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 (Earliest Finish Time) 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. Even though overlapping intervals suffer from a bad worst-case in theory, they enable clusters to reach a maximum load that is up to 50% higher than with disjoint sets.**

***Index Terms*—Flow Time, Lower Bound, Processing Set Restrictions, Replication, Key-Value Stores.**

I. 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 clockwise 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 tend 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 release times (online). For this problem, FIFO (First In First Out) is known to be a good solution: it is $(3-\frac{2}{m})$-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 IV), is a good strategy in some cases, but suffers from inefficient worst case performance with such realistic processing set restrictions (Section V). 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 VI 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 II) and presenting the model (Section III).

II. 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 $(3 - \frac{2}{m})$-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 (Fully Polynomial-Time Approximation Scheme) 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 $(3 - \frac{2}{m})$-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 $2 - \frac{1}{m}$ [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 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 I summarizes existing results on online max-flow minimization. Note that 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. In this table, P, $P|\mathcal{M}_i$, Q and R respectively denote parallel machines, parallel machines with processing set restrictions, related machines, and unrelated machines.

III. Model

Even though our problem originates from key-value stores, we formally formulate it using classical scheduling terms. In particular, we want to schedule a set $\mathbb{T}$ of n tasks $T_1, \ldots, T_n$ on a set $\mathbb{M}$ of 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 $\mathcal{M}_i \subseteq \mathbb{M}$ 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:

$\mathcal{M}_i$(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 $a_i \leq b_i$.

$\mathcal{M}_i$(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 $\mathcal{M}_i \cap \mathcal{M}_j = \emptyset$.

$\mathcal{M}_i$(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 $\mathcal{M}_j \subseteq \mathcal{M}_i$.

$\mathcal{M}_i$(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.

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.

IV. 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],

Table I: Existing results on max-flow optimization.

Env.	Preemption	Algorithm	Type	Approximation/Competitive Ratio	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]

[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 $M_j \in \mathbb{M}$ and for any $1 \leq i \leq n$, let $H_{j,i}$ denote the subset of tasks $T_1, \ldots, T_i$ being assigned to M_j in a schedule Π:

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

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

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

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

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

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
Input: Incoming tasks T_i
Output: Allocated machines μ_i, starting times σ_i

when a new task T_i is released **do**
    $enqueue(i, Q)$

In parallel, do:

when some machines U are idle at time t **do**
    $i \leftarrow dequeue(Q)$
    if $i \neq NIL$ **then**
        $u \leftarrow$ BREAKTIE(U)
        $\mu_i \leftarrow u$
        $\sigma_i \leftarrow t$

Now we show that EFT is equivalent to FIFO for the problem $P|\text{online}-r_i|F_{\max}$. We give the proof in the companion research report [27].

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

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.

Algorithm 2 EFT

Input: Incoming tasks T_i
Output: Allocated machines μ_i, starting times σ_i

1: **when** a new task T_i is released **do**
2: Get U_i according to completion times of machines $\mathbb{M}$ (Equation (1))
3: $u \leftarrow \text{BREAKTIE}(U_i)$
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

V. 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 II gives a summary of the results presented here.

We first study the *inclusive* structure of processing sets. We show in Theorem 1 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 1. *The competitive ratio of any immediate dispatch algorithm is at least* $\lfloor \log_2(m) + 1 \rfloor$ *for the problem* $P|\text{online}{-}r_i, p_i = 1, \mathcal{M}_i$*(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 $T_i \in \mathcal{T}^{(\ell)}$. A final unit task is released at time $r_i = \log_2(m)\varepsilon$.

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 at time $(\log_2(m) - 1)\varepsilon$. This leaves one machine with at least $\log_2(m)$ tasks, where we finally allocate the 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. The proof is an adaptation of the previous one and is available in the companion research report [27].

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

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

Table II: 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.

Processing set structure	Online	Immediate Dispatch	EFT
inclusive		$\geq \lfloor \log_2(m) + 1 \rfloor$ (Th. 1)	
$\lvert\mathcal{M}_i\rvert = k$		$\geq \lfloor \log_k(m) \rfloor$ (Th. 2)	
nested	$\geq \frac{1}{3}\lfloor \log_2(m) + 2 \rfloor$ (Th. 3)		
disjoint, $\lvert\mathcal{M}_i\rvert = k$			$3 - \frac{2}{k}$ (Cor. 1)
interval, $\lvert\mathcal{M}_i\rvert = k$	≥ 2 (Th. 5)		$\geq m - k + 1$ (Th. 6, 7, 8)

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.

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., $|\mathcal{G}_{0,k}| \geq ks_k$.

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 (see detailed proof in [27]).

Theorem 4. *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 $P|\text{online}{-}r_i, \mathcal{M}_i\text{(disjoint)}|F_{\max}$.*

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 $\mathcal{T}_u = \{T_i \in \mathbb{T} \text{ s.t. } \mathcal{M}_i = \mathcal{M}_u\}$. 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 $F^{\mathcal{N}}_{\max}(\mathcal{I}_u) \leq f(|\mathcal{M}_u|)F^{OPT}_{\max}(\mathcal{I}_u)$, where OPT is an optimal offline strategy. As $\mathcal{I}_u$ is a subproblem of $\mathcal{I}$, we also have

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

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, $F^{\mathcal{N}}_{\max}(\mathcal{I}_u) \leq f(|\mathcal{M}_u|)F^{OPT'}_{\max}(\mathcal{I})$, and

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

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

The proof, detailed in [27], consists in a simple adversary argument: we first submit a task with processing set $\{M_2, M_3\}$. When the scheduler has started this task on some machine M_u, we submit two tasks whose processing set is an interval of size 2 containing M_u: either $\{M_1, M_2\}$ or $\{M_3, M_4\}$.

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 IV. 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] (we detail this proof in [27]). It turns out that this is not the case when adding processing interval restrictions. We prove in Theorems 6, 7 and 8 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 and its competitive ratio is bounded in Theorem 6.

Theorem 6. *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 give here a summary of the long and technical proof of this result, available in [27].

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., $\mathcal{M}_i = \{M_\lambda, \ldots, M_{\lambda+k-1}\}$. Let us build the following adversary instance (we illustrate an EFT-MIN schedule of this instance in Figure 1). At each time t, m tasks are submitted:

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

This adversary relies on the key observation that EFT-MIN favors the machine with smallest index when several machines are available at the same time. On the proposed instance, EFT-MIN will rarely use machines with high indices, whereas machine m is able to process the first task only in each round. The proof consists in showing that machines with smaller indices will be overloaded, and their delay propagates up to the desired flow time, while an optimal schedule would allocate each task on the very last machine of its processing set, reaching a flow of 1.

A key notion in this proof is the concept of profile w, defined as follows: $w_t(j) = \max(0, \mathcal{C}_{j,mt} - t)$ is the work allocated on machine M_j and waiting to be processed, just before the adversary releases the m next tasks at time t. We show that EFT-MIN converges to a stable schedule profile w_τ such that for all j,

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

The proof is divided in two steps, summarized below:

Step 1. We prove that while we have not reached or exceeded the stable profile w_τ, the total delay is strictly increasing with time (i.e., we are getting closer to w_τ).

Step 2. We prove that, at each round, either we have not yet reached the stable profile, or some machine has a delay larger than $m - k$.

These two intermediate results allow us to conclude that some machine eventually reaches a max-flow of $m - k + 1$: from **Step 2**, we know that at each round, either we have already exceeded the stable profile w_τ, or we have not yet reached it. In this case, **Step 1** tells us that we will get closer to the stable profile until we reach or exceed it. In both cases, we conclude that some machine has a max-flow of $m - k + 1$.

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. The only condition for Theorem 7 to hold is that among a set of candidate machines, the random tie-break function chooses

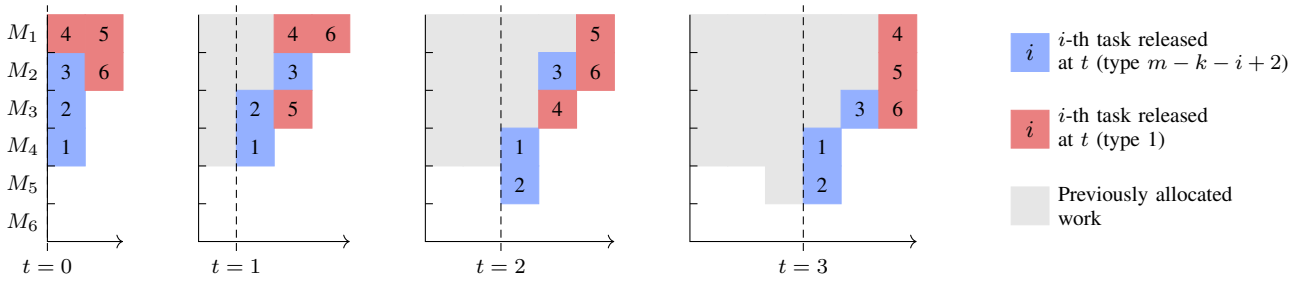


Figure 1: An EFT-MIN schedule of the adversary from time $t = 0$ to $t = 3$, for $m = 6$ and $k = 3$. Colored tasks are released in-order at each time t.

each machine with positive probability, i.e., no machine is systematically discarded when it is a possible candidate. The proof is availabe in [27].

Theorem 7. *The competitive ratio of* EFT-RAND *is at least* $m - k + 1$ *(almost surely) for the problem* $P|\text{online}{-}r_i, p_i = 1, \mathcal{M}_i\text{(interval)}, |\mathcal{M}_i| = k|F_{\max}$*, where* $1 < k < m$*. In other words, there exists an instance for which we have*

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

Finally, this result holds for any tie-break function provided that tasks are not anymore of unitary duration (see proof in [27]).

Theorem 8. *The competitive ratio of* EFT *(with any tie-break policy) is at least* $m - k + 1$ *for the problem* $P|\text{online}{-}r_i, \mathcal{M}_i\text{(interval)}, |\mathcal{M}_i| = k|F_{\max}$.

VI. EXPERIMENTAL RESULTS

In this section, we evaluate the relative impact of structured processing set restrictions on the performance of simple scheduling heuristics. We focus on both interval processing sets, because they are used in actual systems [5]–[7], and disjoint processing sets, because it is the restrictions for which we have the best, and only, approximation ratio (Theorem 4). Moreover, the performance of actual systems are affected by the popularity of requests, which is not uniform, i.e., 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 [27].

A. 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, which has been advocated to model popularity distributions [28]. 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**). As realistic bias strongly depends on the dataset and system usage, each permutation is chosen uniformly as we assume no prior knowledge. Figure 2 shows an example of load distribution for each case.

B. 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 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 3 illustrates these constructions.

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

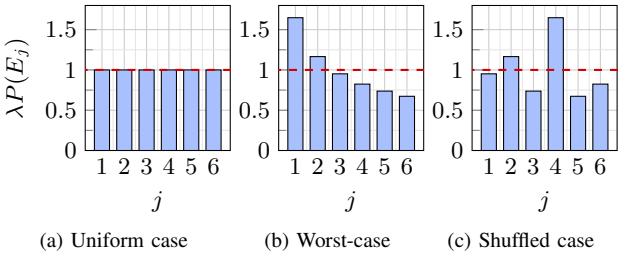
$$I_k(u) = \{M_j \text{ s.t. } j = (j'-1) \bmod m + 1 \\ \text{for all } u \leq j' \leq u+k-1\}.$$

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 6, 7 and 8 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 4 and related corollaries. EFT guarantees a good competitive ratio when minimizing maximum flow time with this structure.



(a) Uniform case (b) Worst-case (c) Shuffled case

Figure 2: Example of load distribution on a cluster of $m = 6$ machines, with $\lambda = m$, for each case.

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:

$$\begin{aligned}
&\textbf{maximize} && \lambda && \text{(3a)}\\
&\text{subject to} && \forall j, \sum_i a_{ij} = \lambda P(E_j), && \text{(3b)}\\
&&& \forall i, \sum_j a_{ij} \leq 1, && \text{(3c)}\\
&&& \forall i,j \text{ s.t. } M_i \notin I_k(j),\ a_{ij} = 0, && \text{(3d)}\\
&&& \forall i,j,\ a_{ij} \geq 0, && \text{(3e)}\\
&&& \lambda \geq 0. && \text{(3f)}
\end{aligned}$$

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 (3b)).
- The average work eventually processed on M_i does not exceed 1 (Equation (3c)).
- 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 (3d)).

C. Experimental Evaluation of Theoretical Maximum Load

In the following experiments, we set the cluster size m to 15, which is a common setup when conducting experiments with scheduling in real key-value store systems [29], [30]. Figure 4a shows the result of our Linear Program (Equations (3)) as a function of bias s and interval size k, for both previously

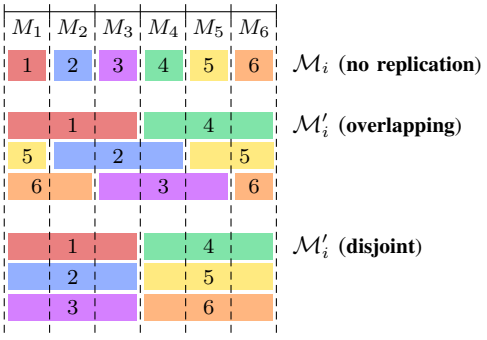


Figure 3: Example of replication strategies in overlapping and disjoint settings, with $k = 3$. For example, suppose that a task T_i is feasible on M_3 only ($\mathcal{M}_i = \{M_3\}$). Then, in overlapping setting (resp. disjoint setting), the new processing set restriction of T_i is $\mathcal{M}'_i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

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 4a 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 4b, 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 ($s = 0$).

D. Simulations with Popularity Bias

Now we simulate EFT scheduling on $m = 15$ machines with a popularity bias, on 10 000 generated unit tasks, which is sufficient to reach a steady state. Figure 5 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-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 6: 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 4). However, there is no efficient worst-case guarantee for the overlapping strategy, as seen in Theorem 6. The question of whether there exists a replication strategy giving both good practical results and theoretical guarantees on EFT scheduling remains open.

VII. 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, storage constraints usually prevent replicating all data on all servers; this is modeled by introducing restrictions on the task processing sets. We provide bounds on the competitive ratio for several structured processing sets. In particular, we show that the competitive ratio of EFT goes from $(3 - \frac{2}{m})$ to $m - k + 1$ for interval processing sets, which are the most commonly used in key-value stores. However, despite the poor theoretical guarantee for EFT, we show experimentally that interval processing sets allow a load up to 50% larger than 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.

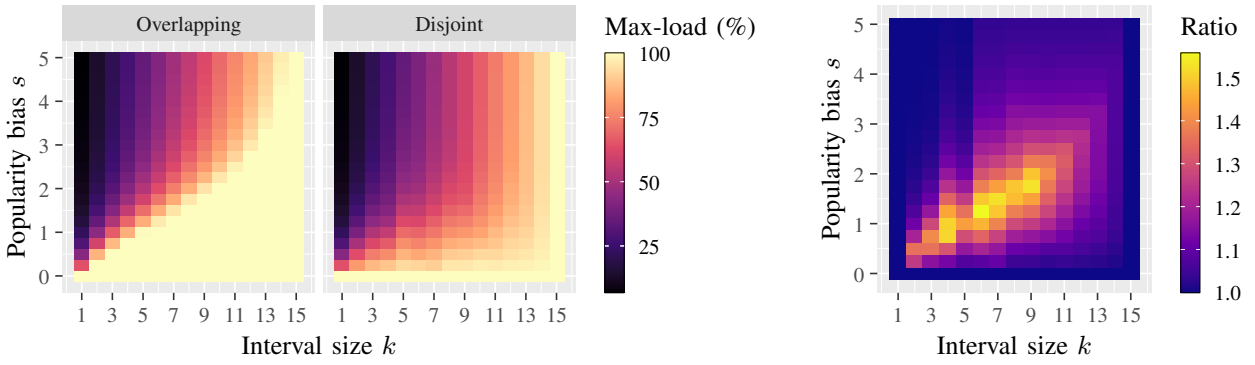
[7] R. Sumbaly, J. Kreps, L. Gao, A. Feinberg, C. Soman, and S. Shah, "Serving large-scale batch computed data with project voldemort," in *FAST 2012*, 2012, p. 18.

[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.



(a) Maximized load obtained by solving the LP (3)

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

Figure 4: 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.

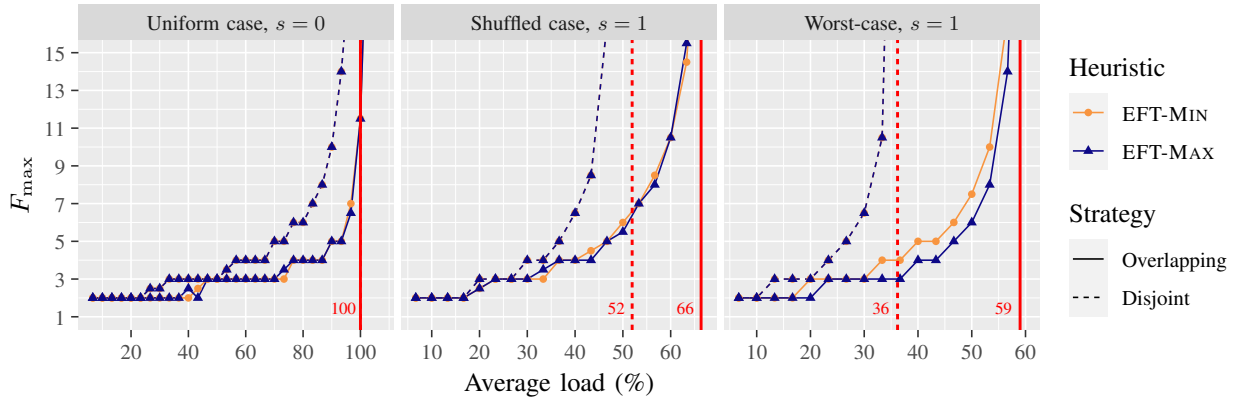


Figure 5: Maximum flow time given by both heuristics EFT-MIN and EFT-MAX as a function of the average load when $k = 3$, for both the overlapping (solid lines) and disjoint (dashed lines) strategies. Each facet corresponds to a distinct case (for the **Worst-case** and **Shuffled case**, we set $s = 1$). Finally, each vertical red line shows the theoretical maximum load value given by the LP (3) in the corresponding case.

[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] https://github.com/anon-ipdps2021/ipdps2021.

[28] D. G. Feitelson, *Workload modeling for computer systems performance evaluation*. Cambridge University Press, 2015.

[29] V. Jaiman, S. B. Mokhtar, V. Quéma, L. Y. Chen, and E. Riviere,

"Héron: Taming tail latencies in key-value stores under heterogeneous workloads," in *SRDS 2018*, 2018, pp. 191–200.

[30] P. L. Suresh, M. Canini, S. Schmid, and A. Feldmann, "C3: cutting tail latency in cloud data stores via adaptive replica selection," in *NSDI 15*, 2015, pp. 513–527.