Bandit Learning for Online Scheduling with Immediate Decision

Zilong Wang[1] **Yuhao Zhang**[1] **Zhewei Wei**[2] **Shuai Li**[*1,3]

[1]Shanghai Jiao Tong University, Shanghai, China
[2]Renmin University of China, Beijing, China,
[3]Shanghai Innovation Institute, Shanghai, China,

Abstract

Motivated by latency-critical streaming domains such as IoT data processing and cloud resource allocation, we investigate the problem of Online Scheduling with Immediate Decision, where a scheduler must instantly decide whether to accept an incoming task without buffering. We consider a system with M identical machines handling tasks with fixed processing lengths and stochastic, initially unknown rewards. A distinct feature of our model is preemption with abandonment: while a machine can interrupt a running task to accept a more valuable new arrival, the preempted task is permanently discarded, yielding no reward and no feedback, which is crucial for learning. We first analyze the setting with known rewards, deriving a worst-case competitive ratio lower bound and proposing the near-optimal Maximum Remaining Density First (MRDF) algorithm. For the challenging case of unknown rewards with censored feedback, we design an efficient bandit algorithm, Scheduling Upper Confidence Bound (S-UCB), which balances exploration and exploitation to achieve an $O(\log T)$ regret over time horizon T. Experimental results demonstrate the efficacy of the proposed algorithms against standard baselines.

1 INTRODUCTION

Online scheduling remains a cornerstone of operations research and computer science, yet the vast majority of traditional models operate under the assumption that tasks can be buffered, queued, or revisited [Graham, 1966, Pruhs et al., 2004, Wang et al., 2024]. While sufficient for static batch processing, this assumption collapses in modern latency-critical domains, such as IoT data streaming, edge computing, and high-frequency cloud resource allocation. These scenario enforce a strict "now or never" constraint, where the scheduler faces an irrevocable choice upon every task arrival: process it immediately, or lose it forever.

Consider an IoT monitoring system where edge devices process high-velocity sensor streams [Hou et al., 2009]. Due to limited memory and strict latency requirements, data packets not processed instantly are overwritten. Similarly, in serverless cloud computing or financial high-frequency trading [Zhou, 2024, Wang et al., 2024], the economic value of a transaction is realized only upon completion; failing to prioritize high-value streams results in significant opportunity costs. We formalize this paradigm as *Online Scheduling with Immediate Decision*. In this setting, M identical machines must instantly accept or reject arriving tasks. A defining and unforgiving feature of our model is *preemption with abandonment*: a machine can interrupt a running task to switch to a potentially more valuable arrival, but the interrupted task is permanently discarded, yielding zero reward.

The problem is further compounded when task rewards are stochastic and initially unknown. For example, the computational value of a cloud task may depend on its type (e.g., image inference vs. log parsing), the distribution of which is revealed only after successful execution. This necessitates a Multi-Armed Bandit (MAB) approach to balance the *Exploration-Exploitation* dilemma. However, unlike standard MAB settings [Lattimore and Szepesvári, 2020], our environment suffers from censored feedback: if the scheduler attempts to "explore" a task but is forced to preempt it later for a more urgent arrival, the system incurs the cost of time without observing any reward signal. This partial observability fundamentally breaks standard regret analysis techniques, creating a significant theoretical gap in learning-augmented scheduling.

In this paper, we adopt a systematic approach to address these challenges, bridging the gap between worst-case competitive analysis and online learning:

- **Theoretical Limits of Scheduling:** We first analyze

[*]Corresponding author. Email: shuaili8@sjtu.edu.cn

Proceedings of the 42nd Conference on Uncertainty in Artificial Intelligence (UAI 2026), PMLR 337:7168–7199.

the known-reward setting to establish a fundamental baseline. We derive a worst-case competitive ratio lower bound of $\tilde{O}(1/(ML_{\max}^{1/M}))$, where $L_{\max}$ is the maximum task length, and prove via a novel adversarial instance that no deterministic algorithm can significantly outperform this threshold.

- **Near-Optimal Scheduling Algorithm:** We propose the "Maximum Remaining Density First" (MRDF) algorithm. We prove that MRDF achieves a competitive ratio matching the theoretical lower bound only up to a logarithmic factor, establishing it as a near-optimal strategy for the known-reward case.
- **Bandit Learning for Online Scheduling:** For the general unknown-reward setting, we design the "Scheduling Upper Confidence Bound" (S-UCB) algorithm. By explicitly coupling UCB indices with the monotone density property of MRDF, S-UCB effectively handles censored feedback, achieving an $O(\log T/\Delta)$ regret. This result demonstrates that efficient learning is possible even when exploration is costly and feedback is interruptible.

2 RELATED WORK

Classic Online Scheduling vs. Immediate Decision. Classic online scheduling has been extensively studied under various metrics like makespan and flow time [Graham, 1966, Pruhs et al., 2004, Leonardi and Raz, 1997, Leonardi, 2003, Fiat and Woeginger, 1999]. However, the vast majority of these works assume tasks can wait in a queue and/or that preempted tasks can be resumed later [Pruhs et al., 2004, Chen et al., 1994, Epstein and Favrholdt, 2005]. This assumption fails in streaming applications where data validity expires instantly. While "immediate decision-making" has been explored in online resource allocation (e.g., online knapsack/packing problems), these works predominantly focus on non-preemptable jobs, where a decision is final and resources are locked until completion [Gallego et al., 2015, Dickerson et al., 2019, Asadpour et al., 2020, Ekbatani et al., 2025]. Our setting is significantly more challenging because preemption requires continuous re-evaluation of decisions at every time step. Reusable-resource prophet models also study immediate accept/reject decisions under stochastic assumptions [Sumita et al., 2022]. These guarantees are not directly comparable to ours: their benchmark is distributional and non-preemptive, whereas our known-reward lower bound is adversarial and shows that no ratio independent of $L_{\max}$ is possible under our immediate-decision benchmark.

Preemptive Scheduling with Abandonment. A few studies have addressed online scheduling with immediate decisions and preemption. Canetti and Irani [1995] explored randomized policies for single-machine scheduling, and Lucier et al. [2013] focused on deadline-sensitive jobs where tasks must finish by a specific time. Crucially, in their settings, preempted jobs might be re-assigned before deadlines and/or the objective emphasizes meeting deadlines rather than maximizing total accumulated reward from fixed-length tasks. Our work targets the general multi-machine setting with irrevocable abandonment, filling a gap in the literature regarding reward maximization under these strict constraints.

Bandits in Resource Allocation. The integration of Multi-Armed Bandits (MAB) into scheduling is an active research area [Lattimore and Szepesvári, 2020]. Recent works have utilized UCB-style optimism for task scheduling or for controlling renewal processes [Gao et al., 2021, Cayci et al., 2019]. Others, like Zhou [2024] and Wang et al. [2024], have studied learning with resource constraints. However, these approaches typically assume a queueing model or guaranteed feedback upon selection. They do not address the immediate decision constraint combined with censored feedback caused by preemption—a key characteristic of the streaming data processing scenarios we investigate. Our S-UCB algorithm is specifically designed to handle the loss of feedback inherent to this "now or never" preemptive environment.

3 SETTING

Online Scheduling with Immediate Decision. We consider a slotted online scheduling problem with horizon T, M identical machines, and K tasks. Each task k has an arrival time b_k, a required processing length $\ell_k \le L_{\max}$, and a reward r_k. A machine can process at most one task per slot. For machine i, let its state just before decisions at time t be either idle or $(a_i(t), q_i(t))$, where $a_i(t)$ is the running task and $q_i(t)$ is its remaining processing time. A running task completes when its remaining time reaches zero after receiving ℓ_k units of uninterrupted processing; if task k completes, the scheduler receives reward r_k and sets $f_k = 1$.

At time t, after processing any completions, the scheduler observes the arrival set $\mathcal{L}_t = \{k : b_k = t\}$ and each arriving task's length. For each machine, it may continue the current task, assign a new arrival to an idle machine, preempt the current task and start a new arrival, or reject arrivals. Each arriving task can be assigned at most once, unassigned arrivals are immediately rejected, and a preempted task is irrevocably abandoned with $f_k = 0$. There is no buffer and no resume-after-preemption. The objective is to maximize the total reward of completed tasks, $\sum_{k=1}^{K} f_k r_k$.

For the known-reward setting, an instance is $I = (M, T, \{(b_k, \ell_k, r_k)\}_{k=1}^{K})$. Let $R_{\mathrm{ALG}}(I)$ denote the total reward obtained by an online algorithm and $R_{\mathrm{OPT}}(I)$ denote the clairvoyant offline optimum on the same realized instance. Since the online learner does not know future ar-

rivals, we evaluate algorithms by competitive ratio [Pruhs et al., 2004]:

$$CR = \inf_{I \in \mathcal{I}} \frac{R_{\mathrm{ALG}}(I)}{R_{\mathrm{OPT}}(I)}.$$

Stochastic Reward Model. In most real-world online scheduling cases, the reward of a task is usually unknown and stochastic [Gao et al., 2021]. We assume there are N task types. The reward of each task with type $n \in [N]$ follows a fixed and unknown 1-subgaussian distribution $\mathcal{D}_n$ with mean μ_n. When task k arrives, the learner observes its type $n(k)$ and length ℓ_k, but not its realized reward. The reward $r_k \sim \mathcal{D}_{n(k)}$ is observed only if k completes; rejected and preempted tasks produce no reward and no feedback. A stochastic instance is $I = (M, T, \{(b_k, \ell_k, n(k))\}_{k=1}^{K}, \{\mathcal{D}_n\}_{n=1}^{N})$, and expectations in Section 5 are taken over reward randomness and possible algorithmic randomness.

4 ONLINE SCHEDULING WITH KNOWN REWARDS

In this section, we provide analysis for the setting of M machines and known rewards. Section 4.1 provides worst-case competitive ratio of order $\tilde{O}(1/(ML_{\max}^{1/M}))$. Then Section 4.2 proposes a scheduling algorithm achieves $\tilde{\Omega}(1/(ML_{\max}^{1/M}))$ competitive ratio.

4.1 WORST-CASE COMPETITIVE RATIO ANALYSIS

In this subsection, we derive a worst-case competitive ratio of order $\tilde{O}(1/(ML_{\max}^{1/M}))$ for M machines scheduling problem, shown in the following theorem.

Theorem 4.1. *There exists an instance containing a set of tasks such that no deterministic algorithm can achieve the competitive ratio greater than* $2\log(L_{\max})/(ML_{\max}^{1/M})$.

The intuitive idea to derive such a worst-case competitive ratio bound is that we construct competitive task pairs for each machine to select at each time t. Each competitive pair comprises a high-reward long task accompanied by multiple low-reward short tasks whose aggregate reward exceeds that of the long task. This construction ensures that regardless of the machine's selection, we can always generate an alternative task selection with higher total rewards.

In the multi-machine setting, the construction of competitive tasks becomes more challenging because we must prevent different machines from selecting the same competitive pair. The lower-bound instance uses a geometric ladder of lengths $1, L_{\max}^{1/M}, \ldots, L_{\max}$. As illustrated in Figure 1, at each relevant time step, $2M$ tasks arrive. Once the M machines select M of them, the remaining M tasks can be paired one-to-one with the selected tasks so that each pair contains two different length scales. The factor $L_{\max}^{1/M}$ comes from this geometric bucketing: within one machine's length class, the largest-to-smallest length ratio is $L_{\max}^{1/M}$, and a density-based online rule can lose a factor proportional to this ratio. The additional factor M comes from comparing statically bucketed machines with the global M-machine offline optimum. The lower-bound construction uses the same ladder, showing that this dependence is not merely an artifact of the analysis. The complete proof is deferred in Appendix A.

4.2 MAXIMUM REMAINING DENSITY FIRST ALGORITHM

In this subsection, we provide the algorithm for the scheduling problem with M machines, attaining a competitive ratio at least $\tilde{\Omega}(1/(ML_{\max}^{1/M}))$ for any instance.

Intuitively, the algorithm assigns machines with different responsibilities. Specifically, the i-th machine ($1 \le i \le M$) only processes the task with length between $L_{\max}^{\frac{i-1}{M}}$ and $L_{\max}^{\frac{i}{M}}$. This design guarantees the length ratio between tasks for each machine is upper bounded by $L_{\max}^{1/M}$.

Algorithm 1 Maximum Remaining Density First (for i-th machine)

Input: $L_{\max}$, M.

Initialize running task $a_i = \emptyset$ and remaining time $q_i = 0$.
for $t = 1, 2, \cdots$ **do**
  if $a_i \neq \emptyset$ and $q_i = 0$ **then**
    Receive reward r_{a_i} and set $a_i = \emptyset$.
  end if
  Receive $\mathcal{L}_t^i = \{k : b_k = t,\ L_{\max}^{\frac{i-1}{M}} < \ell_k \le L_{\max}^{\frac{i}{M}}\}$ and observe (ℓ_k, r_k).
  Set candidate set $\mathcal{C}_t = \mathcal{L}_t^i$.
  if $a_i \neq \emptyset$ and $q_i > L_{\max}^{\frac{i-1}{M}}$ **then**
    Add pseudo task $\tilde{a}_i$ to $\mathcal{C}_t$ with length q_i and reward r_{a_i}.
  end if
  Let $c \in \arg\max_{k \in \mathcal{C}_t} r_k/\ell_k$ if $\mathcal{C}_t \neq \emptyset$.
  if $a_i \neq \emptyset$ and $q_i \le L_{\max}^{\frac{i-1}{M}}$ **then**
    if $\mathcal{C}_t = \emptyset$ or $r_c < L_{\max}^{1/M} r_{a_i}$ **then**
      Set $c = \tilde{a}_i$.
    end if
  end if
  if c is a new arriving task **then**
    Preempt a_i if nonempty, set $a_i = c$, and set $q_i = \ell_c$.
  end if
  If $a_i \neq \emptyset$, process a_i for one slot and set $q_i = q_i - 1$.
end for

For each machine, the algorithm selects the task with maxi-

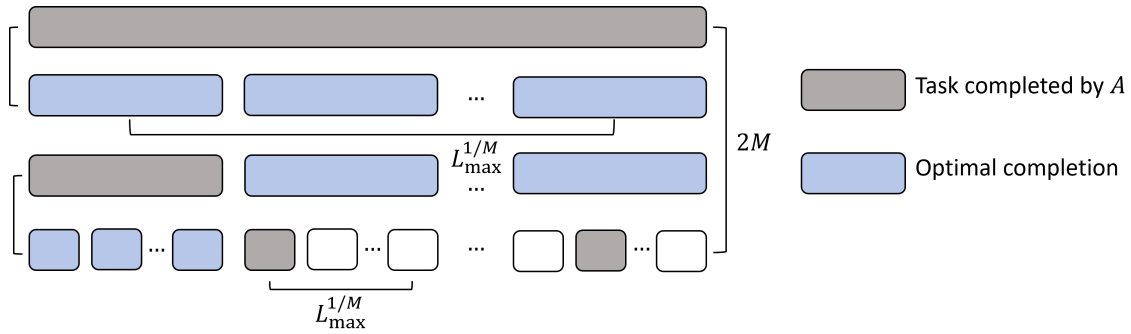

Figure 1: Illustration of the worst-case instance design. At each relevant time, $2M$ tasks arrive and form M competitive pairs across adjacent length scales. If an online algorithm commits to one side of a pair, the adversary can continue the stream so that the offline optimum benefits from the other side, producing the $\tilde{O}(1/(ML_{\max}^{1/M}))$ lower-bound scaling.

mum remaining reward density after reinserting the running task as a pseudo-task. This ensures that the threshold for preempting a running task increases as the task approaches completion. When the remaining length falls below the lower end of the machine's length bucket, the protection rule switches only to a new task whose reward is larger by a factor $L_{\max}^{1/M}$. This design prevents the algorithm from losing more than the within-bucket length-ratio factor, which is the key term in the competitive-ratio analysis.

Theorem 4.2. *For any instance $\mathcal{I}$, the max remaining density first algorithm with M machines can achieve the competitive ratio greater than $1/(3ML_{\max}^{1/M})$.*

Proof sketch: The analysis separates the tasks processed by each machine $i \in [M]$. For every completed task k with $L_{\max}^{\frac{i-1}{M}} < \ell_k \leq L_{\max}^{\frac{i}{M}}$, we compare it with the nearest optimal completed task in the i-th machine. Specifically, those tasks are divided into three sets by their completion times (before k arrives, before k completes, and after k completes). Every optimal task set guarantees that its total rewards do not exceed $L_{\max}^{1/M} r_k$. Since there are at most M optimal tasks at the same time, the corresponding ratio is bounded by $1/(3ML_{\max}^{1/M})$. The complete proof is deferred to Appendix B.

Remark 4.3. Note that constructing a pseudo task for the running task is crucial to obtaining the $\Omega(1/(ML_{\max}^{1/M}))$ competitive ratio. This design ensures the maximum remaining density will increase with time until the machine completes the task. Even though the machine might switch to other tasks, its selecting threshold $\max r/\ell$ will keep increasing, which is crucial to upper bound the summation reward of the optimal schedule by bounding the maximum density at each time.

5 ONLINE SCHEDULING WITH RANDOM AND UNKNOWN REWARDS

In this section, we provide the Scheduling Upper Confidence Bound (S-UCB) algorithm for online scheduling with an unknown and random rewards model introduced in Section 3.

To measure the learning loss of an online algorithm ALG with total reward $R_{\mathrm{ALG}}(T)$, we use an approximate-regret benchmark. This convention is standard in combinatorial multi-armed bandits when the offline oracle is only available up to an approximation factor [Chen et al., 2013, 2016]. In our setting, the source of approximation is different: even with known rewards, Theorem 4.1 shows that adversarial immediate-decision scheduling cannot achieve a constant competitive ratio independent of $L_{\max}$, while Theorem 4.2 shows that MRDF achieves the order-wise attainable factor $\alpha = 1/(3ML_{\max}^{1/M})$. We therefore define the α-approximate regret as the additional loss due to reward learning:

$$\mathrm{Reg}(T) = \mathbb{E}\left[\alpha R_{\mathrm{OPT}}^{\mu}(T) - R_{\mathrm{ALG}}(T)\right],$$

where $R_{\mathrm{OPT}}^{\mu}(T)$ is the offline optimum on the fixed arrival/type/length sequence using true type means, and the expectation is taken over reward randomness and algorithmic randomness.

The key idea in the S-UCB algorithm is that the algorithm utilizes the upper confidence bound (UCB) index for each type of task to balance exploration and exploitation. At each time, the algorithm performs the MRDF algorithm (Algorithm 1) while replacing the reward r_k of each task k by its UCB index $\mathrm{UCB}_{n(k)}$, where $n(k)$ is the type of task k. The UCB index UCB_n of each type $n \in [N]$ is computed as $\mathrm{UCB}_n = \hat{\mu}_n + \sqrt{6\log(T)/T_n}$ after the type has been completed at least once, and as ∞ before that, where $\hat{\mu}_n$ is the empirical reward mean of task n and T_n is the number of completions for tasks with type n.

Intuitively, the UCB index of a type n consists of the exploitation term $\hat{\mu}_n$ and the exploration term $\sqrt{\frac{6\log T}{T_n}}$, which is also noted as the confidence radius. This design corresponds to the fact that the reward mean μ_n of type n is

Algorithm 2 Scheduling Upper Confidence Bound (S-UCB)

Initialize $a_i = \emptyset$, $q_i = 0$, $\hat{\mu}_n = 0$, $T_n = 0$, and $\text{UCB}_n = \infty$ for each type n.
for $t = 1, 2, \cdots$ **do**
  if $a_i \neq \emptyset$ and $q_i = 0$ **then**
    Receive reward r_{a_i}, update $\hat{\mu}_{n(a_i)}$, $T_{n(a_i)}$, and $\text{UCB}_{n(a_i)}$; set $a_i = \emptyset$.
  end if
  Receive $\mathcal{L}_t^i = \{k : b_k = t,\ L_{\max}^{\frac{i-1}{M}} < \ell_k \leq L_{\max}^{\frac{i}{M}}\}$ and observe $(\ell_k, n(k))$.
  Set candidate set $\mathcal{C}_t = \mathcal{L}_t^i$.
  if $a_i \neq \emptyset$ and $q_i > L_{\max}^{\frac{i-1}{M}}$ **then**
    Add pseudo task $\tilde{a}_i$ to $\mathcal{C}_t$ with length q_i and type $n(a_i)$.
  end if
  Let $c \in \arg\max_{k \in \mathcal{C}_t} \text{UCB}_{n(k)}/\ell_k$ if $\mathcal{C}_t \neq \emptyset$.
  if $a_i \neq \emptyset$ and $q_i \leq L_{\max}^{\frac{i-1}{M}}$ **then**
    if $\mathcal{C}_t = \emptyset$ or $\text{UCB}_{n(c)} < L_{\max}^{1/M} \text{UCB}_{n(a_i)}$ **then**
      Set $c = \tilde{a}_i$.
    end if
  end if
  if c is a new arriving task **then**
    Preempt a_i if nonempty, set $a_i = c$, and set $q_i = \ell_c$.
  end if
  If $a_i \neq \emptyset$, process a_i for one slot and set $q_i = q_i - 1$.
end for

upper bounded by UCB_n with high probability $O(1/T)$. Thus the UCB index is an appropriate statistic to balance the exploration and exploitation.

Below, we provide the regret analysis for the S-UCB algorithm. The key technical challenge beyond traditional UCB analysis is action-induced censoring. A selected task may be preempted before completion, consuming processing time but producing neither reward nor sample. A standard pull-count proof therefore fails, because the type count is not updated for selected-but-censored tasks.

S-UCB handles this issue through the same pseudo-task structure that drives MRDF. The running task is reinserted with the same optimistic reward and a decreasing remaining length, so its optimistic remaining density increases as it approaches completion. Hence S-UCB does not need a separate "do not preempt for feedback" rule: it naturally avoids late harmful preemptions unless a new task has sufficiently larger optimistic density. The proof exploits this monotonicity by charging all preempted and censored candidates in an interval to the next completed task. That completed task has UCB density at least as large as those candidates, and its observed reward updates the type count, allowing the confidence-radius argument to bound future mis-orderings.

Theorem 5.1. *When the scheduler follows the S-UCB algorithm (Algorithm 2) with N task types, the learner can obtain the $\frac{1}{3ML_{\max}^{1/M}}$-approximate regret bounded by*

$$\text{Reg}(T) \leq O\left(\frac{NL_{\max}^{1/M} \log T}{\Delta}\right),$$

where $\Delta = \min_{n,n',\ell,\ell',\frac{\mu_n}{\ell} - \frac{\mu_{n'}}{\ell'} > 0} \frac{\mu_n}{\ell} - \frac{\mu_{n'}}{\ell'}$ is the minimum positive *density gap of two tasks.*

Proof. We first introduce some notations. Denote $\mathcal{K}_{\text{ALG}}^i$ as the set of completed tasks for machine i by running the S-UCB. And $\mathcal{K}_{\text{OPT}}^i$ is the set of optimal completed tasks with length $(L_{\max}^{\frac{i-1}{M}}, L_{\max}^{\frac{i}{M}}]$. For each $k \in \mathcal{K}_{\text{ALG}}^i$, we denote $\mathcal{N}_k = \{k' \in \mathcal{K}_{\text{OPT}}^i : b_{k-1} + \ell_{k-1} \leq b_{k'} < b_k + \ell_k\}$ as the set of optimal tasks that is around k, where $k-1$ is the nearest algorithm completed task before k. Denote $\mathcal{G}_k := \forall k' \in \mathcal{N}_k, \mu_{n(k)}/\ell_k \geq \mu_{n(k')}/\ell_{k'}$ as the event that completed task k is truly the task with the highest reward density among all optimal tasks nearby. The regret can be rewritten as

$$\begin{aligned}
\text{Reg}(T) &= \mathbb{E}\left[\alpha \sum_{i=1}^{M} \sum_{k \in \mathcal{K}_{\text{ALG}}^i} \Big(\sum_{k' \in \mathcal{N}_k} r_{k'} - r_k \Big)\right] \\
&= \mathbb{E}\left[\alpha \sum_{i=1}^{M} \sum_{k \in \mathcal{K}_{\text{ALG}}^i} \mathbb{1}\{\mathcal{G}_k\} \Big(\sum_{k' \in \mathcal{N}_k} r_{k'} - r_k \Big)\right] \\
&\quad + \mathbb{E}\left[\alpha \sum_{i=1}^{M} \sum_{k \in \mathcal{K}_{\text{ALG}}^i} \mathbb{1}\{\neg \mathcal{G}_k\} \Big(\sum_{k' \in \mathcal{N}_k} r_{k'} - r_k \Big)\right],
\end{aligned}$$

where $\mathcal{G}_k$ decomposes the regret into two terms: the first is the regret caused by the exploitation (accurate estimation of UCB index), and the second term is the regret caused by exploration (necessary selection to reduce sub-optimal UCB density).

To bound the regret caused by the exploitation, we first state that when $\exists k, \forall k' \neq k, \frac{\text{UCB}_{n(k)}}{\ell_k} > \frac{\text{UCB}_{n(k')}}{\ell_{k'}}$ implies $\frac{\mu_{n(k)}}{\ell_k} > \frac{\mu_{n(k')}}{\ell_{k'}}$, the S-UCB algorithm proceeds the same as the MRDF with known mean reward μ_n. This is because the S-UCB makes the exact same decision as MRDF at each time t. Then under event $\mathcal{G}_k$ for completed task k, S-UCB shares the same competitive ratio property as MRDF since k is actually the task with the highest density. This indicates that the algorithm achieves the competitive ratio larger than $1/(3ML_{\max}^{1/M})$ when UCB index is well estimated, i.e., $\mathcal{G}_k$ holds, and the regret is 0.

Lemma 5.2. *The regret caused by exploiting the UCB index in S-UCB algorithm conditioned on good events is bounded by*

$$\mathbb{E}\left[\sum_{i=1}^{M} \sum_{k \in \mathcal{K}_{ALG}^i} \mathbb{1}\{\mathcal{G}_k\} \Big(\frac{1}{3ML_{\max}^{1/M}} \sum_{k' \in \mathcal{N}_k} r_{k'} - r_k \Big)\right] \leq 0\,.$$

Then we turn to bound the second term, which is the regret caused by exploring sub-optimal UCB density. The following lemma upper bounds the exploration regret. It should be noted that this lemma is technically non-trivial since it integrates the monotone increasing density property of the MRDF algorithm into the analysis of UCB estimation.

Lemma 5.3. *The regret caused by exploring the UCB index in S-UCB algorithm is bounded by*

$$\mathbb{E}\left[\frac{1}{3ML_{\max}^{1/M}}\sum_{i=1}^{M}\sum_{k\in\mathcal{K}_{ALG}^{i}}\mathbb{1}\{\neg\mathcal{G}_k\}\Big(\sum_{k'\in\mathcal{N}_k}r_{k'}-r_k\Big)\right] \le \sum_{n\in[N]}O\left(\frac{L_{\max}^{1/M}\log T}{\Delta_n}\right) \le O\left(\frac{NL_{\max}^{1/M}\log T}{\Delta}\right),$$

where $\Delta_n=\min_{n'\neq n,\ell,\ell',\frac{\mu_{n'}}{\ell}-\frac{\mu_n}{\ell'}>0}\frac{\mu_{n'}}{\ell}-\frac{\mu_n}{\ell'}$ *is the minimum positive density gap* against type n.

Then summing over these two terms, Theorem 5.1 is derived. □

In addition, we also provide the competitive-ratio result for the S-UCB algorithm.

Theorem 5.4. *Given instance I, following the S-UCB algorithm (Algorithm 2), the scheduler can achieve the expected competitive ratio* $\mathbb{E}[CR(I)]=\mathbb{E}\left[R_{ALG(I)}/R_{OPT}(I)\right]$ *with*

$$\mathbb{E}[CR(I)]\ge\frac{1}{3ML_{\max}^{1/M}+\frac{24NL_{\max}^{1/M}\log T}{\mathbb{E}[R_{ALG}(I)]\Delta}},$$

where $R_{ALG}(I)$ is the total completion rewards obtained by S-UCB, and expectation is taken over the randomness of reward distribution.

Moreover, if there are new tasks continuously coming in every round t, the competitive ratio has the bound of $1/\left(3ML_{\max}^{1/M}+NL_{\max}^{1/M}\log T/(T\Delta^2)\right)$. This theorem states that the expected competitive ratio will asymptotically converge to $1/(3ML_{\max}^{1/M})$ when T increases, indicating that S-UCB will perform the same as MRDF when UCB index is well estimated.

Remark 5.5. Comparison with Explore-then-Schedule Strategy: A natural alternative to S-UCB is the Explore-then-Schedule (ETS) strategy (detailed in Appendix E). ETS separates the learning process into two distinct phases. In the exploration phase, it schedules tasks in a round-robin manner to uniformly reduce the confidence radius of all task types. Once the Lower Confidence Bound (LCB) of the optimal task type exceeds the Upper Confidence Bound (UCB) of all sub-optimal ones, it switches to the exploitation phase, running the MRDF algorithm with fixed empirical means.

While ETS guarantees asymptotic convergence, it suffers from high regret during the exploration phase, especially when the number of task types N or the maximum length L_{max} is large. As shown in our regret analysis, the exploration cost scales linearly with NL_{max}. In contrast, S-UCB dynamically balances exploration and exploitation, allowing it to harvest rewards from "good enough" tasks even during the learning process, which is critical in our "now or never" immediate decision setting.

We also derive a gap-independent regret for S-UCB algorithm, showing its $\tilde{O}(\sqrt{T})$ regret convergence when Δ is relatively small.

Theorem 5.6. *Run* S-UCB *(Algorithm 2) for horizon $T\ge 2$, then* S-UCB *satisfies the gap-independent bound*

$$Reg(T)\le 2N+2\sqrt{24}\,NL_{\max}^{1/M}\sqrt{T\log T}=O\left(NL_{\max}^{1/M}\sqrt{T\log T}\right).$$

6 EXPERIMENTS

In this section, we conduct experiments to show the efficiency of our proposed MRDF algorithm (Algorithm 1) and S-UCB algorithm (Algorithm 2).

Synthetic setting. Unless otherwise specified, we use horizon $T=500{,}000$, $M=5$ machines, $N=10$ task types, $L_{\max}=30$, and Gaussian rewards $\mathcal{N}(\mu_n,\sigma^2)$ with $\sigma=1$ and μ_n sampled uniformly from $[0,1]$. Each experiment is repeated 50 times. For additional stress tests requested by reviewers, we also use horizon $T=4000$, $N=30$ task types, Zipf-distributed type arrivals, long-tailed processing lengths, and 5 random seeds.

Baselines. For known rewards, we compare MRDF with:

- Maximum reward (MR), which always processes the task with the highest reward.
- Shortest remaining processing time (SRPT), which always processes the task with the minimum remaining processing time.
- Optimal scheduling strategy (OPT), which needs prior knowledge of reward and processing time information for all tasks before the game. The optimal strategy can be computed by dynamic programming.

For unknown stochastic rewards, we compare S-UCB with Explore-then-Schedule (ETS), Emp-MRDF, Random, and two no-preemption baselines. ETS explores all types uniformly until it is confident and then runs MRDF with empirical means; its details and regret analysis are in Appendix E. Emp-MRDF always runs MRDF with current empirical means. NP-UCB uses UCB density but holds an accepted task until completion, while NP-KM uses true type means under the same no-preemption restriction. We also report

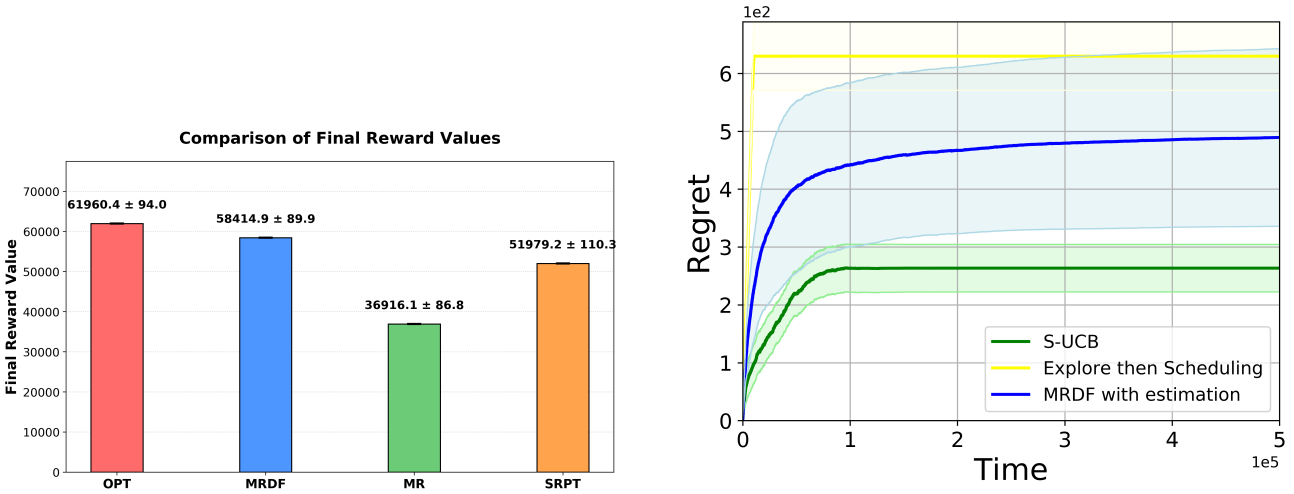

Figure 2: Synthetic experiments. Left: cumulative rewards for OPT, MRDF, MR, and SRPT under known rewards. Right: regret curves for S-UCB, ETS, and Emp-MRDF under unknown stochastic rewards.

MRDF-KM, the MRDF policy run with true type means, as an oracle normalizer rather than a feasible learning baseline.

Figure 2 summarizes the main synthetic results. In the known-reward setting, MRDF is consistently close to the dynamic-programming optimum under uniform processing lengths and outperforms MR and SRPT. In the unknown-reward setting, S-UCB has lower regret and better stability than ETS and Emp-MRDF, confirming that optimistic density is useful even before all type means are accurately estimated.

Table 1 isolates the censored-feedback tradeoff under a deliberately harsher distributional shift than the default synthetic setting. The Zipf arrival law concentrates traffic on a small number of frequent types, while the long-tailed lengths create many opportunities for a scheduler to become trapped by long low-density tasks. Under light load, S-UCB is already within 0.32% of the true-mean MRDF normalizer, whereas NP-UCB loses almost five normalized-reward points because it cannot preempt an accepted task when a shorter high-density arrival appears. The gap widens under load 4.5: S-UCB remains within 1% of MRDF-KM, while Emp-MRDF, ETS, and NP-UCB fall to 0.9465, 0.8661, and 0.9286, respectively. Thus the benefit is not only from optimistic reward estimation; it also comes from using optimism inside a preemptive density rule.

The censoring measurements further distinguish S-UCB from a naive exploration policy. ETS explicitly explores types before committing to MRDF, but this phase creates many selected tasks whose reward observations are destroyed by later preemptions, leading to a censoring rate of 0.233 at load 4.5. S-UCB has a rate of 0.119, close to the MRDF-KM oracle rate of 0.130, even though it has no prior knowledge of type means. This supports the analysis in Lemma 5.3: the pseudo-task mechanism prevents unnecessary late preemptions while still allowing the scheduler to react to high-density arrivals. In other words, S-UCB does not solve censored feedback by avoiding preemption; it makes preemption selective enough that completed tasks still provide the samples needed for learning.

6.1 REAL-WORLD DATASETS

To evaluate the practical efficacy of S-UCB in realistic, high-concurrency streaming environments, we conduct experiments using two large-scale, publicly available production traces from latency-critical and highly saturated domains, where the "now or never" immediate decision constraint naturally arises.

- **Azure Serverless Trace:** [Qiu et al., 2025] This dataset contains two weeks of invocation logs from Azure Functions. Serverless computing mirrors our paradigm, as incoming function invocations must be provisioned instantly; failing to allocate resources immediately results in cold-start penalties or dropped requests. We extract the arrival times and execution durations to represent task arrivals (b_k) and lengths (ℓ_k). The unique `HashApp` identifiers are used to categorize invocations into distinct task types ($n(k)$).
- **Alibaba PAI Cluster Trace:** [Lin et al., 2025] This trace logs the workload of Alibaba's Platform for Artificial Intelligence, predominantly consisting of machine learning training and inference tasks. Inference tasks, in particular, impose strict zero-tolerance latency constraints. We map the workload submissions to our immediate decision model, using the workload signatures to differentiate task types.

Data Mapping and Reward Synthesis. Real-world traces

Table 1: Model-aligned stress tests under Zipf type arrivals, long-tailed processing lengths, $T = 4000$, $N = 30$, and 5 seeds. Normalized reward is the algorithm reward divided by MRDF-KM on the same arrival/type/length sequence. MRDF-KM is an oracle normalizer, not a learning baseline.

Scenario	Algorithm	Normalized Reward (↑)	Censoring Rate (↓)
Load 1.5	**S-UCB (Ours)**	$\mathbf{0.9968 \pm 0.0027}$	N/A
Load 1.5	NP-UCB	0.9512 ± 0.0055	N/A
Load 4.5	MRDF-KM (Oracle)	1.0000	0.130
Load 4.5	**S-UCB (Ours)**	$\mathbf{0.9904 \pm 0.0033}$	**0.119**
Load 4.5	Emp-MRDF	0.9465 ± 0.0050	N/A
Load 4.5	ETS	0.8661 ± 0.0084	0.233
Load 4.5	NP-UCB	0.9286 ± 0.0097	N/A

Table 2: Performance comparison on real-world traces. Results are averaged over 5 independent runs (mean ± standard deviation). MRDF (Oracle) uses true type means; among feasible learning algorithms, S-UCB performs best.

Dataset	Algorithm	Total Reward (↑)	Cumulative Regret (↓)
Azure Serverless	MRDF (Oracle)	$2,193.9 \pm 3.4$	0 ± 0
	S-UCB (Ours)	$2,176.9 \pm 5.4$	17.0 ± 5.0
	Emp-MRDF	$2,154.2 \pm 18.9$	39.7 ± 15.5
	ETS	$2,163.5 \pm 4.4$	30.4 ± 6.1
	Random	$1,563.4 \pm 55.6$	630.5 ± 75.1
Alibaba PAI	MRDF (Oracle)	$15,591.8 \pm 25.3$	0 ± 0
	S-UCB (Ours)	$14,909.3 \pm 26.8$	682.5 ± 13.1
	Emp-MRDF	$14,904.2 \pm 32.2$	687.6 ± 14.5
	ETS	$13,003.1 \pm 33.6$	$2,588.7 \pm 22.0$
	Random	$3,608.0 \pm 398.9$	$11,983.8 \pm 426.0$

inherently provide temporal features (arrival time b_k and processing length l_k) but lack explicit business-level "rewards." To faithfully simulate the bandit learning environment, we adopt a standard synthesis approach [Lattimore and Szepesvári, 2020]. For each dataset, we first identify the top N most frequent application types to form the type space. We then assign a hidden, true mean reward μ_n to each type n. During the simulation, when a task of type n is successfully completed, the scheduler observes a stochastic reward r_k drawn from a truncated Gaussian distribution $\mathcal{N}(\mu_n, \sigma^2)$, bounded strictly above zero. If a task is preempted or rejected, the reward and its observation are entirely censored, rigorously enforcing the partial feedback challenge addressed by our theoretical model.

As shown in Table 2, operating strictly online with zero prior knowledge of the reward distributions, S-UCB achieves cumulative regret of 17.0 ± 5.0 on Azure Serverless and 682.5 ± 13.1 on Alibaba PAI, outperforming Emp-MRDF, ETS, and Random. The Azure trace is relatively favorable to learning because repeated application identifiers produce frequent reward observations; nevertheless, S-UCB still improves over Emp-MRDF by using optimism when early empirical means are noisy. Alibaba PAI is more demanding because machine-learning workloads induce larger variation in processing lengths and denser contention. There, S-UCB and Emp-MRDF are close, indicating that empirical means eventually become informative for frequent task types, but S-UCB retains the best feasible regret and substantially outperforms ETS, whose uniform exploration phase is too expensive in a saturated immediate-decision system. Across both traces, integrating upper confidence bounds with processing lengths through optimistic density lets S-UCB approach the true-mean MRDF oracle while maintaining low variance.

7 DISCUSSION AND EXTENSION

Mitigating worst-case instances via finite buffers. While our primary model enforces strict immediate decisions, practical streaming systems often permit tasks to be held briefly before an irrevocable rejection. To bridge this gap, we analyze a buffered variant where each machine maintains a buffer of capacity B. Under this variant, the scheduler continuously retains up to B pending or running candidates possessing the highest (remaining) densities. As demonstrated below, even a small buffer significantly softens the worst-case penalty.

Theorem 7.1 (Buffered MRDF, informal; see Theorem F.1)**.** *Let $R := L_{\max}^{1/M}$. Under a buffer capacity B, the buffered MRDF policy (B-MRDF) satisfies the following for any instance I:*

$$\frac{R_{\text{B-MRDF}}(I)}{R_{\text{OPT}}(I)} \geq \frac{B+1}{M\big((B+1)+3L_{\max}^{1/M}\big)}.$$

In particular, when $B \ll L_{\max}^{1/M}$, this improves the worst-case competitive ratio by a factor of $\Theta(B)$ compared to the strictly unbuffered setting.

The practical necessity of deterministic scheduling. Theoretical online algorithms frequently employ randomized scheduling policies to achieve superior worst-case competitive bounds. However, we intentionally design our proposed MRDF and S-UCB algorithms to be purely deterministic, as this provides three critical advantages in practical, latency-sensitive environments (such as serverless computing and LLM inference):

- **Predictability and SLA Compliance:** In production systems, predictability is paramount. Deterministic policies ensure consistent execution prioritization based on transparent metrics (e.g., optimistic density). This consistency is essential for satisfying strict Service Level Agreements (SLAs), whereas randomized algorithms risk arbitrarily dropping high-value requests and degrading user trust.
- **Robustness against Action-Induced Censoring:** In our specific bandit formulation, preempted tasks yield no reward and no observation. Algorithmic randomness severely exacerbates the variance associated with this censored feedback. For instance, a randomized policy might abruptly preempt a task just before completion, permanently destroying a valuable learning observation and stalling convergence. By contrast, S-UCB deterministically controls exploration via upper confidence bounds, ensuring sample-efficient learning without wasteful interruptions.
- **System Overhead and Reproducibility:** Deterministic algorithms impose minimal computational overhead, enabling microsecond-level dispatching. Crucially, they also allow for exact deterministic replay, which is an indispensable feature for system debugging and trace-based performance profiling in large-scale cloud clusters.

8 CONCLUSION

Our study on online scheduling with immediate decisions and uncertain rewards, motivated by streaming task applications, has yielded significant results. By initially focusing on the case of known rewards, we are able to establish the worst-case competitive ratio of $\tilde{O}(1/(ML_{\max}^{1/M}))$, which provides a clear benchmark for understanding the performance limits of the scheduling problem under such conditions. The proposed near-optimal scheduling MRDF (Algorithm 1) for this scenario achieves the competitive ratio of $\Omega(1/(ML_{\max}^{1/M}))$. When it comes to the more challenging situation of random and unknown rewards, our proposed S-UCB (Algorithm 2) algorithm attains an $O(\log T/\Delta)$ regret, showcasing its effectiveness in handling the inherent uncertainty. Additionally, we also provide the competitive ratio analysis for the S-UCB algorithm. Extensive experiments compared with baselines also show the effectiveness of our proposed algorithms. The main conceptual message is that immediate-decision scheduling and bandit learning cannot be treated as two separate layers: the scheduling rule determines which rewards are ever observed, and the learning rule must therefore preserve enough completions to make estimation possible. The MRDF pseudo-task construction provides this link by making the density of a running task increase as it approaches completion, and S-UCB inherits the same structure with optimistic rewards. We view richer buffer models, adaptive capacity constraints, and non-stationary reward distributions as natural next steps for extending this framework.

ACKNOWLEDGEMENT

The corresponding author Shuai Li is supported by National Natural Science Foundation of China (62376154).

References

Arash Asadpour, Xuan Wang, and Jiawei Zhang. Online resource allocation with limited flexibility. *Management Science*, 66(2):642–666, 2020.

Ran Canetti and Sandy Irani. Bounding the power of preemption in randomized scheduling. In *Proceedings of the twenty-seventh annual ACM symposium on Theory of computing*, pages 606–615, 1995.

Semih Cayci, Atilla Eryilmaz, and Rayadurgam Srikant. Learning to control renewal processes with bandit feedback. *Proceedings of the ACM on Measurement and Analysis of Computing Systems*, 3(2):1–32, 2019.

Bo Chen, André Van Vliet, and Gerhard J Woeginger. An optimal algorithm for preemptive on-line scheduling. In *Algorithms—ESA'94: Second Annual European Symposium Utrecht, The Netherlands, September 26–28, 1994 Proceedings 2*, pages 300–306. Springer, 1994.

Wei Chen, Yajun Wang, and Yang Yuan. Combinatorial multi-armed bandit: General framework and applications. In *Proceedings of the 30th International Conference on Machine Learning*, pages 151–159, 2013.

Wei Chen, Yajun Wang, Yang Yuan, and Qinshi Wang. Combinatorial multi-armed bandit and its extension to probabilistically triggered arms. *Journal of Machine Learning Research*, 17(50):1–33, 2016.

John Dickerson, Karthik Sankararaman, Kanthi Sarpatwar, Aravind Srinivasan, Kung-Lu Wu, and Pan Xu. Online resource allocation with matching constraints. In *International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, 2019.

Farbod Ekbatani, Yiding Feng, Ian Kash, and Rad Niazadeh. Online job assignment. *arXiv preprint arXiv:2506.06893*, 2025.

Leah Epstein and Lene M Favrholdt. Optimal non-preemptive semi-online scheduling on two related machines. *Journal of Algorithms*, 57(1):49–73, 2005.

Amos Fiat and Gerhard J Woeginger. On-line scheduling on a single machine: Minimizing the total completion time. *Acta Informatica*, 36(4):287–293, 1999.

Guillermo Gallego, Anran Li, Van-Anh Truong, and Xinshang Wang. Online resource allocation with customer choice. *arXiv preprint arXiv:1511.01837*, 2015.

Chao Gao, Tong Mo, Taylor Zowtuk, Tanvir Sajed, Laiyuan Gong, Hanxuan Chen, Shangling Jui, and Wei Lu. Bansor: Improving tensor program auto-scheduling with bandit based reinforcement learning. In *2021 IEEE 33rd International Conference on Tools with Artificial Intelligence (ICTAI)*, pages 273–278. IEEE, 2021.

Ronald L Graham. Bounds for certain multiprocessing anomalies. *Bell system technical journal*, 45(9):1563–1581, 1966.

I-H Hou, Vivek Borkar, and PR Kumar. *A theory of QoS for wireless*. IEEE, 2009.

Tor Lattimore and Csaba Szepesvári. *Bandit algorithms*. Cambridge University Press, 2020.

S Leonardi. A simpler proof of preemptive flow-time approximation. *Approximation and on-line Algorithms*, 2003.

Stefano Leonardi and Danny Raz. Approximating total flow time on parallel machines. In *Proceedings of the twenty-ninth annual ACM symposium on Theory of computing*, pages 110–119, 1997.

Yanying Lin, Shuaipeng Wu, Shutian Luo, Hong Xu, Haiying Shen, Chong Ma, Min Shen, Le Chen, Chengzhong Xu, Lin Qu, et al. Understanding diffusion model serving in production: A top-down analysis of workload, scheduling, and resource efficiency. In *Proceedings of the 2025 ACM Symposium on Cloud Computing*, pages 1–15, 2025.

Brendan Lucier, Ishai Menache, Joseph Naor, and Jonathan Yaniv. Efficient online scheduling for deadline-sensitive jobs. In *Proceedings of the twenty-fifth annual ACM symposium on Parallelism in algorithms and architectures*, pages 305–314, 2013.

Kirk Pruhs, Jiri Sgall, and Eric Torng. Online scheduling., 2004.

Haoran Qiu, Anish Biswas, Zihan Zhao, Jayashree Mohan, Alind Khare, Esha Choukse, Íñigo Goiri, Zeyu Zhang, Haiying Shen, Chetan Bansal, Ramachandran Ramjee, and Rodrigo Fonseca. Modserve: Modality- and stage-aware resource disaggregation for scalable multimodal model serving. In *Proceedings of the 2025 ACM Symposium on Cloud Computing (SoCC 2025)*, New York, NY, USA, 2025. Association for Computing Machinery.

Hanna Sumita, Shinji Ito, Kei Takemura, Daisuke Hatano, Takuro Fukunaga, Naonori Kakimura, and Ken-ichi Kawarabayashi. Online task assignment problems with reusable resources. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 36, pages 5199–5207, 2022.

Jing Wang, Miao Yu, Peng Zhao, and Zhi-Hua Zhou. Learning with adaptive resource allocation. *Proceedings of the 41st International Conference on Machine Learning (ICML 2024)*, 2024.

Zhi-Hua Zhou. Learnability with time-sharing computational resource concerns, 2024. URL https://arxiv.org/abs/2305.02217.

A PROOF OF THEOREM 4.1

The task set is constructed as follows:

For the task 0: $b_0 = 1, \ell_0 = L_{\max}, r_0 = L_{\max}$.

For the task 1: $b_1 = 1, \ell_1 = 1, r_1 = 1$.

For the task i with $2 \le i \le L_{\max}$: $b_i = i, \ell_i = 1, r_i = \frac{2\log L_{\max}}{L_{\max}} \sum_{j=1}^{i-1} r_j$.

Then consider any deterministic scheduling algorithm $\mathcal{A}$. If the algorithm $\mathcal{A}$ selects 1 at time 1, consider another task instance which has only tasks $0, 1$, the ratio it gets is $1/L_{\max} < 2\log(L_{\max})/L_{\max}$.

If $\mathcal{A}$ selects 0 at first but switches to i with $2 \le i \le L_{\max}$, we consider the task instance which only has $0, \cdots, i$, and the rewards $\mathcal{A}$ loses is at least $\sum_{j=1}^{i-1} r_j$, and the corresponding competitive ratio $\mathcal{A}$ gets is at most $r_i / \sum_{j=1}^{i-1} r_j = 2\log(L_{\max})/L_{\max}$.

If $\mathcal{A}$ selects 0 and completes it, the maximum total reward is by selecting tasks $1, \cdots, L_{\max}$. Since $r_i = \frac{2\log(L_{\max})}{L_{\max}} \sum_{j=1}^{i-1} r_j$, we have that

$$\sum_{j=1}^{i} r_j = \left(1 + \frac{2\log L_{\max}}{L_{\max}}\right) \sum_{j=1}^{i-1} r_j = \left(1 + \frac{2\log L_{\max}}{L_{\max}}\right)^{i-1}.$$

The competitive ratio $\mathcal{A}$ achieves is

$$\frac{r_0}{\sum_{i=1}^{L_{\max}} r_i} = \frac{L_{\max}}{(1 + \frac{2\log L_{\max}}{L_{\max}})^{L_{\max}-1}} < \frac{2\log(L_{\max})}{L_{\max}},$$

the inequality holds when $L_{\max} \ge 6$.

All possible behaviors of the algorithm $\mathcal{A}$ have been analyzed, and no one can achieve the ratio larger than $\frac{2\log L_{\max}}{L_{\max}}$. Thus any deterministic can not achieve the competitive ratio larger than $\frac{2\log(L_{\max})}{L_{\max}}$.

Proof. The instance I is designed as follows:

- For the task with length $L_{\max}$, we design a task with $b_{M,1} = 1, \ell_{M,1} = L_{\max}, r_{M,1} = L_{\max}$.
- For the task with length $L_{\max}^{\frac{M-1}{M}}$, we design $2L_{\max}^{\frac{1}{M}}$ tasks, indexed by $(M-1, i, j)$, $i \in [L_{\max}^{\frac{1}{M}}], j \in \{0, 1\}$. For $i = 1$: $b_{M-1,1,0} = b_{M-1,1,1} = 1$, $\ell_{M-1,1,0} = \ell_{M-1,1,1} = L_{\max}^{\frac{M-1}{M}}$, $r_{M-1,1,0} = r_{M-1,1,1} = L_{\max}^{\frac{M-1}{M}}$. For $i > 1$: $b_{M-1,i,0} = b_{M-1,i,1} = (i-1)L_{\max}^{\frac{M-1}{M}} + 1$, $\ell_{M-1,i,0} = \ell_{M-1,i,1} = L_{\max}^{\frac{M-1}{M}}$, $r_{M-1,i,0} = r_{M-1,i,1} = \frac{2\log L_{\max}^{1/M}}{L_{\max}^{1/M}} \sum_{i'=1}^{i-1} r_{M-1,i',0}$.
- For the task with length $L_{\max}^{\frac{m}{M}}$, $m \ge 1$, we design $2L_{\max}^{\frac{M-m}{M}}$ tasks, indexed by $(m, i, j), i \in [L_{\max}^{\frac{M-m}{M}}], j \in \{0, 1\}$. For $i = kL_{\max}^{\frac{1}{M}} + 1$, $0 \le k < M - m$, $b_{m,i,0} = b_{m,i,1} = (i-1)L_{\max}^{\frac{m}{M}} + 1$, $\ell_{m,i,0} = \ell_{m,i,1} = L_{\max}^{\frac{m}{M}}$, $r_{m,i,0} = r_{m,i,1} = r_{m+1,k,0}/L_{\max}^{\frac{1}{M}}$. For $i = kL_{\max}^{\frac{1}{M}} + n$, $1 < n \le L_{\max}^{\frac{1}{M}}$. $b_{m,i,0} = b_{m,i,1} = (i-1)L_{\max}^{\frac{m}{M}} + 1$, $\ell_{m,i,0} = \ell_{m,i,1} = L_{\max}^{\frac{m}{M}}$, $r_{m,i,0} = r_{m,i,1} = \frac{2\log L_{\max}^{1/M}}{L_{\max}^{1/M}} \sum_{n'=1}^{n-1} r_{m,kL_{\max}^{1/M}+n',0}$.
- For the task with length 1, we design $L_{\max}$ tasks, indexed by $(1, i)$, $i \in [L_{\max}]$. For $i = kL_{\max}^{\frac{1}{M}} + 1$, $0 \le k < M$, $b_{1,i} = i$, $\ell_{1,i} = 1$, $r_{1,i} = r_{2,k,0}/L_{\max}^{\frac{1}{M}}$. For $i = kL_{\max}^{\frac{1}{M}} + n$, $1 < n \le L_{\max}^{\frac{1}{M}}$. $b_{1,i} = i$, $\ell_{1,i} = 1$, $r_{1,i} = \frac{2\log L_{\max}^{1/M}}{L_{\max}^{1/M}} \sum_{n'=1}^{n-1} r_{1,kL_{\max}^{1/M}+n'}$.

Consider any deterministic scheduling algorithm $\mathcal{A}$. At time $t = 1$ algorithm $\mathcal{A}$ selects M tasks among these $2M$ tasks. Sort and index the selected tasks by increasing order of length. Similarly, those unselected tasks are also sorted by the increasing order. And thus the selected task $i_k, k \in [M]$ competes with the unselected task i'_k. By the instance design it holds that $\ell_{i_k} \ne \ell_{i'_k}, \forall k \in [M]$. If $\ell_{i_k} < \ell_{i'_k}$, we construct another instance $\mathcal{I}'$ such that task i_k does not appear until task $\ell_{i'_k}$ ends. The competitive ratio of it is larger than $1/L_{\max}^{1/M}$. If $\ell_{i_k} > \ell_{i'_k}$, the instance $\mathcal{I}'$ receives task $\ell_{i'_k}$ as $\mathcal{I}$ does.

If $\mathcal{A}$ switches its selected tasks at some time t. It must switch task i with length ℓ_i to task i' with shorter length ℓ'_i. Since the algorithm does not select tasks with length $\ell_{i'}$ before t, the ratio it loses for task i' is

$$\frac{r_{i'}}{\sum_{j=1}^{i'-1} r_j} = \frac{2\log L_{\max}}{ML_{\max}^{1/M}}.$$

After that, the algorithm resorts the selected tasks and unselected tasks, and then pairs the compete tasks to construct the coming tasks for another instance $\mathcal{I}'$.

If $\mathcal{A}$ completes a task i_k and the competitive task i'_k has shorter length. The loses of rewards is the summation of those shorter tasks, which induces the competitive ratio no larger than

$$\frac{r_{i,k}}{\sum_{i'=1}^{L_{\max}^{1/M}} r_{i'}} \leq \frac{2\log L_{\max}^{1/M}}{L_{\max}^{1/M}} = \frac{2\log L_{\max}}{ML_{\max}^{1/M}}.$$

Thus we have enumerated all possible actions of the algorithm $\mathcal{A}$, and it can not achieve the competitive ratio larger than $\frac{2\log L_{\max}}{ML_{\max}^{1/M}}$.

Then we turn to a more complicated M-machine setting.

Theorem A.1. *There exists an instance containing a set of tasks such that no deterministic algorithm can achieve the competitive ratio greater than* $2\log(L_{\max})/ML_{\max}^{1/M}$.

For the simplicity suppose $L_{\max}^{1/M}$ is an integer. The instance I is designed as follows:

- For the task with length $L_{\max}$, we design a task with $b_{M,1} = 1, \ell_{M,1} = L_{\max}, r_{M,1} = L_{\max}$.
- For the task with length $L_{\max}^{\frac{M-1}{M}}$, we design $2L_{\max}^{\frac{1}{M}}$ tasks, indexed by $(M-1,i,j), i \in [L_{\max}^{\frac{1}{M}}], j \in \{0,1\}$. For $i=1$: $b_{M-1,1,0} = b_{M-1,1,1} = 1$, $\ell_{M-1,1,0} = \ell_{M-1,1,1} = L_{\max}^{\frac{M-1}{M}}$, $r_{M-1,1,0} = r_{M-1,1,1} = L_{\max}^{\frac{M-1}{M}}$. For $i > 1$: $b_{M-1,i,0} = b_{M-1,i,1} = (i-1)L_{\max}^{\frac{M-1}{M}} + 1$, $\ell_{M-1,i,0} = \ell_{M-1,i,1} = L_{\max}^{\frac{M-1}{M}}$, $r_{M-1,i,0} = r_{M-1,i,1} = \frac{2\log L_{\max}^{1/M}}{L_{\max}^{1/M}} \sum_{i'=1}^{i-1} r_{M-1,i',0}$.
- For the task with length $L_{\max}^{\frac{m}{M}}$, $m \geq 1$, we design $2L_{\max}^{\frac{M-m}{M}}$ tasks, indexed by $(m,i,j), i \in [L_{\max}^{\frac{M-m}{M}}], j \in \{0,1\}$. For $i = kL_{\max}^{\frac{1}{M}} + 1$, $0 \leq k < M-m$, $b_{m,i,0} = b_{m,i,1} = (i-1)L_{\max}^{\frac{m}{M}} + 1$, $\ell_{m,i,0} = \ell_{m,i,1} = L_{\max}^{\frac{m}{M}}$, $r_{m,i,0} = r_{m,i,1} = r_{m+1,k,0}/L_{\max}^{\frac{1}{M}}$. For $i = kL_{\max}^{\frac{1}{M}} + n$, $1 < n \leq L_{\max}^{\frac{1}{M}}$. $b_{m,i,0} = b_{m,i,1} = (i-1)L_{\max}^{\frac{m}{M}} + 1$, $\ell_{m,i,0} = \ell_{m,i,1} = L_{\max}^{\frac{m}{M}}$, $r_{m,i,0} = r_{m,i,1} = \frac{2\log L_{\max}^{1/M}}{L_{\max}^{1/M}} \sum_{n'=1}^{n-1} r_{m,kL_{\max}^{1/M}+n',0}$.
- For the task with length 1, we design $L_{\max}$ tasks, indexed by $(1,i)$, $i \in [L_{\max}]$. For $i = kL_{\max}^{\frac{1}{M}} + 1$, $0 \leq k < M$, $b_{1,i} = i$, $\ell_{1,i} = 1$, $r_{1,i} = r_{2,k,0}/L_{\max}^{\frac{1}{M}}$. For $i = kL_{\max}^{\frac{1}{M}} + n$, $1 < n \leq L_{\max}^{\frac{1}{M}}$. $b_{1,i} = i$, $\ell_{1,i} = 1$, $r_{1,i} = \frac{2\log L_{\max}^{1/M}}{L_{\max}^{1/M}} \sum_{n'=1}^{n-1} r_{1,kL_{\max}^{1/M}+n'}$.

Intuitively, by this design, we always have $2M$ tasks at every time $t \in [L_{\max}]$. Among these $2M$ tasks, every two tasks with the nearest lengths form a competitive pair. Specifically, the task with length $L_{\max}$ is challenged by tasks with length $L_{\max}^{\frac{M-1}{M}}$, and tasks with length $L_{\max}^{\frac{m}{M}}$ are challenged by tasks with length $L_{\max}^{\frac{m-1}{M}}$. Since there are M machines, there are at least M tasks abandoned at each time, causing the possibility to construct the instance with competitive ratio of $\log L_{\max}^{1/M}/L_{\max}^{1/M}$.

Consider any deterministic scheduling algorithm $\mathcal{A}$. At time 1 algorithm $\mathcal{A}$ selects M tasks among these $2M$ tasks. Sort and index the selected tasks by increasing order of length. Those unselected tasks are also sorted by the increasing order. And thus the selected task $i_k, k \in [M]$ competes with the unselected task i'_k. By the instance design we have that $\ell_{i_k} \neq \ell_{i'_k}$. If $\ell_{i_k} < \ell_{i'_k}$, we construct another instance $\mathcal{I}'$ and that task i_k does not appear until task $\ell_{i'_k}$ ends. The competitive ratio of it is larger than $1/L_{\max}^{1/M}$.

If $\mathcal{A}$ switches its selected tasks at time t. It must switch to tasks with shorter length. It loses reward of summation of those unselected shorter task before t, which is $L_{\max}^{1/M}/(2\log L_{\max}^{1/M})$ times the short task's reward, leading to a competitive ratio of $2\log L_{\max}^{1/M}/L_{\max}^{1/M}$.

If $\mathcal{A}$ completes a task i_k and the competitive task i'_k has shorter length. The loses of rewards is the summation of those shorter tasks, which is at least $L_{\max}^{1/M}/(2\log L_{\max}^{1/M})$ times r_{i_k}, also leading to the competitive ratio of $2\log L_{\max}^{1/M}/L_{\max}^{1/M}$.

Thus we have enumerated all possible actions of the algorithm $\mathcal{A}$, and it can not achieve the competitive ratio larger than $2\log L_{\max}^{1/M}/L_{\max}^{1/M} = \frac{2\log L_{\max}}{ML_{\max}^{1/M}}$. □

B PROOF OF THEOREM 4.2

Proof. We begin our analysis with one machine setting.

Denote $\mathcal{L}_{ALG}$ as the set of tasks the algorithm completes, $\mathcal{L}_{OPT}$ as the set of tasks of the optimal schedule that maximize the total reward.

For each task $k \in \mathcal{L}_{ALG}$, we analyze the optimal tasks in $\mathcal{L}_{OPT}$ related to k. Denote $k_{pre} \in \arg\max_{k' \in \mathcal{L}_{ALG}, b_{k'} < b_k} b_{k'}$ as the nearest completed task before k by the algorithm. Then we can define the optimal task set $\mathcal{L}_{OPT,k} = \{k' \in \mathcal{L}_{OPT} : b_{k_{pre}} + \ell_{k_{pre}} \le b_{k'} < b_k + \ell_k\}$ as the set of tasks in $\mathcal{L}_{OPT}$ such that it arrives after k_{pre} is completed and before k is completed. By this split it is easy to verify that the union of $\mathcal{L}_{OPT,k}$ covers all tasks in $\mathcal{L}_{OPT}$.

$$\bigcup_{k \in \mathcal{ALG}} \mathcal{L}_{OPT,k} = \mathcal{L}_{OPT}\,.$$

Thus it suffices to analyze the reward relationship between r_k and $\sum_{k' \in \mathcal{L}_{OPT,k}} r_{k'}$ for each task $k \in \mathcal{L}_{ALG}$.

For the set $\mathcal{L}_{OPT,k}$, we further split it into three subsets: $\mathcal{L}_{OPT,k,1} = \{k' \in \mathcal{L}_{OPT,k} : b_{k'} + \ell_{k'} \le b_k\}$ is the set of optimal tasks completed before k arrives; $\mathcal{L}_{OPT,k,2} = \{k' \in \mathcal{L}_{OPT,k} : b_{k'} \ge b_k, b_{k'} + \ell_{k'} \le b_k + \ell_k\}$ is the set of optimal tasks arrives after k arrives and completed before k is completed; $\mathcal{L}_{OPT,k,3} = \{k' \in \mathcal{L}_{OPT,k} : b_{k'} + \ell_{k'} > b_k + \ell_k\}$ is the set of optimal tasks completed after k is completed. Since tasks in $\mathcal{L}_{OPT}$ are disjoint, it is easy to check that

$$\mathcal{L}_{OPT,k,1} \cup \mathcal{L}_{OPT,k,2} \cup \mathcal{L}_{OPT,k,3} = \mathcal{L}_{OPT,k}\,.$$

We first upper bound the rewards of tasks in $\mathcal{L}_{OPT,k,1}$. Since between time $b_{k_{pre}} + \ell_{k_{pre}}$ and b_k the algorithm completes no task, this indicates that the algorithm keeps processing some tasks at those times but switches to other tasks before it is completed, and in the end, the algorithm switches to task k at $t = b_k$ and completes it. Denote u_t as the processing task by algorithm at time t and $\rho_t = r_{u_t}/(\ell_{u_t} - (t - b_{u_t}))$ as the remaining density of that processing task. By the algorithm's selecting rule, ρ_t is the maximum density of all arriving tasks at time t. Thus the total rewards in $\mathcal{L}_{OPT,k,1}$ is maximized by

$$\sum_{k' \in \mathcal{L}_{OPT,k,1}} r_{k'} \le \sum_{t=b_{k_{pre}}+\ell_{k_{pre}}}^{b_k-1} \rho_t\,.$$

Then it suffices to upper bound ρ_t at each time t. Since the algorithm does not complete any task at $t \in [b_{k_{pre}} + \ell_{k_{pre}}, b_k]$, the density ρ_t is increasing with time t.

We first consider the density ρ_t at $t = b_k - 1$. Recall that u_{b_k-1} is the processing task at time $t = b_k - 1$ and $\rho_t = r_{u_t}/(\ell_{u_t} - (t - b_{u_t}))$. Since the algorithm switches to k at time b_k, it indicates that $r_k/\ell_k > r_{u_t}/(\ell_{u_t} - (t - b_{u_t}) - 1)$. Thus $\rho_t = r_{u_t}/(\ell_{u_t} - (t - b_{u_t})) < \frac{r_k}{\ell_k}\frac{\ell_{u_t} - (t - b_{u_t}) - 1}{\ell_{u_t} - (t - b_{u_t})}$, which is maximized by setting $\ell_{u_t} = L_{\max}, b_{u_t} = t$, and thus $\rho_t \le \frac{L_{\max}-1}{L_{\max}}\frac{r_k}{\ell_k}$ at time $t = b_k - 1$.

Then we turn to consider the density ρ_t at $t = b_k - 2$. (1) If $u_{b_k-2} = u_{b_k-1}$, which means the algorithm does not switch at $t = b_k - 1$, and switches to task k at $t = b_k$. Thus we have $\frac{r_k}{\ell_k} > \frac{r_{u_{b_k-2}}}{\ell_{u_{b_k-2}} - (b_k - b_{u_{b_k-2}})}$. This is maximized by setting $b_{u_{b_k-2}} = b_k - 2, \ell_{u_{b_k-2}} = L_{\max}$ which implies $\rho_{b_k-2} \le \frac{L_{\max}-2}{L_{\max}}\frac{r_k}{\ell_k}$. (2) If $u_{b_k-2} \ne u_{b_k-1}$, which means the algorithm switches to another task at $t = b_k - 1$. Then we have already know that $\rho_{b_k-1} \le \frac{L_{\max}-1}{L_{\max}}\frac{r_k}{\ell_k}$, and similar analysis for $t = b_k - 2$ by setting $b_{u_{b_k-2}} = b_k - 2, \ell_{u_{b_k-2}} = L_{\max}$, the density is maximized by $\rho_{b_k-2} \le (\frac{L_{\max}-1}{L_{\max}})^2\frac{r_k}{\ell_k}$. Since $(\frac{L_{\max}-1}{L_{\max}})^2 > \frac{L_{\max}-2}{L_{\max}}$, ρ_{b_k-2} is maximized by $(\frac{L_{\max}-1}{L_{\max}})^2\frac{r_k}{\ell_k}$.

We prove the following bound of ρ_t by induction. Suppose for any $i' \le i$, $\rho_{b_k-i} \le (\frac{L_{\max}-1}{L_{\max}})^i\frac{r_k}{\ell_k}$. Then consider time $t = b_k - (i+1)$, suppose the algorithm processes task u_t at $t = b_k - (i+1)$ and switches to another task at t' with $t < t' \le b_k$. Since the maximum density at t' is $(\frac{L_{\max}-1}{L_{\max}})^{b_k-t'}\frac{r_k}{\ell_k}$. By the selecting rule we have that $\frac{r_{u_t}}{\ell_{u_t} - (t' - b_{u_t})} < (\frac{L_{\max}-1}{L_{\max}})^{b_k-t'}\frac{r_k}{\ell_k}$,

which implies $\frac{r_{u_t}}{\ell_{u_t}-(t-b_{u_t})} < \frac{\ell_{u_t}-(t'-b_{u_t})}{\ell_{u_t}-(t-b_{u_t})}(\frac{L_{\max}-1}{L_{\max}})^{b_k-t'}\frac{r_k}{\ell_k}$. This is maximized by setting $b_{u_t}=t, \ell_{u_t}=L_{\max}$, and we have

$$\begin{aligned}
\rho_t \leq & \frac{L_{\max}-(t'-t)}{L_{\max}}\left(\frac{L_{\max}-1}{L_{\max}}\right)^{b_k-t'}\frac{r_k}{\ell_k} \\
= & \prod_{j=1}^{t'-t}\frac{L_{\max}-j}{L_{\max}-j+1}\left(\frac{L_{\max}-1}{L_{\max}}\right)^{b_k-t}\frac{r_k}{\ell_k} \\
\leq & \left(\frac{L_{\max}-1}{L_{\max}}\right)^{t'-t}\left(\frac{L_{\max}-1}{L_{\max}}\right)^{b_k-t'}\frac{r_k}{\ell_k} \\
\leq & \left(\frac{L_{\max}-1}{L_{\max}}\right)^{b_k-t}\frac{r_k}{\ell_k} \\
= & \left(\frac{L_{\max}-1}{L_{\max}}\right)^{i+1}\frac{r_k}{\ell_k},
\end{aligned}$$

which is maximized by switching at time $t+1$. Thus we can conclude that for each time $t \leq b_k$, $\rho_t \leq \left(\frac{L_{\max}-1}{L_{\max}}\right)^{b_k-t}\frac{r_k}{\ell_k}$. Thus

$$\sum_{t=b_{k_{pre}}+\ell_{k_{pre}}}^{b_k-1}\rho_t \leq \sum_{t=b_{k_{pre}}+\ell_{k_{pre}}}^{b_k-1}\left(\frac{L_{\max}-1}{L_{\max}}\right)^{b_k-t}\frac{r_k}{\ell_k} \leq L_{\max}\frac{r_k}{\ell_k}.$$

This implies that

$$r_k \geq \frac{\ell_k}{L_{\max}}\sum_{t=b_{k_{pre}}+\ell_{k_{pre}}}^{b_k-1}\rho_t \geq \frac{\ell_k}{L_{\max}}\sum_{k'\in\mathcal{L}_{OPT,k,1}} r_{k'}.$$

For optimal tasks in $\mathcal{L}_{OPT,k,2}$, the maximum total reward is obtained by setting optimal task k' with $b_{k'}=t, \ell_{k'}=1, r_{k'}=\frac{r_k}{\ell_k-(t-b_k)}$ (the maximum reward one can set when no switch happens). Then we have

$$\sum_{k'\in\mathcal{L}_{OPT,k,2}} r_{k'} \leq \sum_{t=b_k}^{b_k+\ell_k-1}\frac{r_k}{\ell_k-(t-b_k)} \leq \ell_k r_k.$$

This indicates

$$r_k \geq \frac{1}{\ell_k}\sum_{k'\in\mathcal{L}_{OPT,k,2}} r_{k'}.$$

For optimal tasks in $\mathcal{L}_{OPT,k,3}$, we know that this set contains at most one task that is completed after $b_k+\ell_k$ (other tasks will be classified to the next task set). This optimal task k' is maximized by setting $b_{k'}=b_k+\ell_k-1, \ell_{k'}=L_{\max}, r_{k'}=L_{\max}r_k$. And thus we have

$$r_k \geq \frac{1}{L_{\max}}r_{k'}.$$

Thus together we have

$$\begin{aligned}
r_k \geq & \frac{\ell_k}{L_{\max}}\sum_{k'\in\mathcal{L}_{OPT,k,1}} r_{k'}, \\
r_k \geq & \frac{1}{\ell_k}\sum_{k'\in\mathcal{L}_{OPT,k,2}} r_{k'}, \\
r_k \geq & \frac{1}{L_{\max}}\sum_{k'\in\mathcal{L}_{OPT,k,3}} r_{k'},
\end{aligned}$$

which implies

$$\begin{aligned} r_k \geq & \frac{1}{3L_{\max}} \sum_{k' \in \mathcal{L}_{OPT,k,1} \cup \mathcal{L}_{OPT,k,2} \cup \mathcal{L}_{OPT,k,3}} r_{k'} \\ = & \frac{1}{3L_{\max}} \sum_{k' \in \mathcal{L}_{OPT,k}} r_{k'} \,. \end{aligned}$$

Summing over all tasks $k \in \mathcal{L}_{ALG}$,

$$\begin{aligned} & \sum_{k \in \mathcal{L}_{ALG}} r_k \\ \geq & \frac{1}{3L_{\max}} \sum_{k \in \mathcal{L}_{ALG}} \sum_{k' \in \mathcal{L}_{OPT,k}} r_{k'} \\ = & \frac{1}{3L_{\max}} \sum_{k' \in \mathcal{L}_{OPT}} r_{k'} \,. \end{aligned}$$

The algorithm can achieve $1/(3L_{\max})$ competitive ratio.

Then we turn to the more complicated M machines setting.

Denote $\mathcal{L}_{ALG}$ as the set of tasks the machine $i \in [M]$ completes, $\mathcal{L}_{OPT}$ as the set of tasks of the optimal schedule of tasks length between $L_{\max}^{(i-1)/M}$ and $L_{\max}^{i/M}$ that maximize the total reward.

For each task $k \in \mathcal{L}_{ALG}$, we analyze the optimal tasks in $\mathcal{L}_{OPT}$ related to k. Denote $k_{pre} \in \arg\max_{k' \in \mathcal{L}_{ALG}, b_{k'} < b_k} b_{k'}$ as the nearest completed task before k by the algorithm. Then we can define the optimal task set $\mathcal{L}_{OPT,k} = \{k' \in \mathcal{L}_{OPT} : b_{k_{pre}} + \ell_{k_{pre}} \leq b_{k'} < b_k + \ell_k\}$ as the set of tasks in $\mathcal{L}_{OPT}$ such that it arrives after k_{pre} is completed and before k is completed. By this split it is easy to verify that the union of $\mathcal{L}_{OPT,k}$ covers all tasks in $\mathcal{L}_{OPT}$.

$$\bigcup_{k \in \mathcal{ALG}} \mathcal{L}_{OPT,k} = \mathcal{L}_{OPT} \,.$$

Thus it suffices to analyze the reward relationship between r_k and $\sum_{k' \in \mathcal{L}_{OPT,k}} r_{k'}$ for each task $k \in \mathcal{L}_{ALG}$.

For the set $\mathcal{L}_{OPT,k}$, we further split it into three subsets: $\mathcal{L}_{OPT,k,1} = \{k' \in \mathcal{L}_{OPT,k} : b_{k'} + \ell_{k'} \leq b_k\}$ is the set of optimal tasks completed before k arrives; $\mathcal{L}_{OPT,k,2} = \{k' \in \mathcal{L}_{OPT,k} : b_{k'} \geq b_k, b_{k'} + \ell_{k'} \leq b_k + \ell_k\}$ is the set of optimal tasks arrives after k arrives and completed before k is completed; $\mathcal{L}_{OPT,k,3} = \{k' \in \mathcal{L}_{OPT,k} : b_{k'} + \ell_{k'} > b_k + \ell_k\}$ is the set of optimal tasks completed after k is completed. Since tasks in $\mathcal{L}_{OPT}$ are disjoint, it is easy to check that

$$\mathcal{L}_{OPT,k,1} \cup \mathcal{L}_{OPT,k,2} \cup \mathcal{L}_{OPT,k,3} = \mathcal{L}_{OPT,k} \,.$$

We first upper bound the rewards of tasks in $\mathcal{L}_{OPT,k,1}$. Since between time $b_{k_{pre}} + \ell_{k_{pre}}$ and b_k the algorithm completes no task, this indicates that the algorithm keeps processing some tasks at those times but switches to other tasks before it is completed, and in the end, the algorithm switches to task k at $t = b_k$ and completes it. Denote u_t as the processing task by algorithm at time t and $\rho_t = r_{u_t}/(\ell_{u_t} - (t - b_{u_t}))$ as the remaining density of that processing task. By the algorithm's selecting rule, ρ_t is the maximum density of all arriving tasks at time t. Thus the total rewards in $\mathcal{L}_{OPT,k,1}$ is maximized by

$$\sum_{k' \in \mathcal{L}_{OPT,k,1}} r_{k'} \leq \sum_{t=b_{k_{pre}}+\ell_{k_{pre}}}^{b_k - 1} \rho_t \,.$$

Then it suffices to upper bound ρ_t at each time t. Since the algorithm does not complete any task at $t \in [b_{k_{pre}} + \ell_{k_{pre}}, b_k]$, the density ρ_t is increasing with time t.

We first consider the density ρ_t at $t = b_k - 1$. Recall that $u_{b_k - 1}$ is the processing task at time $t = b_k - 1$ and $\rho_t = r_{u_t}/(\ell_{u_t} - (t - b_{u_t}))$. Since the algorithm switches to k at time b_k, it indicates that $r_k/\ell_k > r_{u_t}/(\ell_{u_t} - (t - b_{u_t}) - 1)$. Thus $\rho_t = r_{u_t}/(\ell_{u_t} - (t - b_{u_t})) < \frac{r_k}{\ell_k} \frac{\ell_{u_t} - (t - b_{u_t}) - 1}{\ell_{u_t} - (t - b_{u_t})}$, which is maximized by setting $\ell_{u_t} = L_{\max}^{1/M}, b_{u_t} = t$, and thus $\rho_t \leq \frac{L_{\max}^{1/M} - 1}{L_{\max}^{1/M}} \frac{r_k}{\ell_k}$ at time $t = b_k - 1$.

Then we turn to consider the density ρ_t at $t = b_k - 2$. (1) If $u_{b_k-2} = u_{b_k-1}$, which means the algorithm does not switch at $t = b_k - 1$, and switches to task k at $t = b_k$. Thus we have $\frac{r_k}{\ell_k} > \frac{r_{u_{b_k-2}}}{\ell_{u_{b_k-2}} - (b_k - b_{u_{b_k-2}})}$. This is maximized by setting $b_{u_{b_k-2}} = b_k - 2, \ell_{u_{b_k-2}} = L_{\max}^{1/M}$ which implies $\rho_{b_k-2} \leq \frac{L_{\max}^{1/M}-2}{L_{\max}^{1/M}} \frac{r_k}{\ell_k}$. (2) If $u_{b_k-2} \neq u_{b_k-1}$, which means the algorithm switches to another task at $t = b_k - 1$. Then we have already know that $\rho_{b_k-1} \leq \frac{L_{\max}^{1/M}-1}{L_{\max}^{1/M}} \frac{r_k}{\ell_k}$, and similar analysis for $t = b_k - 2$ by setting $b_{u_{b_k-2}} = b_k - 2, \ell_{u_{b_k-2}} = L_{\max}^{1/M}$, the density is maximized by $\rho_{b_k-2} \leq (\frac{L_{\max}^{1/M}-1}{L_{\max}^{1/M}})^2 \frac{r_k}{\ell_k}$. Since $(\frac{L_{\max}^{1/M}-1}{L_{\max}^{1/M}})^2 > \frac{L_{\max}^{1/M}-2}{L_{\max}^{1/M}}$, ρ_{b_k-2} is maximized by $(\frac{L_{\max}^{1/M}-1}{L_{\max}^{1/M}})^2 \frac{r_k}{\ell_k}$.

We prove the following bound of ρ_t by induction. Suppose for any $i' \leq i$, $\rho_{b_k-i} \leq (\frac{L_{\max}^{1/M}-1}{L_{\max}^{1/M}})^i \frac{r_k}{\ell_k}$. Then consider time $t = b_k-(i+1)$, suppose the algorithm processes task u_t at $t = b_k-(i+1)$ and switches to another task at t' with $t < t' \leq b_k$. Since the maximum density at t' is $(\frac{L_{\max}^{1/M}-1}{L_{\max}^{1/M}})^{b_k-t'} \frac{r_k}{\ell_k}$. By the selecting rule we have that $\frac{r_{u_t}}{\ell_{u_t}-(t'-b_{u_t})} < (\frac{L_{\max}^{1/M}-1}{L_{\max}^{1/M}})^{b_k-t'} \frac{r_k}{\ell_k}$, which implies $\frac{r_{u_t}}{\ell_{u_t}-(t-b_{u_t})} < \frac{\ell_{u_t}-(t'-b_{u_t})}{\ell_{u_t}-(t-b_{u_t})} (\frac{L_{\max}^{1/M}-1}{L_{\max}^{1/M}})^{b_k-t'} \frac{r_k}{\ell_k}$. This is maximized by setting $b_{u_t} = t, \ell_{u_t} = L_{\max}^{i/M}$, and we have

$$\begin{aligned}
\rho_t \leq & \frac{L_{\max}^{1/M} - (t'-t)}{L_{\max}^{1/M}} \left(\frac{L_{\max}^{1/M}-1}{L_{\max}^{1/M}}\right)^{b_k-t'} \frac{r_k}{\ell_k} \\
= & \prod_{j=1}^{t'-t} \frac{L_{\max}^{1/M} - j}{L_{\max}^{1/M} - j + 1} \left(\frac{L_{\max}^{1/M}-1}{L_{\max}^{1/M}}\right)^{b_k-t} \frac{r_k}{\ell_k} \\
\leq & \left(\frac{L_{\max}^{1/M}-1}{L_{\max}^{1/M}}\right)^{t'-t} \left(\frac{L_{\max}^{1/M}-1}{L_{\max}^{1/M}}\right)^{b_k-t'} \frac{r_k}{\ell_k} \\
\leq & \left(\frac{L_{\max}^{1/M}-1}{L_{\max}^{1/M}}\right)^{b_k-t} \frac{r_k}{\ell_k} \\
= & \left(\frac{L_{\max}^{1/M}-1}{L_{\max}^{1/M}}\right)^{i+1} \frac{r_k}{\ell_k},
\end{aligned}$$

which is maximized by switching at time $t + 1$. Thus we can conclude that for each time $t \leq b_k$, $\rho_t \leq \left(\frac{L_{\max}^{1/M}-1}{L_{\max}^{1/M}}\right)^{b_k-t} \frac{r_k}{\ell_k}$. Thus

$$\sum_{t=b_{k_{pre}}+\ell_{k_{pre}}}^{b_k-1} \rho_t \leq \sum_{t=b_{k_{pre}}+\ell_{k_{pre}}}^{b_k-1} \left(\frac{L_{\max}^{1/M}-1}{L_{\max}^{1/M}}\right)^{b_k-t} \frac{r_k}{\ell_k} \leq L_{\max}^{1/M} \frac{r_k}{\ell_k}.$$

This implies that

$$r_k \geq \frac{\ell_k}{L_{\max}^{1/M}} \sum_{t=b_{k_{pre}}+\ell_{k_{pre}}}^{b_k-1} \rho_t \geq \frac{\ell_k}{L_{\max}^{1/M}} \sum_{k' \in \mathcal{L}_{OPT,k,1}} r_{k'}.$$

For optimal tasks in $\mathcal{L}_{OPT,k,2}$, the maximum total reward is obtained by setting optimal task k' with $b_{k'} = t, \ell_{k'} = L_{\max}^{(i-1)/M}, r_{k'} = \frac{r_k}{\ell_k-(t-b_k)} \ell_{k'}$ (the maximum reward one can set when no switch happens). Then we have

$$\sum_{k' \in \mathcal{L}_{OPT,k,2}} r_{k'} \leq \sum_{t=b_k}^{b_k+\ell_k-1} \frac{r_k}{\ell_k - (t-b_k)} \leq \ell_k r_k.$$

This indicates

$$r_k \geq \frac{1}{\log \ell_k} \sum_{k' \in \mathcal{L}_{OPT,k,2}} r_{k'}.$$

For optimal tasks in $\mathcal{L}_{OPT,k,3}$, we know that this set contains at most one task that is completed after $b_k + \ell_k$ (other tasks will be classified to the next task set). This optimal task k' is maximized by setting $b_{k'} = b_k + \ell_k - 1, \ell_{k'} = L_{\max}^{i/M}, r_{k'} = L_{\max}^{1/M} r_k$. And thus we have

$$r_k \geq \frac{1}{L_{\max}^{1/M}} r_{k'} \,.$$

Thus together we have

$$\begin{aligned} r_k \geq & \frac{\ell_k}{L_{\max}^{1/M}} \sum_{k' \in \mathcal{L}_{OPT,k,1}} r_{k'} \,, \\ r_k \geq & \frac{1}{\ell_k} \sum_{k' \in \mathcal{L}_{OPT,k,2}} r_{k'} \,, \\ r_k \geq & \frac{1}{L_{\max}^{1/M}} \sum_{k' \in \mathcal{L}_{OPT,k,3}} r_{k'} \,, \end{aligned}$$

which implies

$$r_k \geq \frac{1}{3L_{\max}^{1/M}} \sum_{k' \in \mathcal{L}_{OPT,k,1} \cup \mathcal{L}_{OPT,k,2} \cup \mathcal{L}_{OPT,k,3}} r_{k'} = \frac{1}{3L_{\max}^{1/M}} \sum_{k' \in \mathcal{L}_{OPT,k}} r_{k'} \,.$$

Since there are totally M machines, the sum of rewards in $\mathcal{L}_{OPT}$ needs to time M. Summing over all tasks $k \in \mathcal{L}_{ALG}$,

$$\sum_{k \in \mathcal{L}_{ALG}} r_k \geq \frac{1}{3ML_{\max}^{1/M}} \sum_{k \in \mathcal{L}_{ALG}} \sum_{k' \in \mathcal{L}_{OPT,k}} r_{k'} = \frac{1}{3ML_{\max}^{1/M}} \sum_{k' \in \mathcal{L}_{OPT}} r_{k'} \,.$$

The algorithm can achieve $1/(3ML_{\max}^{1/M})$ competitive ratio.

□

C ANALYSIS FOR S-UCB ALGORITHM

C.1 PROOF OF LEMMA 5.3

The regret caused by exploring the UCB index in S-UCB algorithm is bounded by

$$\mathbb{E}\left[\frac{1}{3ML_{\max}^{1/M}} \sum_{i=1}^{M} \sum_{k \in \mathcal{K}_{\text{ALG}}^i} \mathbb{1}\{\neg \mathcal{G}_k\} \Big(\sum_{k' \in \mathcal{N}_k} r_{k'} - r_k \Big)\right] \leq O\left(\sum_{n \in [N]} \frac{L_{\max}^{1/M} \log T}{\Delta_n} \right).$$

Proof. We decompose the exploration regret term by task type:

$$\begin{aligned} & \mathbb{E}\left[\frac{1}{3ML_{\max}^{1/M}} \sum_{i=1}^{M} \sum_{k \in \mathcal{K}_{\text{ALG}}^i} \mathbb{1}\{\neg \mathcal{G}_k\} \Big(\sum_{k' \in \mathcal{N}_k} r_{k'} - r_k \Big)\right] \\ \leq & \mathbb{E}\left[\frac{1}{3ML_{\max}^{1/M}} \sum_{i=1}^{M} \sum_{n=1}^{N} \sum_{k \in \mathcal{K}_{\text{ALG}}^i,\, n(k)=n} \mathbb{1}\{\neg \mathcal{G}_k\} \Big(\sum_{k' \in \mathcal{N}_k} r_{k'} - r_k \Big)\right]. \end{aligned}$$

We define the failure event $\mathcal{F} = \{\exists n \in [N], |\hat{\mu}_n(t) - \mu_n| > \sqrt{\frac{6\log(T)}{T_n(t)}}\}$. The probability of the failure event is bounded by the following lemma (same as before).

Lemma C.1. *Let $\mathcal{F} = \{\exists n \in [N], |\hat{\mu}_n(t) - \mu_n| > \sqrt{\frac{6\log(T)}{T_n(t)}}\}$ be the bad event that some tasks' rewards are not estimated well at time t, we have that*

$$\mathbb{P}(\mathcal{F}) \leq 2N/T\,.$$

Proof.

$$\begin{aligned}
\mathbb{P}(\mathcal{F}) &= \mathbb{P}\left(\exists 1 \leq t \leq T, n \in [N] : |\hat{\mu}_n(t) - \mu_n| > \sqrt{\frac{6\log T}{T_n(t)}}\right) \\
&\leq \sum_{t=1}^{T} \sum_{n\in[N]} \mathbb{P}\left(|\hat{\mu}_n(t) - \mu_n| > \sqrt{\frac{6\log T}{T_n(t)}}\right) \\
&\leq \sum_{t=1}^{T} \sum_{n\in[N]} \sum_{s=1}^{t} \mathbb{P}\left(T_n(t) = s, |\hat{\mu}_n(t) - \mu_n| > \sqrt{\frac{6\log T}{s}}\right) \\
&\leq \sum_{t=1}^{T} \sum_{n\in[N]} t \cdot 2\exp(-3\log T) \\
&\leq 2N/T\,,
\end{aligned}$$

where the second last inequality is derived from Hoeffding's inequality (since rewards are 1-subgaussian). □

Condition on $\neg\mathcal{F}$. Then for any type n we have $\mathrm{UCB}_n(t) \leq \mu_n + 2\sqrt{\frac{6\log T}{T_n(t)}}$. Importantly, when comparing *densities*, the confidence radius is divided by the task length.

Step 1: per-completed-task exploration charge. Fix a machine i and a completed task $k \in \mathcal{K}^i_{\mathrm{ALG}}$ with $n(k) = n$ and $\neg\mathcal{G}_k$. Let

$$k^\star \in \arg\max_{k' \in \mathcal{N}_k} \frac{\mu_{n(k')}}{\ell_{k'}}$$

be the nearby optimal task with the largest *true* density, and define the (positive) density gap

$$\Delta_k := \frac{\mu_{n(k^\star)}}{\ell_{k^\star}} - \frac{\mu_{n(k)}}{\ell_k} \;>\; 0.$$

Since $k^\star \in \mathcal{K}^i_{\mathrm{OPT}}$ and $k \in \mathcal{K}^i_{\mathrm{ALG}}$, both lengths lie in $(L_{\max}^{\frac{i-1}{M}}, L_{\max}^{\frac{i}{M}}]$, hence

$$\ell_{k^\star} \leq L_{\max}^{1/M}\, \ell_k. \tag{1}$$

By the pseudo-task construction in S-UCB, whenever $k^\star$ arrives while k is (eventually) completed, the algorithm must have selected a (pseudo)task with UCB density at least that of $k^\star$. In particular, this implies (on $\neg\mathcal{F}$)

$$\frac{\mathrm{UCB}_{n(k)}}{\ell_k} \;\geq\; \frac{\mathrm{UCB}_{n(k^\star)}}{\ell_{k^\star}} \;\geq\; \frac{\mu_{n(k^\star)}}{\ell_{k^\star}} \;=\; \frac{\mu_{n(k)}}{\ell_k} + \Delta_k, \tag{2}$$

where the second inequality uses optimism $\mathrm{UCB}_{n(k^\star)} \geq \mu_{n(k^\star)}$ under $\neg\mathcal{F}$. On the other hand, under $\neg\mathcal{F}$,

$$\frac{\mathrm{UCB}_{n(k)}}{\ell_k} \leq \frac{\mu_{n(k)}}{\ell_k} + \frac{2}{\ell_k}\sqrt{\frac{6\log T}{T_{n(k)}(t_k)}}, \tag{3}$$

where t_k is the completion time of k. Combining (2) and (3) yields

$$\Delta_k \leq \frac{2}{\ell_k}\sqrt{\frac{6\log T}{T_{n(k)}(t_k)}}. \tag{4}$$

As in the original proof, each such mis-ordering contributes regret at most $\Delta_k \ell_{k^\star}$. Using (1) and (4),

$$\Delta_k \ell_{k^\star} \leq \left(\frac{2}{\ell_k}\sqrt{\frac{6\log T}{T_{n(k)}(t_k)}}\right) \cdot (L_{\max}^{1/M}\ell_k) = 2L_{\max}^{1/M}\sqrt{\frac{6\log T}{T_{n(k)}(t_k)}}. \tag{5}$$

Step 2: sum type-by-type (removes the extra N factor). Fix a type $n \in [N]$ and consider the sequence of (bad) completed tasks of type n across all machines. Let S_n be the total number of such bad completions, and index them by completion order $s = 1, 2, \ldots, S_n$ so that at the s-th completion of type n we have $T_n(t) = s$. Summing (5) over these bad completions yields

$$\sum_{\substack{i\in[M],\, k\in\mathcal{K}^i_{\mathrm{ALG}} \\ n(k)=n,\, \neg\mathcal{G}_k}} \Delta_k \ell_{k^\star} \;\le\; 2L_{\max}^{1/M}\sqrt{6\log T}\sum_{s=1}^{S_n}\frac{1}{\sqrt{s}}.$$

We now bound S_n. For every bad completion of type n, we have $\Delta_k \ge \Delta_n$ by definition of Δ_n. Using (4) and $\ell_k \ge 1$,

$$\Delta_n \le \Delta_k \le 2\sqrt{\frac{6\log T}{T_n(t_k)}} \quad\Longrightarrow\quad T_n(t_k) \le \frac{24\log T}{\Delta_n^2}.$$

Hence $S_n \le \left\lfloor 24\log T/\Delta_n^2 \right\rfloor$. Using $\sum_{s=1}^{m} s^{-1/2} \le 2\sqrt{m}$,

$$\sum_{\substack{i\in[M],\, k\in\mathcal{K}^i_{\mathrm{ALG}} \\ n(k)=n,\, \neg\mathcal{G}_k}} \Delta_k \ell_{k^\star} \;\le\; 2L_{\max}^{1/M}\sqrt{6\log T}\cdot 2\sqrt{\frac{24\log T}{\Delta_n^2}} = \frac{48\, L_{\max}^{1/M}\log T}{\Delta_n}.$$

Summing over all $n \in [N]$ and adding the failure-event penalty,

$$\begin{aligned}
&\mathbb{E}\left[\frac{1}{3ML_{\max}^{1/M}}\sum_{i=1}^{M}\sum_{k\in\mathcal{K}^i_{\mathrm{ALG}}} \mathbb{1}\{\neg\mathcal{G}_k\}\Big(\sum_{k'\in\mathcal{N}_k} r_{k'} - r_k\Big)\right] \\
&\le \mathbb{E}\left[\cdot \mid \neg\mathcal{F}\right] + T\mathbb{P}\left(\mathcal{F}\right) \;\le\; \sum_{n=1}^{N}\frac{48\, L_{\max}^{1/M}\log T}{\Delta_n} \;+\; 2N \;\le\; O\left(\frac{NL_{\max}^{1/M}\log T}{\Delta}\right),
\end{aligned}$$

where the last inequality uses $\sum_{n=1}^{N} 1/\Delta_n \le N/\Delta$. This completes the proof. □

C.2 PROOF OF LEMMA 5.2

For completed task k, conditioned on event $\mathcal{G}_k$, it holds that $\forall k' \in \mathcal{N}_k, \mu_{n(k)}/\ell_k > \mu_{n(k')}/\ell_{k'}$.

We can define the optimal task set $\mathcal{L}_{OPT,k} = \{k' \in \mathcal{L}_{OPT} : b_{k-1} + \ell_{k-1} \le b_{k'} < b_k + \ell_k\}$ as the set of tasks in $\mathcal{L}_{OPT}$ such that it arrives after $k-1$ is completed and before k is completed. By this split it is easy to verify that the union of $\mathcal{L}_{OPT,k}$ covers all tasks in $\mathcal{L}_{OPT}$.

$$\bigcup_{k\in\mathcal{K}_{\mathrm{ALG}}} \mathcal{L}_{OPT,k} = \mathcal{L}_{OPT}\,.$$

Thus it suffices to analyze the reward relationship between r_k and $\sum_{k'\in\mathcal{L}_{OPT,k}} r_{k'}$ for each task $k \in \mathcal{L}_{ALG}$.

For the set $\mathcal{L}_{OPT,k}$, we further split it into three subsets: $\mathcal{L}_{OPT,k,1} = \{k' \in \mathcal{L}_{OPT,k} : b_{k'} + \ell_{k'} \le b_k\}$ is the set of optimal tasks completed before k arrives; $\mathcal{L}_{OPT,k,2} = \{k' \in \mathcal{L}_{OPT,k} : b_{k'} \ge b_k, b_{k'} + \ell_{k'} \le b_k + \ell_k\}$ is the set of optimal tasks arrives after k arrives and completed before k is completed; $\mathcal{L}_{OPT,k,3} = \{k' \in \mathcal{L}_{OPT,k} : b_{k'} + \ell_{k'} > b_k + \ell_k\}$ is the set of optimal tasks completed after k is completed. Since tasks in $\mathcal{L}_{OPT}$ are disjoint, it is easy to check that

$$\mathcal{L}_{OPT,k,1} \cup \mathcal{L}_{OPT,k,2} \cup \mathcal{L}_{OPT,k,3} = \mathcal{L}_{OPT,k}\,.$$

We first upper bound the rewards of tasks in $\mathcal{L}_{OPT,k,1}$ for S-UCB algorithm conditioned on $\mathcal{G}_k$. Since between time $b_{k-1} + \ell_{k-1}$ and b_k the algorithm completes no task, this indicates that the algorithm keeps processing some tasks at those times but switches to other tasks before it is completed, and in the end, the algorithm switches to task k at $t = b_k$ and completes it. Denote u_t as the processing task by algorithm at time t and $\rho_t = r_{u_t}/(\ell_{u_t} - (t - b_{u_t}))$ as the remaining

density of that processing task. By the algorithm's selecting rule, ρ_t is the maximum density of all arriving tasks at time t. Thus the total rewards in $\mathcal{L}_{OPT,k,1}$ is maximized by

$$\sum_{k' \in \mathcal{L}_{OPT,k,1}} r_{k'} \leq \sum_{t=b_{k-1}+\ell_{k-1}}^{b_k-1} \rho_t \,.$$

Then it suffices to upper bound ρ_t at each time t. Since the algorithm does not complete any task at $t \in [b_{k-1}+\ell_{k-1}, b_k]$, the density ρ_t is increasing with time t and k has the highest density by $\mathcal{G}_k$.

We first consider the density ρ_t at $t = b_k - 1$. Recall that u_{b_k-1} is the processing task at time $t = b_k - 1$ and $\rho_t = r_{u_t}/(\ell_{u_t} - (t - b_{u_t}))$. Since the algorithm switches to k at time b_k, it indicates that $r_k/\ell_k > r_{u_t}/(\ell_{u_t} - (t - b_{u_t}) - 1)$. Thus $\rho_t = r_{u_t}/(\ell_{u_t} - (t - b_{u_t})) < \frac{r_k}{\ell_k} \frac{\ell_{u_t}-(t-b_{u_t})-1}{\ell_{u_t}-(t-b_{u_t})}$, which is maximized by setting $\ell_{u_t} = L_{\max}^{1/M}, b_{u_t} = t$, and thus $\rho_t \leq \frac{L_{\max}^{1/M}-1}{L_{\max}^{1/M}} \frac{r_k}{\ell_k}$ at time $t = b_k - 1$.

Then we turn to consider the density ρ_t at $t = b_k - 2$. (1) If $u_{b_k-2} = u_{b_k-1}$, which means the algorithm does not switch at $t = b_k - 1$, and switches to task k at $t = b_k$. Thus we have $\frac{r_k}{\ell_k} > \frac{r_{u_{b_k-2}}}{\ell_{u_{b_k-2}}-(b_k-b_{u_{b_k-2}})}$. This is maximized by setting $b_{u_{b_k-2}} = b_k - 2, \ell_{u_{b_k-2}} = L_{\max}^{1/M}$ which implies $\rho_{b_k-2} \leq \frac{L_{\max}^{1/M}-2}{L_{\max}^{1/M}} \frac{r_k}{\ell_k}$. (2) If $u_{b_k-2} \neq u_{b_k-1}$, which means the algorithm switches to another task at $t = b_k - 1$. Then we have already know that $\rho_{b_k-1} \leq \frac{L_{\max}^{1/M}-1}{L_{\max}^{1/M}} \frac{r_k}{\ell_k}$, and similar analysis for $t = b_k - 2$ by setting $b_{u_{b_k-2}} = b_k - 2, \ell_{u_{b_k-2}} = L_{\max}^{1/M}$, the density is maximized by $\rho_{b_k-2} \leq (\frac{L_{\max}^{1/M}-1}{L_{\max}^{1/M}})^2 \frac{r_k}{\ell_k}$. Since $(\frac{L_{\max}^{1/M}-1}{L_{\max}^{1/M}})^2 > \frac{L_{\max}^{1/M}-2}{L_{\max}^{1/M}}$, ρ_{b_k-2} is maximized by $(\frac{L_{\max}^{1/M}-1}{L_{\max}^{1/M}})^2 \frac{r_k}{\ell_k}$.

We prove the following bound of ρ_t by induction. Suppose for any $i' \leq i$, $\rho_{b_k-i} \leq (\frac{L_{\max}^{1/M}-1}{L_{\max}^{1/M}})^i \frac{r_k}{\ell_k}$. Then consider time $t = b_k-(i+1)$, suppose the algorithm processes task u_t at $t = b_k-(i+1)$ and switches to another task at t' with $t < t' \leq b_k$. Since the maximum density at t' is $(\frac{L_{\max}^{1/M}-1}{L_{\max}^{1/M}})^{b_k-t'} \frac{r_k}{\ell_k}$. By the selecting rule we have that $\frac{r_{u_t}}{\ell_{u_t}-(t'-b_{u_t})} < (\frac{L_{\max}^{1/M}-1}{L_{\max}^{1/M}})^{b_k-t'} \frac{r_k}{\ell_k}$, which implies $\frac{r_{u_t}}{\ell_{u_t}-(t-b_{u_t})} < \frac{\ell_{u_t}-(t'-b_{u_t})}{\ell_{u_t}-(t-b_{u_t})} (\frac{L_{\max}^{1/M}-1}{L_{\max}^{1/M}})^{b_k-t'} \frac{r_k}{\ell_k}$. This is maximized by setting $b_{u_t} = t, \ell_{u_t} = L_{\max}^{i/M}$, and we have

$$\begin{aligned}
\rho_t \leq & \frac{L_{\max}^{1/M} - (t'-t)}{L_{\max}^{1/M}} \left(\frac{L_{\max}^{1/M}-1}{L_{\max}^{1/M}}\right)^{b_k-t'} \frac{r_k}{\ell_k} \\
= & \prod_{j=1}^{t'-t} \frac{L_{\max}^{1/M} - j}{L_{\max}^{1/M} - j + 1} \left(\frac{L_{\max}^{1/M}-1}{L_{\max}^{1/M}}\right)^{b_k-t} \frac{r_k}{\ell_k} \\
\leq & \left(\frac{L_{\max}^{1/M}-1}{L_{\max}^{1/M}}\right)^{t'-t} \left(\frac{L_{\max}^{1/M}-1}{L_{\max}^{1/M}}\right)^{b_k-t'} \frac{r_k}{\ell_k} \\
\leq & \left(\frac{L_{\max}^{1/M}-1}{L_{\max}^{1/M}}\right)^{b_k-t} \frac{r_k}{\ell_k} \\
= & \left(\frac{L_{\max}^{1/M}-1}{L_{\max}^{1/M}}\right)^{i+1} \frac{r_k}{\ell_k} \,,
\end{aligned}$$

which is maximized by switching at time $t+1$. Thus we can conclude that for each time $t \leq b_k$, $\rho_t \leq \left(\frac{L_{\max}^{1/M}-1}{L_{\max}^{1/M}}\right)^{b_k-t} \frac{r_k}{\ell_k}$. Thus

$$\sum_{t=b_{k_{pre}}+\ell_{k_{pre}}}^{b_k-1} \rho_t \leq \sum_{t=b_{k_{pre}}+\ell_{k_{pre}}}^{b_k-1} \left(\frac{L_{\max}^{1/M}-1}{L_{\max}^{1/M}}\right)^{b_k-t} \frac{r_k}{\ell_k} \leq L_{\max}^{1/M} \frac{r_k}{\ell_k} \,.$$

This implies that

$$r_k \geq \frac{\ell_k}{L_{\max}^{1/M}} \sum_{t=b_{k_{pre}}+\ell_{k_{pre}}}^{b_k-1} \rho_t \geq \frac{\ell_k}{L_{\max}^{1/M}} \sum_{k' \in \mathcal{L}_{OPT,k,1}} r_{k'} .$$

For optimal tasks in $\mathcal{L}_{OPT,k,2}$, the maximum total reward is obtained by setting optimal task k' with $b_{k'} = t, \ell_{k'} = L_{\max}^{(i-1)/M}, r_{k'} = \frac{r_k}{\ell_k-(t-b_k)}\ell_{k'}$ (the maximum reward one can set when no switch happens). Then we have

$$\sum_{k' \in \mathcal{L}_{OPT,k,2}} r_{k'} \leq \sum_{t=b_k}^{b_k+\ell_k-1} \frac{r_k}{\ell_k-(t-b_k)} \leq \ell_k r_k .$$

This indicates

$$r_k \geq \frac{1}{\log \ell_k} \sum_{k' \in \mathcal{L}_{OPT,k,2}} r_{k'} .$$

For optimal tasks in $\mathcal{L}_{OPT,k,3}$, we know that this set contains at most one task that is completed after $b_k + \ell_k$ (other tasks will be classified to the next task set). This optimal task k' is maximized by setting $b_{k'} = b_k + \ell_k - 1, \ell_{k'} = L_{\max}^{i/M}, r_{k'} = L_{\max}^{1/M} r_k$. And thus we have

$$r_k \geq \frac{1}{L_{\max}^{1/M}} r_{k'} .$$

Thus together we have

$$\begin{aligned}
r_k \geq & \frac{\ell_k}{L_{\max}^{1/M}} \sum_{k' \in \mathcal{L}_{OPT,k,1}} r_{k'} , \\
r_k \geq & \frac{1}{\ell_k} \sum_{k' \in \mathcal{L}_{OPT,k,2}} r_{k'} , \\
r_k \geq & \frac{1}{L_{\max}^{1/M}} \sum_{k' \in \mathcal{L}_{OPT,k,3}} r_{k'} ,
\end{aligned}$$

which implies

$$r_k \geq \frac{1}{3L_{\max}^{1/M}} \sum_{k' \in \mathcal{L}_{OPT,k,1} \cup \mathcal{L}_{OPT,k,2} \cup \mathcal{L}_{OPT,k,3}} r_{k'} = \frac{1}{3L_{\max}^{1/M}} \sum_{k' \in \mathcal{L}_{OPT,k}} r_{k'} .$$

Since there are totally M machines, the sum of rewards in $\mathcal{L}_{OPT}$ needs to time M. Summing over all tasks $k \in \mathcal{L}_{ALG}$,

$$\sum_{k \in \mathcal{L}_{ALG}} r_k \geq \frac{1}{3ML_{\max}^{1/M}} \sum_{k \in \mathcal{L}_{ALG}} \sum_{k' \in \mathcal{L}_{OPT,k}} r_{k'} = \frac{1}{3ML_{\max}^{1/M}} \sum_{k' \in \mathcal{L}_{OPT}} r_{k'} .$$

Thus the S-UCB algorithm can achieve $1/(3ML_{\max}^{1/M})$ competitive ratio conditioned on $\mathcal{G}_k$, implying 0 approximate regret.

C.3 PROOF OF THEOREM 5.6

Proof. We follow the proof template of Theorem 5.1 and Lemmas 5.2–5.3, but replace the gap-dependent final step by a standard gap-free conversion.

Step 1: Concentration event. For each type $n \in [N]$ and each integer $s \geq 1$, let $\hat{\mu}_{n,s}$ be the empirical mean of the first s *completed* rewards of type n on a given machine (equivalently, the s-th update used in Algorithm 2), and define the good event

$$\mathcal{E} := \left\{ \forall n \in [N],\ \forall s \in \{1, \dots, T\} :\ \left|\hat{\mu}_{n,s} - \mu_n\right| \leq \sqrt{\frac{6 \log T}{s}} \right\}.$$

By 1-subgaussian concentration and a union bound over (n, s) (as in Lemma C.1),

$$\Pr(\mathcal{E}^c) \leq \frac{2N}{T}. \tag{6}$$

Moreover, on $\mathcal{E}$, for any type n with $T_n(t) \geq 1$ (number of *completed* type-n tasks up to time t), the UCB index used by S-UCB satisfies

$$\mu_n \ \leq\ UCB_n(t) \ =\ \hat{\mu}_{n,T_n(t)} + \sqrt{\frac{6 \log T}{T_n(t)}} \ \leq\ \mu_n + 2\sqrt{\frac{6 \log T}{T_n(t)}}. \tag{7}$$

Step 2: Regret decomposition. Using the same decomposition as in Theorem 6.1, write the approximate regret as a sum over completed tasks and split by the correctness event G_k (the event that the completed task k has the largest *true* density among the nearby optimal tasks N_k):

$$Reg(T) = E\left[\sum_{i=1}^{M} \sum_{k \in K_i^{ALG}} \frac{1}{3ML_{\max}^{1/M}} \Big(\sum_{k' \in N_k} r_{k'} - r_k \Big)\right] = (\mathsf{I}) + (\mathsf{II}),$$

where (I) collects the terms multiplied by $\mathbb{1}\{G_k\}$ and (II) collects the terms multiplied by $\mathbb{1}\{\neg G_k\}$.

Exactly as in Lemma 6.2, conditioned on G_k the algorithm behaves like MRDF with correctly ordered densities within each window, hence achieves the $\frac{1}{3ML_{\max}^{1/M}}$ competitive baseline on that window; therefore

$$(\mathsf{I}) = 0. \tag{8}$$

It remains to upper bound (II), the regret caused by exploration / mis-ordering of densities.

Step 3: A per-comparison bound (gap-dependent $\Rightarrow$ gap-free). Fix any machine i, and consider any completed task $k \in K_i^{ALG}$ with type $n := n(k)$ and length $\ell := \ell_k$. On $\neg G_k$, there exists some $k' \in N_k$ with type $n' := n(k')$ and length $\ell' := \ell_{k'}$ such that

$$\frac{\mu_{n'}}{\ell'} \ >\ \frac{\mu_n}{\ell}.$$

Define the (positive) density gap for this comparison

$$\Delta_{k,k'} \ :=\ \frac{\mu_{n'}}{\ell'} - \frac{\mu_n}{\ell} \ >\ 0.$$

The same UCB argument as in Lemma 6.3 shows that, on $\mathcal{E}$, such a wrong ordering can only persist while the completion count $T_n(t)$ is small: if $T_n(t) > \frac{24 \log T}{\Delta_{k,k'}^2}$, then by (7),

$$\frac{UCB_n(t)}{\ell} \ \leq\ \frac{\mu_n}{\ell} + 2\sqrt{\frac{6 \log T}{T_n(t)}} \ \leq\ \frac{\mu_n}{\ell} + \Delta_{k,k'} \ =\ \frac{\mu_{n'}}{\ell'} \ \leq\ \frac{UCB_{n'}(t)}{\ell'},$$

contradicting that S-UCB selects a task of type n over the better-density alternative. Hence, on $\mathcal{E}$, the number of *completed* tasks that can be charged to this specific mis-ordering is at most $\frac{24 \log T}{\Delta_{k,k'}^2}$. Trivially, it is also at most T (a single machine can complete at most one task per round), so we may write the bound as

$$\#\{\text{charged completions for this comparison}\} \ \leq\ \min\left\{ \frac{24 \log T}{\Delta_{k,k'}^2},\ T \right\}. \tag{9}$$

As in Lemma 6.3, each such completion contributes at most $\Delta_{k,k'} \cdot \ell' \le \Delta_{k,k'} L_{\max}$ to the (additive) regret term inside (II).[1] Combining with (9), the regret contribution of this comparison is bounded by

$$L_{\max} \cdot \Delta_{k,k'} \cdot \min\left\{ \frac{24 \log T}{\Delta_{k,k'}^2},\ T \right\} \ = \ L_{\max} \cdot \min\left\{ \frac{24 \log T}{\Delta_{k,k'}},\ \Delta_{k,k'} T \right\}.$$

Finally, apply the elementary inequality $\min\{a/x, bx\} \le 2\sqrt{ab}$ for all $a, b, x > 0$ with $a = 24 \log T$ and $b = T$ to obtain the *gap-free* bound

$$L_{\max}^{1/M} \cdot \min\left\{ \frac{24 \log T}{\Delta_{k,k'}},\ \Delta_{k,k'} T \right\} \ \le \ 2\sqrt{24}\, L_{\max}^{1/M} \sqrt{T \log T}. \tag{10}$$

Step 4: Summing over types and machines, plus the bad event. Summing (10) over all possible mis-ordering pairs of types (at most $N(N-1) \le N^2$ pairs), and using (8), yields on $\mathcal{E}$:

$$Reg(T) \ \le \ 2\sqrt{24}\, N L_{\max}^{1/M} \sqrt{T \log T}.$$

Accounting for $\mathcal{E}^c$ via the same crude bound as in Lemma 6.3 gives an additive term $T \Pr(\mathcal{E}^c) \le 2N$ by (6). Therefore,

$$Reg(T) \ \le \ 2N + 2\sqrt{24}\, N L_{\max}^{1/M} \sqrt{T \log T},$$

which completes the proof. □

D PROOF OF THEOREM 5.4

Theorem D.1. *Given instance I, following the S-UCB algorithm (Algorithm 2), the scheduler can achieve the expected competitive ratio with*

$$\mathbb{E}[R] \ge \frac{1}{3M L_{\max}^{1/M} + N^2 L_{\max}^{1/M} \log T / (R_{ALG}(I)\Delta)},$$

where $R_{ALG}(I)$ is the total completion rewards obtained by S-UCB, and expectation is taken over the randomness of reward distribution.

Proof.

$$\begin{aligned}
\mathbb{E}[R] =& \mathbb{E}\left[\frac{R_{\text{ALG}}}{R_{\text{OPT}}} \right] \\
=& \mathbb{E}\left[\frac{\sum_{k \in \mathcal{K}_{\text{ALG}}} r_k}{\sum_{k \in \mathcal{K}_{\text{OPT}}} r_k} \right] \\
=& \mathbb{E}\left[\frac{\sum_{k \in \mathcal{K}_{\text{ALG}}} \mathbb{1}\{\mathcal{G}_k\}\, r_k + \sum_{k \in \mathcal{K}_{\text{ALG}}} \mathbb{1}\{\neg \mathcal{G}_k\}\, r_k}{\sum_{k \in \mathcal{K}_{\text{ALG}}} \mathbb{1}\{\mathcal{G}_k\} \sum_{k' \in \mathcal{N}_k} r_{k'} + \sum_{k \in \mathcal{K}_{\text{ALG}}} \mathbb{1}\{\neg \mathcal{G}_k\} \sum_{k' \in \mathcal{N}_k} r_{k'}} \right] \\
\ge& \mathbb{E}\left[\frac{\sum_{k \in \mathcal{K}_{\text{ALG}}} \mathbb{1}\{\mathcal{G}_k\}\, r_k + \sum_{k \in \mathcal{K}_{\text{ALG}}} \mathbb{1}\{\neg \mathcal{G}_k\}\, r_k}{3M L_{\max}^{1/M} \sum_{k \in \mathcal{K}_{\text{ALG}}} \mathbb{1}\{\mathcal{G}_k\}\, r_k + 3M L_{\max}^{1/M} \sum_{k \in \mathcal{K}_{\text{ALG}}} \mathbb{1}\{\neg \mathcal{G}_k\}\, r_k + 24 N^2 L_{\max} \log T / \Delta} \right] \\
=& \mathbb{E}\left[\frac{1}{3M L_{\max}^{1/M} + 24 N^2 L_{\max} \log T / \Delta R_{\text{ALG}}(I)} \right].
\end{aligned}$$

□

[1] This is the same per-completion charging used in the gap-dependent proof: the regret is measured in reward units, and the worst-case conversion from density gap to reward loss uses $\ell' \le L_{\max}$.

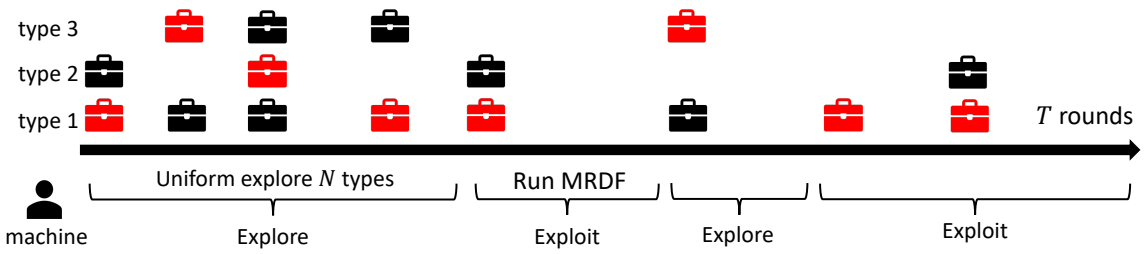

Figure 3: Illustration of the Explore-then-Schedule algorithm. During the explore stage, the algorithm samples task types to shrink confidence intervals. Once the best density is statistically separated, it switches to the exploit stage and runs MRDF with fixed empirical means.

E EXPLORE THEN SCHEDULING ALGORITHM

The key idea to balance exploration and exploitation is that the algorithm always performs MRDF algorithm (Algorithm 1) when it is confident to determine the maximum remaining density of the coming tasks. This confidence is computed by constructing an upper confidence bound and a lower confidence bound for each task's mean reward estimation. And the algorithm will enter the exploration stage if it is still not confident to perform the MRDF algorithm. This motivates us to design the following Explore then Scheduling algorithm (Algorithm 3).

The algorithm first initializes $\text{UCB}_n, \text{LCB}_n, T_n$ for each type $n \in [N]$, where $\text{UCB}_n, \text{LCB}_n$ are the upper confidence bound (UCB) index and lower confidence bound (LCB) index for type n, respectively. T_n is the number of times task n is completed. Denote $\mathcal{M}$ as the task set the machine is processing and it is empty at the beginning. Define FLAG as the sign of the current algorithm's stage, and it is set as "Exploit" stage at first.

At time t, if the current processing task $k \in \mathcal{M}$ is completed, i.e., $b_k + \ell_k = t$, then the algorithm will update the corresponding $\text{UCB}_n, \text{LCB}_n, T_n$ for type $n = n(k)$ (Line 4). Specifically, $T_n = T_n + 1$, the corresponding upper confidence bound $\text{UCB}_n = \hat{\mu}_n + \sqrt{\frac{6 \log T}{T_n}}$, and lower confidence bound $\text{LCB}_n = \hat{\mu}_n - \sqrt{\frac{6 \log T}{T_n}}$, where $\hat{\mu}_n$ is the empirical reward mean of task n. After that, the algorithm resets the FLAG to the Exploit stage (Lines 5 - 9).

Then tasks set $\mathcal{L}_t$ comes and the algorithm observes b_k, ℓ_k and $n(k)$ for each coming task $k \in \mathcal{L}_t$ (Line 10). If the current algorithm's stage FLAG = Explore, then it means the algorithm needs to explore the current processing task $k \in \mathcal{M}$, and thus the algorithm will do nothing at time t (Line 11 - 12). When current stage FLAG = Exploit, similarly with MRDF (Algorithm 1), the algorithm first constructs a pseudo task k' for $k \in \mathcal{M}$ with $b_{k'} = t$, $\ell_{k'} = \ell_k - (t - b_k)$, $n(k') = n(k)$, and add this pseudo task k' to $\mathcal{L}_t$ (Lines 15 - 18.) Then the algorithm checks if the maximum r/ℓ can be distinguished, i.e., $\exists k, \forall k' \neq k, \frac{\text{LCB}_{n(k)}}{\ell_k} > \frac{\text{UCB}_{n(k')}}{\ell_{k'}}$. This indicates that task k has a maximum r/ℓ with high probability, and thus the algorithm runs the same process as the online algorithm with the known mean reward (Lines 19 - 21). Otherwise there exists an overlap between the largest LCB index and another UCB index, the algorithm will turn to the Explore stage and select the task with minimum $T_{n(k)}$ to guarantee the uniform exploration over all types (Lines 22 - 26).

The Explore-then-Schedule algorithm alternates between two regimes. In the explore regime, it keeps sampling insufficiently observed task types until the confidence intervals separate the best empirical density from all competitors. In the exploit regime, it runs MRDF with the fixed empirical means. This separation makes ETS easy to analyze, but it is less sample efficient than S-UCB because it delays reward maximization until uniform exploration is sufficiently complete.

Theorem E.1. *When the learner follows the Online then Scheduling algorithm (Algorithm 3), the learner can obtain the* $\frac{1}{3ML_{\max}^{1/M}}$*-approximate regret bounded by*

$$\text{Reg}(T) \leq \frac{24NL_{\max} \log T}{\Delta^2} + 2N\,,$$

where $\Delta = \min_{n \neq n', \ell, \ell'} |\frac{\mu_n}{\ell} - \frac{\mu_{n'}}{\ell'}|$ *is the minimum density gap of two tasks.*

Denote $r(t)$ as the reward the algorithm gets at time t, and $r_{\text{OPT}}(t)$ is the reward the compared optimal scheduling strategy

Algorithm 3 Explore then Scheduling

Initialize: $\text{UCB}_n = \infty, \text{LCB}_n = -\infty, T_n = 0, \forall n \in [N]$, FLAG = Exploit, $\mathcal{M} = \emptyset$;
for $t = 1, 2, \cdots$ **do**
  if $\mathcal{M} \neq \emptyset$ and $b_k + \ell_k = t, k \in \mathcal{M}$ **then**
    Receive reward r_k, update $\text{UCB}_{n(k)}, \text{LCB}_{n(k)}$, and $T_{n(k)}$;
    $\mathcal{M} = \emptyset$;
    if FLAG = Explore **then**
      FLAG = Exploit;
    end if
  end if
  Receive coming tasks set $\mathcal{L}_t = \{k \mid b_k = t\}$, observe $b_k, \ell_k, n(k)$ for each $k \in \mathcal{L}_t$;
  if FLAG = Explore **then**
    Continue processing current task $k \in \mathcal{M}$;
  end if
  if FLAG = Exploit **then**
    if $\mathcal{M} \neq \emptyset$ **then**
      For $k \in \mathcal{M}$, construct pseudo task k' with $b_{k'} = t, \ell_{k'} = \ell_k - (t - b_k), n(k') = n(k)$;
      $\mathcal{L}_t = \mathcal{L}_t \cup \{k'\}$;
    end if
    Find task $k \in \arg\max_{k' \in \mathcal{L}_t} \frac{\text{LCB}_{n(k')}}{\ell_{k'}}$;
    if $\frac{\text{LCB}_{n(k)}}{\ell_k} > \max_{k' \in \mathcal{L}_t \setminus \{k\}} \frac{\text{UCB}_{n(k')}}{\ell_{k'}}$ **then**
      $\mathcal{M} = \{k\}$;
    else
      FLAG = Explore;
      Find $k' \in \arg\min_{k'' \in \mathcal{L}_t} T_{n(k'')}$;
      $\mathcal{M} = \{k'\}$;
    end if
  end if
end for

OPT gets at time t. Then the regret can be rewritten as

$$\begin{aligned}
&\text{Reg}(T) \\
=&\mathbb{E}\left[\frac{1}{3L_{\max}}\sum_{t=1}^{T} r_{\text{OPT}}(t) - \sum_{t=1}^{T} r(t)\right] \\
=&\mathbb{E}\left[\sum_{t=1}^{T} \mathbb{1}\{\text{FLAG}_t = \text{Exploit}\}\left(\frac{1}{3L_{\max}} r_{\text{OPT}}(t) - r(t)\right)\right] \\
+&\mathbb{E}\left[\sum_{t=1}^{T} \mathbb{1}\{\text{FLAG}_t = \text{Explore}\}\left(\frac{1}{3L_{\max}} r_{\text{OPT}}(t) - r(t)\right)\right],
\end{aligned}$$

where FLAG_t is the FLAG at time t performed by Algorithm 3. The regret is decomposed by two terms: the first is the regret caused by the exploitation stage, and the second term is the regret caused by exploration stage.

To bound the regret caused by the exploitation stage, the following two lemmas state that with high probability, the algorithm performs the same as the MRDF algorithm (Algorithm 3) with known mean reward, which attains $1/(3L_{\max})$ competitive ratio. Thus the regret can be bounded by a constant independent of T.

Lemma E.2. *Let* $\mathcal{F} = \{\exists i \in [N], |\hat{\mu}_i(t) - \mu_i| > \sqrt{\frac{6\log(T)}{T_i(t)}}\}$ *be the bad event that some tasks' rewards are not estimated well at time t, we have that*

$$\mathbb{P}(\mathcal{F}) \leq 2N/T\,.$$

Proof.

$$\begin{aligned}
\mathbb{P}(\mathcal{F}) &= \mathbb{P}\left(\exists 1 \le t \le T, i \in [N] : |\hat{\mu}_i(t) - \mu_i| > \sqrt{\frac{6\log T}{T_i(t)}}\right) \\
&\le \sum_{t=1}^{T} \sum_{i\in[N]} \mathbb{P}\left(|\hat{\mu}_i(t) - \mu_i| > \sqrt{\frac{6\log T}{T_i(t)}}\right) \\
&\le \sum_{t=1}^{T} \sum_{i\in[N]} \sum_{s=1}^{t} \mathbb{P}\left(T_i(t) = s, |\hat{\mu}_i(t) - \mu_i| > \sqrt{\frac{6\log T}{s}}\right) \\
&\le \sum_{t=1}^{T} \sum_{i\in[N]} t \cdot 2\exp(-3\log T) \\
&\le 2N/T\,,
\end{aligned}$$

where the second last inequality is derived from the Hoeffding inequality. □

Lemma E.3. *When* $\exists k, \forall k' \neq k, \frac{LCB_{n(k)}(t)}{\ell_k} > \frac{UCB_{n(k')}(t)}{\ell_{k'}}$*, under event* $\neg\mathcal{F}$*, the algorithm proceeds the same as the online algorithm with known mean reward* μ_n*.*

Proof. When $\exists k, \forall k' \neq k, \frac{\mathrm{LCB}_{n(k)}(t)}{\ell_k} > \frac{\mathrm{UCB}_{n(k')}(t)}{\ell_{k'}}$, under event $\neg\mathcal{F}$, it indicates that $\exists k, \forall k' \neq k, \frac{\mu_{n(k)}(t)}{\ell_k} > \frac{\mu_{n(k')}(t)}{\ell_{k'}}$. Thus the task with the maximum remaining density is exactly k, and the algorithm proceeds the same as MRDF. □

This indicates that with high probability the algorithm achieves the competitive ratio larger than $1/(3L_{\max})$, and the regret is caused by the low probability of event $\mathcal{F}$.

$$\begin{aligned}
&\mathbb{E}\left[\sum_{t=1}^{T} \mathbb{1}\{\mathrm{FLAG}_t = \mathrm{Exploit}\}\left(\frac{1}{3L_{\max}} r_{\mathrm{OPT}}(t) - r(t)\right)\right] \\
\le& T\mathbb{P}(\mathcal{F}) \le 2N\,.
\end{aligned}$$

Then we turn to bound the second term, which is the regret caused by exploration stage. This term is upper bounded by the maximum reward per task times the number of explorations for each type of task. By algorithm design we have that the learner always explores the task with the minimum number of selections (Line 24 in Algorithm 3). This guarantees that all types of tasks are explored in a round-robin manner. The following lemma upper bounds the exploration times for each task.

Lemma E.4. *The number of times the algorithm is exploring is bounded by* $24NL_{\max}\log T/\Delta^2$*.*

Proof. Given two types n, n', we have that conditioned on $\neg\mathcal{F}$, when $T_n(t) \ge 24\log T/\Delta^2$ and $T_{n'}(t) \ge 24\log T/\Delta^2$, then $\forall \ell, \ell'$, $\frac{\mathrm{LCB}_n(t)}{\ell} > \frac{\mathrm{UCB}_{n'}(t)}{\ell'}$ or $\frac{\mathrm{UCB}_n(t)}{\ell} < \frac{\mathrm{LCB}_{n'}(t)}{\ell'}$. By the algorithm design, the learner will always explore the task with the minimum $T_n(t)$, which means each task will reach $\log T/\Delta^2$ uniformly. To explore one task, the leaner will cost at most $L_{\max}$ rounds. Thus the number of times the algorithm is exploring is bounded by $24NL_{\max}\log T/\Delta^2$. □

This indicates that

$$\begin{aligned}
&\mathbb{E}\left[\sum_{t=1}^{T} \mathbb{1}\{\mathrm{FLAG}_t = \mathrm{Explore}\}\left(\frac{1}{3L_{\max}} r_{\mathrm{OPT}}(t) - r(t)\right)\right] \\
\le& \frac{24NL_{\max}\log T}{\Delta^2}\,.
\end{aligned}$$

Then summing over these two terms, theorem is derived.

Theorem E.5 (Improved gap-dependent approximate regret of S-UCB)**.** *Assume the stochastic reward model: there are N task types, and each type $n \in [N]$ has an unknown reward distribution D_n that is 1-subgaussian with mean μ_n. Let $L_{\max}$ be the maximum processing length and let*

$$R \;:=\; L_{\max}^{1/M}.$$

Run S-UCB *(Algorithm 2) for horizon $T \geq 2$ and define the $\frac{1}{3MR}$-approximate regret as in the main paper:*

$$Reg(T) := E\left[\frac{1}{3MR}\, R_{OPT}(T) - R_{ALG}(T)\right].$$

For each type n, define its minimum positive *density gap*

$$\Delta_n \;:=\; \min_{\substack{n' \neq n,\ \ell,\ell' \\ \mu_{n'}/\ell - \mu_n/\ell' > 0}} \left(\frac{\mu_{n'}}{\ell} - \frac{\mu_n}{\ell'}\right), \qquad \Delta := \min_{n \in [N]} \Delta_n.$$

Then S-UCB *satisfies the gap-dependent bound*

$$Reg(T) \;\leq\; 48\, R \log T \cdot \sum_{n=1}^{N} \frac{1}{\Delta_n} \;+\; 2N \;\leq\; \frac{48\, N\, R \log T}{\Delta} \;+\; 2N \;=\; O\left(\frac{N\, L_{\max}^{1/M} \log T}{\Delta}\right).$$

Proof. We follow the regret decomposition in the proof of Theorem 6.1, and we only strengthen the exploration analysis (Appendix C.2) by (i) keeping the $1/\ell$ factor in the UCB-density confidence term and (ii) using the within-class length ratio $R = L_{\max}^{1/M}$.

Step 0: notation (same as Theorem 6.1). Let K_i^{ALG} be the set of tasks completed by machine $i \in [M]$ under S-UCB. Let K_i^{OPT} be the set of tasks completed by OPT whose lengths lie in the i-th length interval $\left(L_{\max}^{(i-1)/M},\, L_{\max}^{i/M}\right]$. For each completed task $k \in K_i^{\mathrm{ALG}}$, let k^- be the nearest completed task before k (on machine i), and define the nearby optimal set

$$N_k := \Big\{k' \in K_i^{\mathrm{OPT}} :\; b_{k^-} + \ell_{k^-} \leq b_{k'} < b_k + \ell_k\Big\}.$$

Define the event

$$G_k := \Big\{\forall k' \in N_k :\; \mu_{n(k)}/\ell_k \geq \mu_{n(k')}/\ell_{k'}\Big\}.$$

As shown in Appendix C.1, conditioned on G_k (i.e., the completed task has the largest true density among N_k), the same charging argument as MRDF implies 0 approximate regret on that window. Hence, the approximate regret is entirely due to the exploration/mis-ordering events $\neg G_k$. (Formally, one can repeat the decomposition in Theorem 6.1 and use the Appendix C.1 argument to drop the G_k-part.)

Step 1: a high-probability confidence event. Let $T_n(t)$ be the number of *completed* tasks of type n up to time t (across all machines), and $\hat{\mu}_n(t)$ the corresponding empirical mean. Define the bad event

$$F := \left\{\exists t \in [T], \exists n \in [N] :\; \left|\hat{\mu}_n(t) - \mu_n\right| > \sqrt{\frac{6 \log T}{T_n(t)}}\right\}.$$

By a standard union bound with subgaussian concentration (as in Lemma C.1),

$$\Pr(F) \;\leq\; \frac{2N}{T}. \tag{11}$$

On $\neg F$, for every type n with $T_n(t) \geq 1$,

$$\mu_n \;\leq\; \mathrm{UCB}_n(t) \;=\; \hat{\mu}_n(t) + \sqrt{\frac{6 \log T}{T_n(t)}} \;\leq\; \mu_n + 2\sqrt{\frac{6 \log T}{T_n(t)}}. \tag{12}$$

(If $T_n(t) = 0$, S-UCB initializes $\mathrm{UCB}_n(t) = +\infty$, which is also optimistic.)

Step 2: reduce regret to the exploration term under $\neg F$. Let $Reg_{\exp}(T)$ denote the exploration part (the $\neg G_k$ part) of the decomposition in Theorem 6.1:

$$Reg_{\exp}(T) := E\left[\frac{1}{3MR}\sum_{i=1}^{M}\sum_{k\in K_i^{\mathrm{ALG}}}\mathbf{1}\{\neg G_k\}\left(\sum_{k'\in N_k} r_{k'} - r_k\right)\right].$$

Since $\frac{1}{3MR} \le 1$ and $r_k \ge 0$, we can upper bound

$$Reg_{\exp}(T) \;\le\; E\left[\sum_{i=1}^{M}\sum_{k\in K_i^{\mathrm{ALG}}}\mathbf{1}\{\neg G_k\}\left(\sum_{k'\in N_k} r_{k'} - r_k\right)\right]. \tag{13}$$

Thus

$$Reg(T) \;\le\; Reg_{\exp}(T) \;+\; T\Pr(F),$$

where the additive term $T\Pr(F)$ is the same crude control used in Appendix C.2.

It remains to bound $Reg_{\exp}(T)$ on $\neg F$.

Step 3: a per-completed-task exploration charge. Fix a machine i and a completed task $k \in K_i^{\mathrm{ALG}}$ with $\neg G_k$. Let $k^\star \in \arg\max_{k'\in N_k} \mu_{n(k')}/\ell_{k'}$ be the optimal task in N_k with maximum true density. Define the (positive) density gap

$$\Delta_k := \frac{\mu_{n(k^\star)}}{\ell_{k^\star}} - \frac{\mu_{n(k)}}{\ell_k} \;>\; 0.$$

As in Appendix C.2, the regret contribution of this mis-ordering can be upper bounded by $\Delta_k \ell_{k^\star}$ (interpreted as "density gap $\times$ a relevant time scale").

Now use the key facts:

- (Length-class ratio) Since both k and $k^\star$ lie in the same machine-i length interval $\left(L_{\max}^{(i-1)/M},\, L_{\max}^{i/M}\right]$, we have

$$\ell_{k^\star} \;\le\; \frac{L_{\max}^{i/M}}{L_{\max}^{(i-1)/M}}\ell_k \;=\; R\,\ell_k. \tag{14}$$

- (UCB mis-order implies a confidence-radius lower bound) Since S-UCB completes k without being preempted by $k^\star$ within the window, the (pseudo-)task representation of k must have had UCB density at least that of $k^\star$ whenever $k^\star$ arrived. In particular, it implies

$$\frac{\mathrm{UCB}_{n(k)}}{\ell_k} \;\ge\; \frac{\mathrm{UCB}_{n(k^\star)}}{\ell_{k^\star}} \;\ge\; \frac{\mu_{n(k^\star)}}{\ell_{k^\star}} \;=\; \frac{\mu_{n(k)}}{\ell_k} + \Delta_k,$$

  where the second inequality uses $\mathrm{UCB}_{n(k^\star)} \ge \mu_{n(k^\star)}$ under $\neg F$ (cf. (12)). On the other hand, by (12),

$$\frac{\mathrm{UCB}_{n(k)}}{\ell_k} \;\le\; \frac{\mu_{n(k)}}{\ell_k} + \frac{2}{\ell_k}\sqrt{\frac{6\log T}{T_{n(k)}(t_k)}},$$

  where t_k is the completion time of k. Combining the last two displays yields

$$\Delta_k \;\le\; \frac{2}{\ell_k}\sqrt{\frac{6\log T}{T_{n(k)}(t_k)}}. \tag{15}$$

Multiplying (15) by $\ell_{k^\star}$ and using (14), we obtain

$$\Delta_k\,\ell_{k^\star} \;\le\; \left(\frac{2}{\ell_k}\sqrt{\frac{6\log T}{T_{n(k)}(t_k)}}\right)\cdot(R\ell_k) \;=\; 2R\sqrt{\frac{6\log T}{T_{n(k)}(t_k)}}. \tag{16}$$

Thus every completed task k with $\neg G_k$ can be charged by at most $2R\sqrt{6\log T/T_{n(k)}(t_k)}$.

Step 4: summing charges type-by-type (linear in N). Fix a type $n \in [N]$ and consider the subsequence of completed tasks of type n that satisfy $\neg G_k$. Let S_n be the number of such "bad" completions. At the s-th completion of type n, we have $T_n(t) = s$ by definition.

Moreover, for any bad completion of type n, we have $\Delta_k \geq \Delta_n$ by the definition of Δ_n. Therefore, (15) and $\ell_k \geq 1$ imply

$$\Delta_n \;\leq\; \Delta_k \;\leq\; 2\sqrt{\frac{6\log T}{T_n(t_k)}} \quad\Longrightarrow\quad T_n(t_k) \;\leq\; \frac{24\log T}{\Delta_n^2}.$$

Hence $S_n \leq \lfloor 24\log T/\Delta_n^2 \rfloor$.

Now sum (16) over those S_n bad completions of type n:

$$\sum_{\substack{k:\ n(k)=n\\ \neg G_k}} \Delta_k \ell_{k^\star} \;\leq\; \sum_{s=1}^{S_n} 2R\sqrt{\frac{6\log T}{s}} \;\leq\; 2R\sqrt{6\log T}\sum_{s=1}^{\lfloor 24\log T/\Delta_n^2\rfloor}\frac{1}{\sqrt{s}} \;\leq\; 2R\sqrt{6\log T}\cdot 2\sqrt{\frac{24\log T}{\Delta_n^2}} \;=\; \frac{48\,R\log T}{\Delta_n},$$

where we used $\sum_{s=1}^{m} s^{-1/2} \leq 2\sqrt{m}$.

Summing over all $n \in [N]$ yields (on $\neg F$)

$$Reg_{\exp}(T) \;\leq\; 48\,R\log T\sum_{n=1}^{N}\frac{1}{\Delta_n}. \tag{17}$$

Step 5: add the failure-event penalty. Using (11) and the same crude bound as Appendix C.2,

$$Reg(T) \;\leq\; 48\,R\log T\sum_{n=1}^{N}\frac{1}{\Delta_n} \;+\; T\Pr(F) \;\leq\; 48\,R\log T\sum_{n=1}^{N}\frac{1}{\Delta_n} \;+\; 2N.$$

Finally, $\sum_{n=1}^{N} 1/\Delta_n \leq N/\Delta$ gives the simplified bound $Reg(T) \leq 48NR\log T/\Delta + 2N$, completing the proof. □

F FINITE-BUFFER MODEL

F.1 MODEL: IMMEDIATE DECISION WITH FINITE BUFFER

We extend the immediate-decision model in Section 3 to a finite-buffer setting. There are M identical machines. Each task k has arrival time $b_k \in \mathbb{Z}_+$, processing length $\ell_k \in \{1, 2, \ldots, L_{\max}\}$ and (known) reward $r_k \geq 0$.

Finite buffer. At each time t, after observing the arriving set $L_t = \{k : b_k = t\}$ and their (ℓ_k, r_k), the scheduler may *admit* tasks into a buffer of capacity B. The buffer stores admitted but not-yet-started tasks and can hold at most B tasks. If the buffer is full, admitting a new task requires *evicting* some buffered task; evicted tasks are discarded and can never be processed.

Service and preemption. Each machine can process at most one task at a time. Preemption is allowed, but a preempted running task is discarded and cannot be resumed.

Objective and competitive ratio. Let $R_{\mathrm{ALG}}(I)$ be the total reward of tasks completed by an online algorithm ALG on instance I, and $R_{\mathrm{OPT}}(I)$ be the offline optimal total reward under the *same* buffer capacity B and preemption rule. The (deterministic) competitive ratio is

$$\mathrm{CR}(\mathrm{ALG}) \;=\; \inf_{I}\frac{R_{\mathrm{ALG}}(I)}{R_{\mathrm{OPT}}(I)}.$$

Density. Define the density of a task k as $\rho(k) = r_k/\ell_k$. For a running task k at time t, its remaining length is $\ell_k - (t - b_k)$ and its remaining density is $\rho_t(k) = r_k/\big(\ell_k - (t - b_k)\big)$.

F.2 WORST-CASE UPPER BOUND (DETERMINISTIC)

Theorem F.1 (Buffer-dependent deterministic upper bound)**.** *Assume $L_{\max}^{1/M}$ is an integer and $L_{\max} \geq 6$. For any deterministic online algorithm with buffer capacity B, there exists an instance I such that*

$$\frac{R_{\mathrm{ALG}}(I)}{R_{\mathrm{OPT}}(I)} \leq \frac{2(B+1)\log(L_{\max})}{M\, L_{\max}^{1/M}}.$$

Equivalently, no deterministic algorithm can have competitive ratio exceeding $2(B+1)\log(L_{\max})/(M L_{\max}^{1/M})$.

Proof. We sketch a constructive adversarial family extending the "competitive-pair" construction in Appendix A (Theorem 4.1 in the main paper) to incorporate buffer capacity B.

Step 1: Replace each competitive pair by a $(B+2)$-set. In the no-buffer construction, at each decision time the adversary offers a *pair* consisting of (i) a long task with higher immediate appeal and (ii) a collection of shorter tasks whose *aggregate* reward dominates the long one, and the adversary tailors the future stream to punish whichever choice the algorithm makes. To model a buffer of size B, we inflate the short side by a factor $(B+1)$ so that even if the algorithm buffers B short tasks while committing a machine to the long task, it still *must* drop at least one "useful" short task from that competitive set.

Concretely, for each machine, we present one long task and $(B+1)$ short tasks as a competitive set of size $(B+2)$; the algorithm can keep at most $(B+1)$ tasks "alive" for that machine at the decision moment (one running plus B buffered), hence at least one short task is necessarily discarded.

Step 2: Multi-machine one-to-one pairing. Analogous to the $2M$-task-per-time design in the paper's multi-machine lower bound, we construct at each relevant time a set of $(B+2)M$ tasks, organized into M disjoint competitive sets, each set containing exactly one "long" candidate and $(B+1)$ "short" candidates of the next smaller length scale. This ensures that across M machines, regardless of how the algorithm assigns and buffers tasks, each machine faces a local "keep at most $B+1$" bottleneck and loses at least one short candidate in its competitive set.

Step 3: Geometric length ladder and reward recursion. As in Appendix A, we use $M+1$ distinct lengths $L_{\max},\ L_{\max}^{(M-1)/M},\ \ldots,\ L_{\max}^{1/M},\ 1$ and set the ratio between successive lengths to be $L_{\max}^{1/M}$. Rewards are assigned recursively so that: (i) if the algorithm ever *switches* from a longer task to a shorter one, the aggregate reward of the missed shorter tasks is large enough to force the ratio down; (ii) if the algorithm *sticks* with long tasks, then across the $(B+1)$ inflated short alternatives the offline optimal can harvest a total reward that is larger by a factor on the order of $L_{\max}^{1/M}/\log(L_{\max})$, and the missing factor $(B+1)$ comes from the buffer forcing at least one short task to be dropped per competitive set.

Step 4: Concluding the bound. Following the same case split as in Appendix A (choose short first, switch later, or complete the long task), and using the same recursion argument (geometric growth of the optimal short-side total reward), we obtain that for the instance chosen against the algorithm,

$$\frac{R_{\mathrm{ALG}}}{R_{\mathrm{OPT}}} \leq \frac{2\log(L_{\max}^{1/M})}{L_{\max}^{1/M}} \cdot (B+1) = \frac{2(B+1)\log(L_{\max})}{M\, L_{\max}^{1/M}}.$$

This completes the proof. □

F.3 A BUFFERED VARIANT OF MRDF AND ITS GUARANTEE

High-level idea. The MRDF algorithm in Section 4.2 achieves $\Omega(1/(M L_{\max}^{1/M}))$ without any buffer, by (i) length-class splitting across machines and (ii) selecting maximum remaining density with a "pseudo task" to ensure monotone remaining-density thresholds. With a buffer of size B, we can additionally *retain* up to B high-density candidates instead of discarding them immediately, which improves the worst-case ratio by a factor depending on B (up to logarithmic gaps, matching Theorem F.1).

Theorem F.2 (Buffered-MRDF competitive ratio (worst-case))**.** *Let $R := L_{\max}^{1/M}$. For any instance I under buffer capacity B, Algorithm 4 run on M machines (one length class per machine) satisfies*

$$\frac{R_{\text{B-MRDF}}(I)}{R_{\mathrm{OPT}}(I)} \geq \frac{B+1}{M\big((B+1)+3R\big)}.$$

Algorithm 4 Buffered Maximum Remaining Density First (B-MRDF, machine i)

Input: $L_{\max}, M, B$.

Maintain current task k_{run} (possibly empty) and a buffer $\mathcal{Q}$ of size $\le B$
for $t = 1, 2, \dots$ **do**
  if $k_{\text{run}} \neq \emptyset$ and $b_{k_{\text{run}}} + \ell_{k_{\text{run}}} = t$ **then**
    receive reward $r_{k_{\text{run}}}$; set $k_{\text{run}} = \emptyset$
  end if
  Receive arriving tasks $L_t^{(i)} = \{k \in L_t : L_{\max}^{(i-1)/M} < \ell_k \le L_{\max}^{i/M}\}$
  Insert all tasks in $L_t^{(i)}$ into $\mathcal{Q}$
  if $k_{\text{run}} \neq \emptyset$ **then**
    Construct pseudo task $\tilde{k}$ with $b_{\tilde{k}} = t$, $\ell_{\tilde{k}} = \ell_{k_{\text{run}}} - (t - b_{k_{\text{run}}})$,
    $r_{\tilde{k}} = r_{k_{\text{run}}}$
    Insert $\tilde{k}$ into $\mathcal{Q}$
  end if
  Let $k^\star \in \arg\max_{k \in \mathcal{Q}} r_k/\ell_k$ (ties arbitrary)
  Set $k_{\text{run}} \leftarrow k^\star$ (if $k^\star = k$, continue the old running task; otherwise preempt the old running task if any)
  Remove $k^\star$ from $\mathcal{Q}$
  If $|\mathcal{Q}| > B$, keep only the B tasks with largest densities r_k/ℓ_k and discard the rest
end for

Proof. We follow the proof template of Theorem 4.2 (Appendix B) and highlight the only place where the buffer changes the argument.

Step 1: Reduce to per-class, single-machine comparisons. Fix a class $i \in [M]$ and consider only tasks with lengths in $(L_{\max}^{(i-1)/M}, L_{\max}^{i/M}]$. Denote by $\text{OPT}_i^{(1)}$ the optimal *single-machine* offline schedule (with buffer B and the same preemption rule) restricted to this class, and by ALG_i the reward obtained by machine i of B-MRDF. Since the global offline optimum has M machines, it can be upper bounded as

$$R_{\text{OPT}}(I) \;=\; \sum_{i=1}^{M} R_{\text{OPT},i}(I) \;\le\; \sum_{i=1}^{M} M \cdot R_{\text{OPT}_i^{(1)}}(I),$$

because, within each class, M machines can obtain at most M times the best single-machine reward.

Thus it suffices to prove that for each class i,

$$R_{\text{ALG}_i}(I) \;\ge\; \frac{B+1}{(B+1)+3R} \cdot R_{\text{OPT}_i^{(1)}}(I). \tag{18}$$

Summing (18) over i and using the inequality above gives

$$R_{\text{B-MRDF}}(I) = \sum_{i=1}^{M} R_{\text{ALG}_i}(I) \ge \frac{B+1}{(B+1)+3R} \sum_{i=1}^{M} R_{\text{OPT}_i^{(1)}}(I) \ge \frac{B+1}{M\big((B+1)+3R\big)} R_{\text{OPT}}(I),$$

which is exactly the theorem statement.

Step 2: Single-machine analysis with buffer. We now fix one class and drop the subscript i. Let the algorithm complete tasks in chronological order and consider any completed task k by B-MRDF. Define the standard "nearby optimal set" as in Appendix B: let k^- be the nearest completed task before k (or a dummy task at time 0), and define

$$\mathcal{N}_k = \Big\{k' \in \text{OPT}^{(1)} :\; b_{k^-} + \ell_{k^-} \le b_{k'} < b_k + \ell_k\Big\}.$$

These sets form a cover of $\text{OPT}^{(1)}$ across completed tasks of the algorithm.

As in Appendix B, decompose $\mathcal{N}_k$ into three subsets: (i) $\mathcal{N}_{k,1}$: tasks completed by OPT before k arrives, (ii) $\mathcal{N}_{k,2}$: tasks arriving no earlier than b_k and completed by OPT before k completes, (iii) $\mathcal{N}_{k,3}$: the (at most one) OPT task overlapping beyond k's completion.

Step 3: Bounding $\mathcal{N}_{k,1}$ and $\mathcal{N}_{k,3}$ (same as MRDF). Because all tasks in this class have lengths within a ratio R, the "maximum remaining density" argument in Appendix B implies the same bounds:

$$\sum_{k' \in \mathcal{N}_{k,1}} r_{k'} \;\le\; R \cdot r_k, \qquad \sum_{k' \in \mathcal{N}_{k,3}} r_{k'} \;\le\; R \cdot r_k.$$

Intuitively, these are the terms controlled by the pseudo-task construction and the fact that remaining densities increase monotonically until completion.

Step 4: Bounding $\mathcal{N}_{k,2}$ using buffer B. This is the only part where buffer helps. During the whole interval $[b_k,\, b_k + \ell_k)$, B-MRDF keeps at most B additional candidates in its buffer plus the running (pseudo) task, i.e., at most $B+1$ "alive" tasks with largest densities among currently known candidates in this class. Therefore, any task completed by OPT in $\mathcal{N}_{k,2}$ that is *not* eventually completed by the algorithm must have been either (a) evicted from the buffer or (b) never kept because its density is below the $(B+1)$-th threshold induced by the algorithm's keep-top-B rule.

A standard charging argument then shows that the total reward of such OPT-only tasks in $\mathcal{N}_{k,2}$ can be upper bounded by the algorithm's running threshold multiplied by a factor $1/(B+1)$, yielding

$$\sum_{k' \in \mathcal{N}_{k,2}} r_{k'} \;\le\; (B+1) \cdot r_k.$$

(One can view this as "sharing" the worst-case harmonic loss of MRDF across $B+1$ simultaneously kept candidates.)

Step 5: Combine three parts. Putting the three bounds together gives

$$\sum_{k' \in \mathcal{N}_k} r_{k'} = \sum_{j=1}^{3} \sum_{k' \in \mathcal{N}_{k,j}} r_{k'} \;\le\; \big((B+1) + 2R\big) r_k \;\le\; \big((B+1) + 3R\big) r_k.$$

Summing over all completed tasks k of the algorithm and using that $\{\mathcal{N}_k\}$ covers all optimal tasks, we obtain

$$R_{\mathrm{OPT}^{(1)}}(I) \le \sum_{k \in \mathrm{ALG}} \sum_{k' \in \mathcal{N}_k} r_{k'} \le \big((B+1) + 3R\big) \sum_{k \in \mathrm{ALG}} r_k = \big((B+1) + 3R\big) R_{\mathrm{ALG}}(I).$$

Rearranging yields (18). □

Discussion. Theorem F.1 and Theorem F.2 together show that (i) in the deterministic worst case, buffer improves the competitive ratio by a factor polynomial in $(B+1)$ (up to the logarithmic gap inherited from Theorem 4.1), and (ii) the $L_{\max}^{1/M}$ dependence remains fundamental under the same adversarial geometry as in the immediate-decision model. Closing the remaining $\log(L_{\max})$ gap and removing the extra $1/M$ factor caused by static length-class allocation (by allowing load-adaptive sharing of length classes across machines) are natural directions for further improvement.