt+i} = m - i + 1$.

(ii) A_t: *the event for which all tasks* $T_{i \le m-k}$ *released at* t *are scheduled on their last machine, i.e.,*

$$A_t = \bigcap_{i=1}^{m-k} A_{t,i}.$$

(iii) $\mathcal{A}_t$: *the event for which all tasks released after time* t *are scheduled on their last machine, i.e.,*

$$\mathcal{A}_t = \bigcap_{t'>t} A_{t'}.$$

Definition 3 (Rand policy.)**.** Rand *corresponds to any randomized policy for which there exists a constant* $\theta > 0$ *such that* $P(A_t) \le 1 - \theta$ *if there exists a task* $T_{i \le m-k}$ *released at time* t *such that* $|U_{mt+i}| > 1$.

We assume that all events A_t *are mutually independent, i.e., for a given* t,

$$P(\mathcal{A}_t) = \prod_{t'>t} P(A_{t'}).$$

Theorem 9. *The competitive ratio of* EFT-Rand *is at least* $m - k + 1$ *(almost surely) for the problem* $P|\text{online}-r_i, p_i = 1, \mathcal{M}_i\,\textit{(interval)}, |\mathcal{M}_i| = k|F_{\max}$*, where* $1 < k < m$*. In other words, there exists an instance for which we have*

$$P\left(F_{\max} \ge (m - k + 1)F_{\max}^{OPT}\right) = 1.$$

Definition 4. *We define the weighted distance on machine* M_j *at time* t *as*

$$\varphi_t(j) = 2^{w_\tau(j)}(m - k + 1 - w_t(j)).$$

For any j_1, j_2 *such that* $1 \le j_1 \le j_2 \le m$*, the partial weighted distance between* M_{j_1} *and* M_{j_2} *at time* t *is defined as*

$$\Phi_t(j_1, j_2) = \sum_{j=j_1}^{j_2} \varphi_t(j),$$

and the total weighted distance is denoted by $\Phi_t = \Phi_t(1, m)$.

Lemma 5. *At any time t, (i) if there exists a task $T_{i\leq m-k}$ released at t such that $\mu_{mt+i} \neq m-i+1$ then $\Phi_{t+1} < \Phi_t$, (ii) otherwise $\Phi_{t+1} \leq \Phi_t$.*

Proof: **Case (i).** At a given time t, suppose there exists at least one task $T_{i\leq m-k}$ released at t and that is not scheduled on its last machine, i.e., $\mu_{mt+i} \neq m-i+1$. Let i be the highest index of such a task. We will study the value of $\Phi_t - \Phi_{t+1}$ in two steps: first, the value of $\Phi_t(1, m-i) - \Phi_{t+1}(1, m-i)$ on machines $M_1, \dots, M_{m-i}$; second, the value of $\Phi_t(m-i+1, m) - \Phi_{t+1}(m-i+1, m)$ on machines $M_{m-i+1}, \dots, M_m$.

From 1 to $m-i$. We choose i to be the highest index such that T_i is not put on its last machine; this means that all tasks $T_{i'}$ such that $i < i' \leq m-k$ are scheduled on their last machine $M_{m-i'+1}$, and the last k tasks are scheduled on any of the first k machines (because they are of type 1). In summary, all tasks $T_{i'}$ such that $i \leq i' \leq m$ are scheduled on the first $m-i$ machines, and there are $m-i+1$ such tasks.

Any machine M_j among $M_1, \dots, M_{m-i}$ can process at most 1 task between t and $t+1$. Let $q_{t,j}$ be the number of tasks released at time t and scheduled on M_j.

Hence, $w_{t+1}(j) \geq w_t(j) - 1 + q_{t,j}$, and then

$$2^{w_\tau(j)}(m-k+1-w_{t+1}(j)) \leq 2^{w_\tau(j)}(m-k+1-w_t(j)+1-q_{t,j}).$$

Therefore, $\varphi_{t+1}(j) \leq \varphi_t(j) + 2^{w_\tau(j)} - 2^{w_\tau(j)} q_{t,j}$.

By combining, we have

$$\sum_{j=1}^{m-i} \varphi_{t+1}(j) \leq \sum_{j=1}^{m-i} \varphi_t(j) + \sum_{j=1}^{m-i} 2^{w_\tau(j)} - \sum_{j=1}^{m-i} 2^{w_\tau(j)} q_{t,j}.$$

Note that

$$\sum_{j=1}^{m-i} 2^{w_\tau(j)} q_{t,j} \geq \sum_{i'=i}^{m} 2^{w_\tau(\mu_{mt+i'})},$$

as we have shown that at least the last $m-i+1$ tasks released at t are scheduled on the first $m-i$ machines.

Then,

$$\Phi_{t+1}(1, m-i) \leq \Phi_t(1, m-i) + \sum_{j=1}^{m-i} 2^{w_\tau(j)} - \sum_{i'=i}^{m} 2^{w_\tau(\mu_{mt+i'})}. \tag{10}$$

Now we notice that

$$\begin{aligned}\sum_{i'=i}^{m} 2^{w_\tau(\mu_{mt+i'})} &= 2^{w_\tau(\mu_{mt+i})} + \sum_{i'=i+1}^{m-k} 2^{w_\tau(\mu_{mt+i'})} + \sum_{i'=m-k+1}^{m} 2^{w_\tau(\mu_{mt+i'})} \\ &= 2^{w_\tau(\mu_{mt+i})} + \sum_{j=k+1}^{m-i} 2^{m-j} + \sum_{j=1}^{k} 2^{m-k} \\ &= 2^{w_\tau(\mu_{mt+i})} + \sum_{j=1}^{m-i} 2^{w_\tau(j)}.\end{aligned}$$

Finally, by simplifying Eq. (10),

$$\Phi_t(1, m-i) - \Phi_{t+1}(1, m-i) \geq 2^{w_\tau(\mu_{mt+i})}. \tag{11}$$

From $m-i+1$ to m. We saw earlier that the last $m-i+1$ tasks released at time t must have been scheduled on the first $m-i$ machines. We deduce that only the first $i-1$ tasks can have been put on the last i machines. There are more machines than tasks; therefore, there exists at least one machine M_j such that $j > m-i$ that did not receive any task at time t. M_j can process at most one task between t and $t+1$, so we have $w_{t+1}(j) \geq w_t(j) - 1$, and then

$$\varphi_t(j) - \varphi_{t+1}(j) \geq -2^{w_\tau(j)}.$$

In the worst case, all machines M_j such that $j > m-i$ receive no task. Then we have

$$\sum_{j=m-i+1}^{m} (\varphi_t(j) - \varphi_{t+1}(j)) \geq -\sum_{j=m-i+1}^{m} 2^{w_\tau(j)},$$

and then

$$\Phi_t(m-i+1, m) - \Phi_{t+1}(m-i+1, m) \geq -\sum_{j=m-i+1}^{m} 2^{w_\tau(j)}. \tag{12}$$

Now we combine Eq. (11) and (12), and we get

$$\Phi_t - \Phi_{t+1} \geq 2^{w_\tau(\mu_{mt+i})} - \sum_{j=m-i+1}^{m} 2^{w_\tau(j)}.$$

Because $\mu_{mt+i} \leq m-i$, we have $2^{w_\tau(\mu_{mt+i})} \geq 2^{w_\tau(m-i)}$, and as $i \leq m-k$, $2^{w_\tau(m-i)} = 2^i$ and $m-i+1 \geq k+1$. Therefore,

$$\Phi_t - \Phi_{t+1} \geq 2^i - \sum_{j=m-i+1}^{m} 2^{m-j} = 2^i - \sum_{j'=0}^{i-1} 2^{j'} = 2^i - \left(2^i - 1\right) = 1,$$

and we conclude that $\Phi_t - \Phi_{t+1} > 0$.

Case (ii). Now suppose that at a given time t, all tasks $T_{i \leq m-k}$ released at t are scheduled on their last machine, i.e., $\mu_{mt+i} = m-i+1$.

From 1 to k. Only the last k tasks released at time t can have been put on the first k machines. Moreover, these machines can process at most k tasks between t and $t+1$. Hence,

$$\sum_{j=1}^{k} w_{t+1}(j) \geq \sum_{j=1}^{k} w_t(j) + k - k = \sum_{j=1}^{k} w_t(j),$$

and then

$$2^{m-k} \sum_{j=1}^{k} (m-k+1-w_{t+1}(j)) \leq 2^{m-k} \sum_{j=1}^{k} (m-k+1-w_t(j)),$$

which gives

$$\sum_{j=1}^{k} 2^{w_\tau(j)} (m-k+1-w_{t+1}(j)) \leq \sum_{j=1}^{k} 2^{w_\tau(j)} (m-k+1-w_t(j)).$$

Therefore,

$$\Phi_t(1,k) - \Phi_{t+1}(1,k) \geq 0. \tag{13}$$

From $k+1$ to m. All tasks $T_{i\leq m-k}$ are put on their last machines. Then all machines $M_{k+1},\dots,M_m$ receive exactly one task at time t, and we have $w_{t+1}(j) = w_t(j)$ for these machines, i.e., $\varphi_t(j) - \varphi_{t+1}(j) = 0$.

Hence,

$$\sum_{j=k+1}^{m}(\varphi_t(j) - \varphi_{t+1}(j)) = 0,$$

and then

$$\Phi_t(k+1,m) - \Phi_{t+1}(k+1,m) = 0. \tag{14}$$

By combining Eq. (13) and (14), we get $\Phi_t - \Phi_{t+1} \geq 0$. ■

Lemma 6. *At any time t, if $\mu_{mt+i} = m-i+1$ and $|U_{mt+i}| = 1$ for all task $T_{i\leq m-k}$ released at t, then $w_t(j+1) < w_t(j)$ for all $k \leq j < m$.*

Proof: Suppose that for a given time t, all tasks $T_{i\leq m-k}$ are scheduled on their last machine. This means that all machines $M_{k+1},\dots,M_m$ receive only one task at time t. Let T_i be such a task (we have $\mu_{mt+i} = m-i+1$). Suppose that there is no tie for T_i ($|U_{mt+i}| = 1$). By definition of the tie, we have

$$\mathcal{C}_{m-i+1,mt+i-1} < \mathcal{C}_{j,mt+i-1}$$

for all j such that $m-k-i+2 \leq j < m-i+1$. Moreover, we have $\mathcal{C}_{m-i+1,mt+i-1} = \mathcal{C}_{m-i+1,mt}$ and $\mathcal{C}_{j,mt+i-1} = \mathcal{C}_{j,mt}$, because all tasks $T_{i'<i}$ have been put on their last machine $M_{m-i'+1}$, and we have $m-i'+1 > m-i+1$.

Hence,

$$\mathcal{C}_{m-i+1,mt} < \mathcal{C}_{j,mt},$$

and then

$$t + w_t(m-i+1) < t + w_t(j),$$

which gives $w_t(m-i+1) < w_t(j)$. In particular, $w_t(m-i+1) < w_t(m-i)$. As this is true for all $1 \leq i \leq m-k$, we have $w_t(j+1) < w_t(j)$ for all $k \leq j < m$. ■

Proof of Theorem 9: It is clear from Lemma 5 that Φ is non-increasing: at any time t, $\Phi_{t+1} \leq \Phi_t$. Then there are two cases: either (i) we can find a time $t' > t$ such that $\Phi_{t'} < \Phi_t$ for all t, or (ii) there exists a time t such that $\Phi_{t'} = \Phi_t$ for all $t' > t$.

Case (i). Suppose that for all t, there exists a future time $t' > t$ such that $\Phi_{t'} < \Phi_t$. As $\Phi_t \in \mathbb{Z}$ for all t, there must exist a time t^* such that $\Phi_{t^*} \leq 0$, i.e., $\sum_j \varphi_{t^*}(j) \leq 0$. Then, there exists at least one j such that $\varphi_{t^*}(j) \leq 0$. By definition, we deduce that $m-k+1-w_{t^*}(j) \leq 0$, thus $w_{t^*}(j) \geq m-k+1$.

The last scheduled task T_i on M_j will complete at time $t^*+m-k+1$, and we have $r_i \leq t^*$. Therefore, $F_{\max} \geq F_i \geq m-k+1$.

Case (ii). Now suppose that there exists a time t such that $\Phi_{t'} = \Phi_t$ for all future time $t' > t$. By contraposition of Lemma 5, for all $t' > t$, we have $\mu_{mt'+i} = m - i + 1$ for all $T_{i \leq m-k}$ released at time t', i.e., the first $m-k$ tasks released at each t' are put on their last machine.

Suppose by contradiction that for all $t' > t$, there is a task $T_{i \leq m-k}$ released at t' such that $|U_{mt'+i}| > 1$. Then we have $P(A_{t'}) \leq 1 - \theta$, therefore $P(\mathcal{A}_t) = 0$ (because $\theta > 0$).

Then, with probability 1, there exists at least one time $t' > t$ such that $|U_{mt'+i}| = 1$ for all $T_{i \leq m-k}$ released at t'. By Lemma 6, we have $w_{t'}(j+1) < w_{t'}(j)$ for all $k \leq j < m$, i.e., $w_{t'}(k) \geq m - k$. ■

Finally, this result holds for any tie-break function provided that tasks are not anymore of unitary duration.

Theorem 10. *The competitive ratio of* EFT *(with any tie-break policy) is at least $m - k + 1$ for the problem $P|\text{online}{-}r_i, \mathcal{M}_i\,(interval), |\mathcal{M}_i| = k|F_{\max}$.*

Proof: The proof relies on the same instance as in Theorem 8, with some additional tasks with smaller duration. Original tasks from the instance of Theorem 8 are called *regular* tasks. Our objective is to enforce the following property:

Property 1. *Consider a machine M_i at time t, right before the allocation of regular tasks released at t. During time interval $[t-1; t]$, M_i has $h \geq 0$ regular tasks waiting for execution (excluding the eventual one that is already started). These tasks will be completed at time $t + h + i\delta$.*

The value of δ will be set later to a very small value so that (i) m delays of δ is smaller than the duration of a regular task (1 time unit) and (ii) the total volume of small tasks can be considered as negligible in the optimal solution. Once a value of $\delta < 1/m$ is chosen, we set $\varepsilon < \delta/(2m)$. As we will see below, the $i\delta$ delays on each processor allow emulating the original EFT-Min algorithm, which breaks ties among available processors by choosing the one with minimum index.

We now explain how small tasks are added to the original schedule. Consider any integer time t (including $t = 0$). We have two rounds of small tasks submitted at time t, right before the regular tasks. We consider the set of processors that do not process regular tasks during time interval $[t-1, t]$ (all processors in the case of $t = 0$). Let m_{idle} be the number of such processors.

Intuitively, we first submit m_{idle} small dummy tasks at time t that are scheduled by EFT using its tie break policy (which we do not control). All dummy tasks have different durations such that there is no tie anymore among these machines for the second round. In the second round, we submit tasks whose duration is carefully crafted to ensure that each machine finishes its computation at the prescribed time $t + i\delta$.

First round. We first initialize a counter $c \leftarrow 1$. At time t, while there exists an idle processor M_{i_c} with $i_c \geq 0$, we submit a task T^1_c of duration $c\varepsilon$ with an interval covering processor M_{i_c} (i.e. $M_{i_c} \in \mathcal{M}(T^1_c)$, for example interval $[i_c, i_c + k]$ if $i_c + k < m$, and $[m-k, m]$ otherwise). We then increment the counter $c \leftarrow c + 1$.

Second round. When all tasks of the first round are submitted and allocated, we submit new tasks based on the allocation of the tasks of the first round. For each $c = 1, \dots, m_{idle}$, we consider the processor M_i where the task T^1_c of the first round has been allocated. We submit a task $T^2_{c,i}$ of duration $i\delta - c\varepsilon$ with an interval covering M_i (as above).

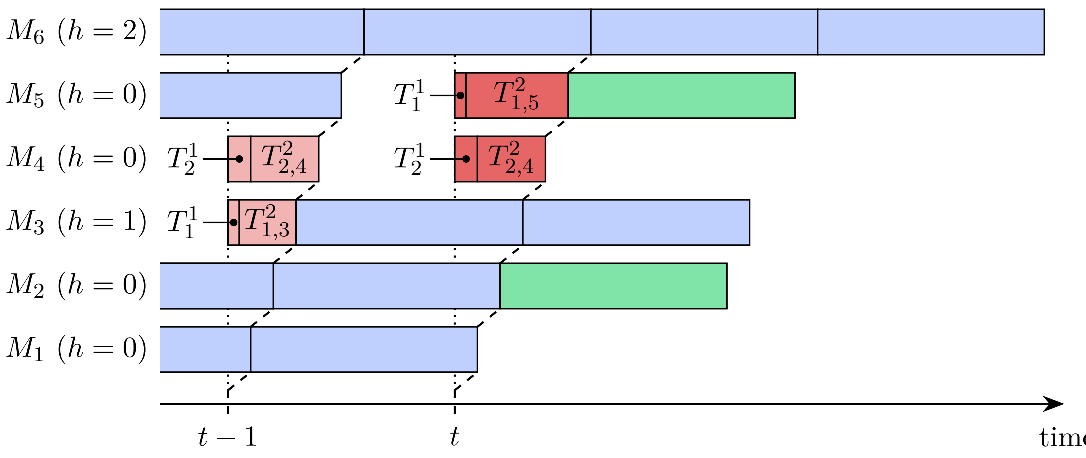


Figure 7: Illustration of the construction of the instance for Theorem 10 when adding small tasks at time t. Regular tasks submitted before t are depicted in blue. Small tasks added to ensure the common delay of $i\delta$ are in red (dark red for step t, light red for step $t-1$), and regular tasks submitted at step t are in green. Only two processors are not processing any regular tasks before time t (M_4 and M_5) and require small tasks to ensure the common delay of $i\delta$.

We now prove that Property 1 is verified at all time, by induction on the time t.

Let us first consider the beginning of the schedule ($t = 0$): small tasks are submitted for all idle processors M_i with $i \geq 0$, before the submission of regular tasks. Each task T_c^1 of the first round must be allocated and started at time $t = 0$ on some idle processor, not necessarily M_{i_c}. However, at the end of this first round, all processors must be processing a small task, as the scheduling algorithm never leaves a processor idle when there is some task to perform on it. Tasks submitted during the first round will complete at times $t + c\varepsilon$, with $c = 1, \ldots, m$. Thus, the latest completion time for the first round is equal to $t + m\varepsilon$.

We now move to the second round. Note that since $\varepsilon < \delta/(2m)$, the duration of a task $T_{c,i}^2$ of the second round is greater than $(i - c/(2m))\delta$ and is positive as $c < m$ and $i \geq 1$. Note also that M_i is the first machine available in the interval of $T_{c,i}^2$: and on all other machines, either the small task of the first round completes later, or it has already been allocated a task of the second round, which lasts at least $i\delta - m\varepsilon > m\epsilon$ and thus will complete later. Hence, task $T_{c,i}^2$ is necessarily allocated to M_i, and completes at time $t + (c\varepsilon) + (i\delta - c\varepsilon) = t + i\delta$. This proves the property for time $t = 0$.

We now prove the property for $t + 1$, assuming it is correct for t. We consider a machine M_i, which has $h \geq 0$ regular tasks waiting for execution during interval $[t-1; t]$ and $r \geq 0$ new regular tasks released at time t are allocated to M_i. We distinguish two cases:

- During interval $[t; t+1]$, M_i starts a regular task either because it has at least one waiting task in interval ($h > 1$) or a new task released at time t is allocated to it ($r > 1$). By induction, the machine M_i will start this task at time $t + i\delta$ and end it at time $t + 1 + i\delta$. Excluding the started task, there remains $h' = h + r - 1 \geq 0$ waiting tasks in interval $[t; t+1]$. All the regular tasks waiting for execution will be completed at time $t + 1 + i\delta + h' = (t+1) + h' + i\delta$ with $h' \geq 0$. Hence, the property is true at time $t + 1$ for M_i.
- During interval $[t; t+1]$, M_i starts no regular task ($h = 0$ and $r = 0$). At time $t + 1$,

all machines are either idle like M_i (when $h = 0$ and $r = 0$) or computing a regular task (allocated before $t+1$). M_i is allocated a small task T_c^1 in the first round at time $t+1$ (it is available by induction hypothesis) and completes at time $t+1+c\varepsilon$. Since there are at most m machines without regular tasks in interval $[t; t+1]$, all small tasks of the first round are completed before or at time $t+m\epsilon < t+\delta$. In the second round, we prove that the task $T_{c,i}^2$ must be allocated on M_i. As seen before, at time $t+1+c\varepsilon$, all idle machines either completes their tasks of the first round later than M_i or are already computing a task of the second round that completes later. Machines M_j with $j \geq 0$ that are computing regular tasks will be available at the soonest at time $t+j\delta$ to start a regular task by induction hypothesis. Thus, each machine M_j will completes at time $t+1+j\delta$, which is much later than when M_i completes its task from the first round. Hence, $T_{c,i}^2$ is allocated to M_i, and completes at time $t+1+i\delta$.

Lemma 7. *With the additional (non regular) tasks, the execution of any FIFO algorithm (with any tie-break policy) follows the original FIFO policy (with tie-break by selecting the processor with smallest index), up to a delay of $i\delta$ for each processor M_i.*

This lemma is proven by noticing that compared to the original setting, processors are not available simultaneously for regular tasks, but with a small delay $i\delta$ of increasing value for increasing processor index. Hence, whenever a regular task can be processed on several processors in the original setting, now the FIFO policy forces the processor with smaller index to execute it, as it was done in the original FIFO policy.

The instance used in the proof of Theorem 10 requires at most m^3 steps (each made of m tasks) to reach a maximum flow of $m-k+1$ for the EFT-Min policy. The modified instance enforces such a maximum flow for a EFT scheduler with any tie-break policy. The total volume of small tasks added to this instance at each time step is bounded by $\sum_{i=1}^{m} i\delta = m(m+1)\delta/2$. Hence the total volume of small tasks during the whole instance is bounded by $m^5\delta/2$. Choosing $\delta = o(1/m^5)$ makes this total volume negligible is front of the duration of a single regular task. We consider an optimal schedule of the original schedule and allocate the additional small tasks to any processor of their interval. The maximum delay for any processor is of order $o(1)$. Hence the maximum flow of this modified optimal algorithm is $1+o(1)$, which proves the asymptotic competitive ratio of $m-k+1$. ■

7 Experimental Results

In this section, we evaluate the relative impact of structured processing set restrictions on the performance of simple scheduling heuristics when the popularity of requests is not uniform, i.e., when certain tasks restricted to the same processing set appear more frequently than others. We begin by explaining our model of popularity before developing the process we used to evaluate the theoretical maximum load permitted by data item replication. Finally, we perform simulations to provide an experimental perspective to the bounds derived in the previous section. All the related code, data and analysis are available online[1].

[1]TODO

7.1 Model of Popularity

Let us consider a cluster of m machines, where tasks have a unit processing time and are released according to a Poisson process with parameter λ (in other words, λ tasks are released in average at each time unit). $\frac{\lambda}{m}$ measures the average load on the whole cluster; thus, when $\lambda = m$, the cluster is loaded at 100%.

For now, suppose that each task can be processed by only one specific machine, i.e., we have $|\mathcal{M}_i| = 1$ for all task T_i. This corresponds to what happens in key-value stores when data items are not replicated: each task T_i carries a key, which is uniquely associated to a data item in the system, and this data item is held by only one machine of the cluster. Therefore, T_i has no choice but to be sent and processed on this specific machine.

In practice, some data items are requested more frequently than others during the service lifetime; depending on the data partitioning and popularity bias on requested keys, some machines will potentially have to process more tasks than others, leading to a biased distribution on machine popularity. Let E_j be the event in which a task must be processed by machine M_j (because it requests a key held by M_j), which occurs with probability $P(E_j)$. Thus, $\lambda P(E_j)$ is the average number of tasks sent on M_j at each time unit, and measures the load of M_j. Note that because of the non-uniform popularity bias $P(E_j)$, the load of a given machine can be greater than 100% (even if the average cluster load is below 100%). In this case, the machine completely saturates as there is no replication.

Let us consider that the machine popularity follows a Zipf distribution. We have $P(E_j) = \frac{1}{j^s H_{m,s}}$, where $s \geq 0$ is the shape parameter of the distribution and $H_{m,s}$ is the m-th generalized harmonic number of order s. We use s to control the popularity bias: the larger s, the more the popularity heterogeneity increases. In the following, we focus on three specific situations. When $s = 0$, the distribution degenerates to the uniform distribution, i.e., no machine is more popular than another (we call this case the **Uniform case**). When $s > 0$, the Zipf distribution has the particularity to generate a monotonically decreasing load on machines $M_1, \ldots, M_m$. This corresponds to a worst case, as the first machines concentrate most of the workload (**Worst-case**). Finally, we randomly permute $P(E_j)$ to match with more realistic settings (**Shuffled case**). Figure 8 shows an example of load distribution for each case.

7.2 Analysis of Theoretical Maximum Load

We want to find the theoretical maximum cluster load (that is, finding the maximum value of λ such that the load on each machine is below 100%) one can achieve when data items are replicated

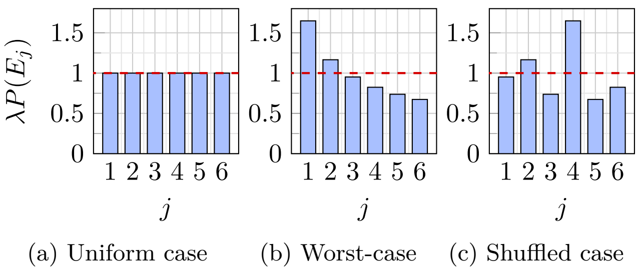
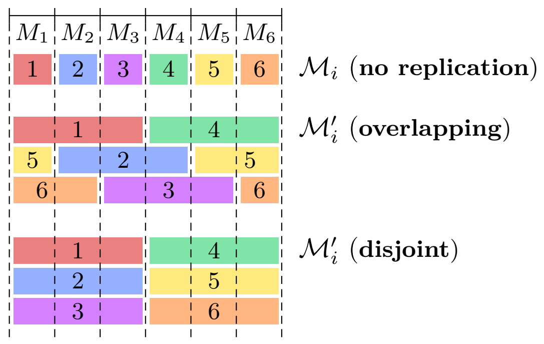


Figure 8: Example of load distribution on a cluster of $m = 6$ machines, with $\lambda = m$, for each case.



M_1
M_2
M_3
M_4
M_5
M_6
1 2 3 4 5 6
$\mathcal{M}_i$ (**no replication**)
1 4
5 2 5
6 3 6
$\mathcal{M}'_i$ (**overlapping**)
1 4
2 5
3 6
$\mathcal{M}'_i$ (**disjoint**)

Figure 9: Example of replication strategies in overlapping and disjoint settings, with $k = 3$. For example, suppose that a task T_i is feasible on M_3 only ($\mathcal{M}_i = \{M_3\}$). Then, in overlapping setting (resp. disjoint setting), the new processing set restriction of T_i is $\mathcal{M}'_i = \{M_3, M_4, M_5\}$ (resp. $\mathcal{M}'_i = \{M_1, M_2, M_3\}$).

across the cluster. Up to now, as we did not consider replication yet, we supposed that each task could only be processed by a single machine (the one holding its requested key). In this case, we clearly have $\lambda \leq 1/\max_j P(E_j)$.

Let us give more choices to each task by adding more machines to the processing sets $\mathcal{M}_i$. This can be seen as replicating data items. Our goal is to study how extending $\mathcal{M}_i$ under a popularity bias affects performance metrics such as the maximum flow time or the maximum cluster load, and how structures in processing sets impact them.

For each task T_i, we build a new set $\mathcal{M}'_i$ from $\mathcal{M}_i$ by defining a replication strategy; in other words, starting from a set with a single machine $\mathcal{M}_i = \{M_u\}$, we replicate the keys held by M_u on all machines of $\mathcal{M}'_i$. We focus on strategies that consist in adding $k-1$ machines (with $1 \leq k \leq m$) to the set, such that $\mathcal{M}'_i$ constitutes an interval of size k, i.e., $\mathcal{M}'_i = I_k(u)$. We describe two manners to build $I_k(u)$ from M_u. Figure 9 illustrates these constructions.

Overlapping intervals. There are m distinct overlapping replication intervals of size k, arranged as a ring:

$$\begin{aligned} I_k(u) = \{M_j \text{ s.t. } j = (j' - 1) \bmod m + 1 \\ \text{for all } u \leq j' \leq u + k - 1\}. \end{aligned}$$

This constitutes the basic replication strategy of key-value stores: machines are arranged as a ring, and data items held by a given machine are replicated on the successors of this machine [5,6]. We have seen in Theorems 8, 9 and 10 that EFT does not always provide a good competitive ratio when minimizing maximum flow time with this structure.

Disjoint intervals. We divide the cluster into $\lceil \frac{m}{k} \rceil$ disjoint replication intervals of size k:

$$I_k(u) = \{M_j \text{ s.t. } u' + 1 \leq j \leq \min(m, u' + k)\},$$

where $u' = k \left\lfloor \frac{u-1}{k} \right\rfloor$. This corresponds to the situation seen in Theorem 6 and related corollaries. EFT guarantees a good competitive ratio when minimizing maximum flow time with this structure.

After replication, all tasks that could only run on a given machine M_j can now be processed by any machine of $I_k(j)$. To quantify the gain on maximum cluster load permitted by a given replication strategy, we solve the following optimization problem modeled as a Linear Program:

$$\begin{aligned}
&\textbf{maximize} && \lambda && \text{(15a)}\\
&\text{subject to} && \forall j,\ \sum_i a_{ij} = \lambda P(E_j), && \text{(15b)}\\
& && \forall i,\ \sum_j a_{ij} \leq 1, && \text{(15c)}\\
& && \forall i,j \text{ s.t. } M_i \notin I_k(j),\ a_{ij} = 0, && \text{(15d)}\\
& && \forall i,j,\ a_{ij} \geq 0, && \text{(15e)}\\
& && \lambda \geq 0. && \text{(15f)}
\end{aligned}$$

a_{ij} denotes the average amount of work (in tasks per time unit) that is eventually processed by machine M_i and that corresponds to tasks originally restricted to machine M_j. We consider the following constraints:

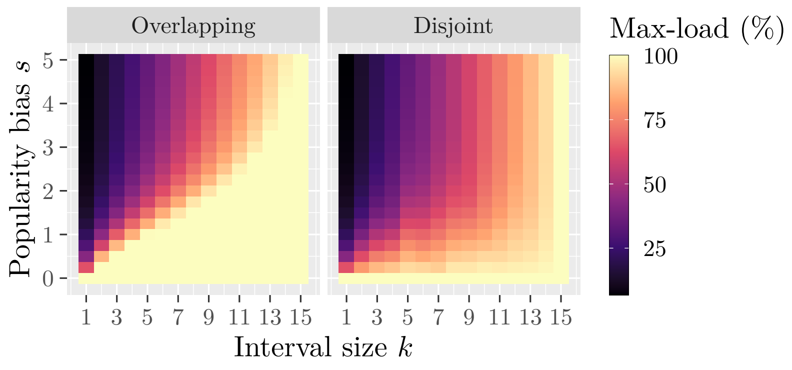
- The total work corresponding to tasks originally restricted on M_j is exactly equal to the initial work of M_j (Equation (15b)).
- The average work eventually processed on M_i does not exceed 1 (Equation (15c)).
- We can transfer work from M_j to M_i if and only if M_i belongs to the interval of size k generated from M_j according to the considered replication strategy, i.e., all tasks that could originally run exclusively on M_j can now also run on M_i (Equation (15d)).

7.3 Experimental Evaluation of Theoretical Maximum Load

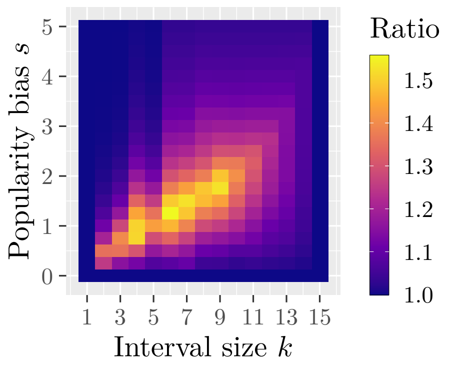
In the following experiments, we set the cluster size m to 15. Figure 10a shows the result of our Linear Program (Equations (15)) as a function of bias s and interval size k, for both previously described replication strategies, in the **Shuffled case** (median over 100 different permutations).

At first glance, it seems that the disjoint strategy is less efficient than the overlapping strategy to cope with high cluster load when non-uniform popularity biases are introduced. For example, for $s = 1$ and $k = 5$, Figure 10a indicates that the cluster can theoretically tolerate a maximum load of 100% when intervals overlap, whereas the disjoint strategy allows reaching a maximum load of 70%.

The overlapping strategy superiority is clearly confirmed by Figure 10b, which shows the gain on the maximum load permitted by overlapping replication intervals over the disjoint strategy. The overlapping strategy allows the cluster to handle loads that are up to 50% higher than the disjoint strategy (e.g., for $s = 1.25$ and $k = 6$), and we can observe a gain up to 35% for common situations in key-value stores, when $0 < s \leq 1.5$ (moderate popularity bias) and $k = 3$ (standard replication factor in most implementations). Note that the popularity bias has obviously no effect when data are fully replicated ($k = m$), and that replication strategies exhibit no difference on the tolerable load when no bias is introduced ($s = 0$).



(a) Maximized load obtained by solving the LP (15)



(b) Ratio between the max-load of each strategy

Figure 10: Maximized load for both overlapping and disjoint strategies, for each $0 \leq s \leq 5$ (by steps of 0.25) and $1 \leq k \leq m$, in the **Shuffled case**. In Figure (a), we show the median value obtained from 100 different permutations of weights $P(E_j)$. In Figure (b), we show the ratio between the median max-loads of both replication strategies.

7.4 Simulations with Popularity Bias

Now we simulate EFT scheduling on 15 machines with a popularity bias, on 10 000 generated unit tasks. Figure 12 illustrates the impact of both replication strategies on maximum flow time in the EFT-Min scheduler and its counterpart EFT-Max (which selects the candidate machine with highest index). We consider the three cases of popularity bias (in **Worst-case** and **Shuffled case**, we set $s = 1$). We repeat the experiment 10 times, and we take the median among max-flow values. We set $k = 3$ to match with a realistic key-value store system.

In the **Uniform case**, no difference is visible between EFT-Min and EFT-Max; however, overlapping replication intervals give better results than the disjoint strategy (e.g., for an average cluster load of 90%, EFT exhibits a max-flow of 5 when intervals overlap, whereas it gives a max-

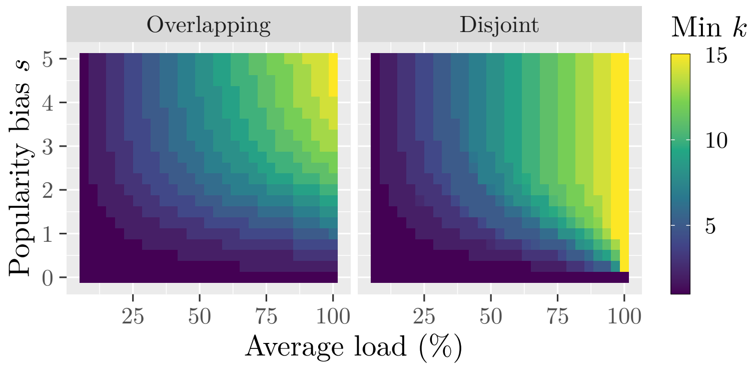


Figure 11: Minimum interval size k needed to handle a given load under a given popularity bias s $(0 \leq s \leq 5)$, for both the overlapping and disjoint strategies.

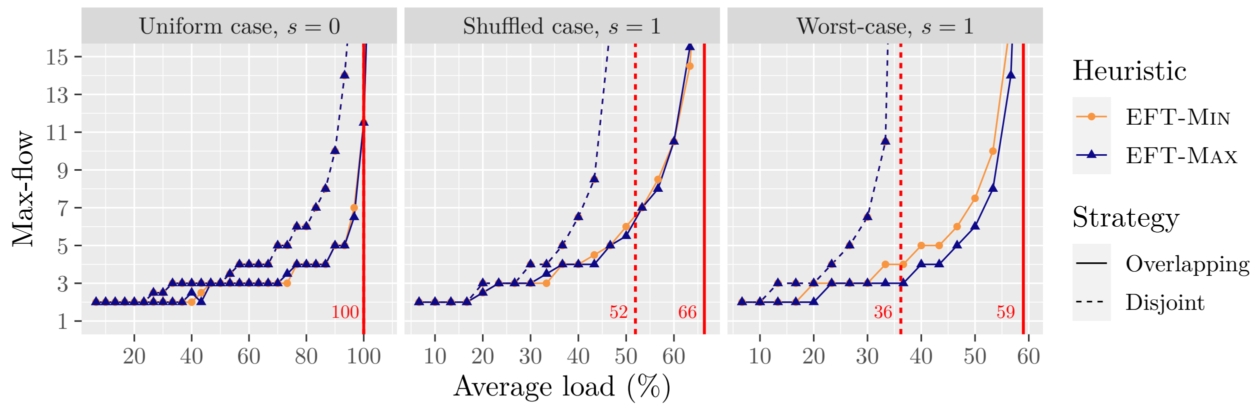


Figure 12: Maximum flow time given by both heuristics EFT-MIN and EFT-MAX as a function of the average load when $k = 3$, for both the overlapping (solid lines) and disjoint (dashed lines) strategies. Each facet corresponds to a distinct case (for the **Worst-case** and **Shuffled case**, we set $s = 1$). Finally, each vertical red line shows the theoretical maximum load value given by the LP (15) in the corresponding case.

flow of 10 with disjoint intervals). When randomly dispatched popularity biases are introduced (**Shuffled case**), we see the relative gain of the overlapping strategy increasing. This is even more obvious when we consider the **Worst-case**. We also see EFT-MAX becoming more efficient than EFT-MIN for the overlapping strategy, which is consistent with the situation in Theorem 8: when breaking a tie, EFT-MIN will select the most popular machine, whereas EFT-MAX does the opposite (as we are in a worst-case, popularity biases are sorted in decreasing order), leading to a smaller max-flow. However, the gain permitted by the scheduling heuristic is rather marginal compared to the gain allowed by a carefully chosen replication structure.

The replication strategy where intervals overlap, commonly used in key-value stores, exhibits better results than the disjoint strategy when popularity biases are introduced, even if the max-flow of EFT in disjoint setting is bounded (Theorem 6). However, there is no efficient worst-case guarantee for the overlapping strategy, as seen in Theorem 8. The question of whether there exists a replication strategy giving both good practical results and theoretical guarantees on EFT scheduling remains open.

8 Conclusion

The high throughput and scalability needs of key-value stores require immediate dispatch algorithms in which requests are allocated to servers as soon as they arrive (such as EFT). In the absence of processing set restrictions, EFT benefits from favorable competitive guarantee for the maximum flow time. However, data cannot be replicated on all servers and we show the competitive ratio of EFT grows at least with the number of machines with several structured processing sets. Among those, interval processing sets are commonly used in key-value stores and our study confirms their ability to efficiently balance load compared to disjoint processing sets.

Future directions include devising a structured processing set, or replication strategy, that would provide efficient performance on average and in the worst case. Moreover, the current bound on the competitive ratio of EFT with interval processing sets could be extended to other immediate dispatch algorithms by relying on an adversary for instance.

References

[1] D. Featherston, "Cassandra: Principles and application," *Department of Computer Science UIUC*, 2010.

[2] A. D. Sicoe, G. L. Miotto, L. Magnoni, S. Kolos, and I. Soloviev, "A persistent back-end for the atlas tdaq online information service (p-beast)," in *Journal of Physics: Conference Series*, vol. 368, no. 1, 2012, p. 012002.

[3] B. Atikoglu, Y. Xu, E. Frachtenberg, S. Jiang, and M. Paleczny, "Workload analysis of a large-scale key-value store," in *SIGMETRICS '12*, 2012, pp. 53–64.

[4] J. Dean and L. A. Barroso, "The tail at scale," *Commun. ACM*, vol. 56, no. 2, pp. 74–80, 2013.

[5] G. DeCandia, D. Hastorun, M. Jampani, G. Kakulapati, A. Lakshman, A. Pilchin, S. Sivasubramanian, P. Vosshall, and W. Vogels, "Dynamo: amazon's highly available key-value store," in *SOSP 2007*, 2007, pp. 205–220.

[6] A. Lakshman and P. Malik, "Cassandra: a decentralized structured storage system," *ACM SIGOPS Oper. Syst. Rev.*, vol. 44, no. 2, pp. 35–40, 2010.

[7] R. Sumbaly, J. Kreps, L. Gao, A. Feinberg, C. Soman, and S. Shah, "Serving large-scale batch computed data with project voldemort," in *FAST 2012*, 2012, p. 18.

[8] D. M. Cavalcante, V. A. E. de Farias, F. R. C. Sousa, M. R. P. Paula, J. C. Machado, and J. N. de Souza, "Popring: A popularity-aware replica placement for distributed key-value store," in *CLOSER 2018*, 2018, pp. 440–447.

[9] A. Makris, K. Tserpes, D. Anagnostopoulos, and J. Altmann, "Load balancing for minimizing the average response time of get operations in distributed key-value stores," in *ICNSC 2017*, 2017, pp. 263–269.

[10] R. L. Graham, E. L. Lawler, J. K. Lenstra, and A. R. Kan, "Optimization and approximation in deterministic sequencing and scheduling: a survey," vol. 5, pp. 287–326, 1979.

[11] M. A. Bender, S. Chakrabarti, and S. Muthukrishnan, "Flow and stretch metrics for scheduling continuous job streams," in *SODA*, vol. 98, 1998, pp. 270–279.

[12] S. Anand, K. Bringmann, T. Friedrich, N. Garg, and A. Kumar, "Minimizing maximum (weighted) flow-time on related and unrelated machines," *Algorithmica*, vol. 77, no. 2, pp. 515–536, 2017.

[13] M. A. Bender, "New algorithms and metrics for scheduling," Ph.D. dissertation, 1998.

[14] M. Mastrolilli, "Notes on max flow time minimization with controllable processing times," *Computing*, vol. 71, no. 4, pp. 375–386, 2003.

[15] ——, “Scheduling to minimize max flow time: Off-line and on-line algorithms,” *Int. J. Found. Comput. Sci.*, vol. 15, no. 2, pp. 385–401, 2004.

[16] E. L. Lawler and J. Labetoulle, “On preemptive scheduling of unrelated parallel processors by linear programming,” *J. ACM*, vol. 25, no. 4, pp. 612–619, 1978.

[17] J. Labetoulle, E. L. Lawler, J. K. Lenstra, and A. R. Kan, “Preemptive scheduling of uniform machines subject to release dates,” in *Progress in combinatorial optimization*, 1984, pp. 245–261.

[18] A. Legrand, A. Su, and F. Vivien, “Minimizing the stretch when scheduling flows of divisible requests,” *J. Sched.*, vol. 11, no. 5, pp. 381–404, 2008.

[19] C. Ambühl and M. Mastrolilli, “On-line scheduling to minimize max flow time: an optimal preemptive algorithm,” *Oper. Res. Lett.*, vol. 33, no. 6, pp. 597–602, 2005.

[20] N. Bansal and B. Cloostermans, “Minimizing maximum flow-time on related machines,” *Theory Comput.*, vol. 12, no. 1, pp. 1–14, 2016.

[21] N. Bansal, “Minimizing flow time on a constant number of machines with preemption,” *Oper. Res. Lett.*, vol. 33, no. 3, pp. 267–273, 2005.

[22] N. Bansal and J. Kulkarni, “Minimizing flow-time on unrelated machines,” in *STOC 2015*, 2015, pp. 851–860.

[23] P. Brucker, B. Jurisch, and A. Krämer, “Complexity of scheduling problems with multi-purpose machines,” *Ann. Oper. Res.*, vol. 70, pp. 57–73, 1997.

[24] J. Y.-T. Leung and C.-L. Li, “Scheduling with processing set restrictions: A survey,” *International Journal of Production Economics*, vol. 116, no. 2, pp. 251–262, 2008.

[25] K. Lee, J. Y. Leung, and M. L. Pinedo, “Makespan minimization in online scheduling with machine eligibility,” *Ann. Oper. Res.*, vol. 204, no. 1, pp. 189–222, 2013.

[26] J. Y.-T. Leung and C.-L. Li, “Scheduling with processing set restrictions: A literature update,” *International Journal of Production Economics*, vol. 175, pp. 1–11, 2016.

[27] B. Simons, “Multiprocessor scheduling of unit-time jobs with arbitrary release times and deadlines,” *SIAM J. Comput.*, vol. 12, no. 2, pp. 294–299, 1983.