Resource Management for Stochastic Parallel Synchronous Tasks: Bandits to the Rescue

Anonymous Author(s)

Abstract

In scheduling real-time tasks, we face the challenge of meeting hard deadlines while optimizing for some other objective, such as minimizing energy consumption. Formulating the optimization as a Multi-Armed Bandit (MAB) problem allows us to use MAB strategies to balance the exploitation of good choices based on observed data with the exploration of potentially better options. In this paper, we integrate hard real-time constraints with MAB strategies for resource management of a Stochastic Parallel Synchronous Task. On a platform with $\boldsymbol{M}$ cores available for the task, $\boldsymbol{m} \leq \boldsymbol{M}$ cores are initially assigned. Prior work has shown how to compute a virtual deadline such that assigning all $\boldsymbol{M}$ cores to the task if it has not completed by this virtual deadline guarantees that the deadline will be met. An MAB strategy is used to select the value of $\boldsymbol{m}$. A Dynamic Power Management (DPM) energy model considering CPU sockets and sleep states is described. Experimental evaluation shows that MAB strategies learn consistently suitable $\boldsymbol{m}$, and perform well compared to binary exponential search and greedy methods.

Keywords: Multicore scheduling, Multi-Armed Bandit, DAG task, Parallel Synchronous Task

1 Introduction

When scheduling real-time tasks, we often want to manage the use of resources to optimize some objective and also ensure that the tasks' deadlines are met. The objective may be, for example, minimizing energy consumption or maximizing the processor time available for other tasks in uninterrupted time periods. The problem can be modeled as a Multi-Armed Bandit (MAB) problem (Slivkins, 2019), where a decision maker repeatedly selects one of several fixed options, known as arms or actions; the impact of each action is not known a priori, and it is influenced by uncertainty. MAB strategies optimize for the average case while balancing exploration and exploitation to achieve the best expected result over a time period. The average behavior is often the main concern in resource management beyond strict timing requirements. For

example, minimizing the energy consumption implies minimizing the integral of the power consumption over time, essentially minimizing the average power consumption. This paper gives an example of the integration of strict timing guarantees with MAB strategies to minimize energy consumption.

A scheduler initially assigns m cores to a DAG task but can use $M \geq m$ identical computing cores if needed. For a compute-bound task and a work-conserving schedule, Papadopoulos et al. (2022) have shown that the task's deadline is met if it is assigned all M cores at a virtual deadline V. V depends on the initial number of cores m assigned and the worst-case properties of the DAG task. We rely on these results to ensure that deadlines are met and outline an MAB approach to select m over time, balancing exploration and exploitation. In Papadopoulos et al. (2022), resource management strategies are evaluated, selecting m based on response time and m of the most recent task invocation. These strategies aim for the lowest possible m that keeps the response times below $V(m)$. The MAB approach differs in two important ways compared to these strategies. 1) Optimization is done with respect to the expected reward over time instead of the most recent observation. 2) The reward function is decoupled from the arm selection, allowing for optimizing the resource management towards any goal dependent on the arm response times.

The reward function is constructed to minimize the energy consumption under an energy model of a multicore system with CPU sockets and Dynamic Power Management (DPM) with sleep states. The energy model is based on data from Schöne et al. (2015).

Contribution: The main contribution is the *MAB application* to improve average behavior while ensuring hard real-time guarantees. For this application, we define and use a *Stochastic Parallel Synchronous Task*, a special case of a DAG task but a generalization of the Parallel Synchronous Task (Saifullah et al., 2013). We derive *response time bounds* for different initial core allocations. Each arm represents a choice of initial core allocation. The MAB is implemented as a bootstrap/ bag approximation (Oza and Russell, 2001) of Thompson sampling (Slivkins, 2019). In the proposed partial-feedback MAB, information about unexplored arms is derived from arms that have been explored, using the response time bounds.

In the evaluation, the proposed MAB core allocation strategy is compared with an MAB not using the derived response time bounds, the Binary-Exponential Search (BES) strategy from Papadopoulos et al. (2022) adapted to the energy model, and a greedy strategy based on the energy model. The evaluation is performed for several selected task structures with different computation time variances and deadlines.

Outline of the paper: A background on Multi-Armed Bandits is given in Section 2. In Section 3 related work regarding task models with precedence constraints, bandit scheduling and energy-aware scheduling is outlined. Notation, the task model, and scheduling along with definitions are presented in Section 4. In Section 5 the resource management problem is formulated (Section 5.1). Methods from Papadopoulos et al. (2022) are introduced with an example (Section 5.2). The proposed partial feedback MAB and the response time bounds are presented in Section 5.3. The energy model for the reward function is outlined in Section 6, and the evaluation in Section 7. In Section 8 conclusions and future work are discussed.

2 MAB Background

A MAB problem is a reinforcement learning problem in which an algorithm makes decisions over time under uncertainty (Slivkins, 2019). The algorithm selects one out of K possible actions, called **arms**, in each of T rounds. Each action generates a reward according to a fixed but unknown probability distribution, and the goal is to maximize the total reward over the T rounds, the **horizon**. There are many applications of MAB approaches, including healthcare, finance, dynamic pricing, recommender systems, anomaly detection, and telecommunications (Bouneffouf et al., 2020).

A standard approach to comparing different MAB algorithms is the concept of regret. Here, the sum of rewards for an algorithm over a horizon of T rounds is compared with the sum of expected rewards when consistently choosing the arm with the highest possible expected reward. That is, with a fixed but unknown reward distribution $\mathcal{D}_k$ of each arm k, the mean reward of an arm is denoted $\mu_k = \mathbb{E}[\mathcal{D}_k]$. The highest possible mean reward is $\rho^\uparrow = \max_k \mu_k$. The regret $R(T)$ over horizon T of an algorithm that at each round i takes an action a_i leading to reward $\rho(a_i)$ is defined as:

$$R(T) = \rho^\uparrow \cdot T - \sum_{i=1}^{T} \rho(a_i) \tag{1}$$

Feedback from a chosen action can be structured into three types (Slivkins, 2019). Bandit feedback provides the reward for the chosen arm and no additional information. In a complete feedback setting, the agent can retrospectively observe the reward for all arms. Partial feedback implies that further information is provided beyond the reward of the chosen arm. In our case, an arm's response time provides information about the task's properties that is useful for all arms.

The reward model can be i.i.d. (independent and identically distributed), where the rewards of each arm are drawn from the same probability distribution, independent of the round and previous actions and rewards. Other reward models include rewards chosen by an adversary or evolving according to a random process (Slivkins, 2019). In our case, we consider i.i.d. rewards.

The arm choice at a round i is based on the current estimates of the mean rewards μ_k of each arm. Algorithms often consider confidence intervals of the mean rewards. One common algorithm, Successive Elimination, removes an arm a from consideration when the upper bound of the confidence interval for μ_a is lower than the lower bound of the confidence interval for the mean reward μ_b of some other arm b. Another common algorithm, UCB1, always selects the arm with the highest upper confidence bound on the mean reward. The intuition is that upper confidence bound is high due to the arm being a good choice, or due to the arm being unexplored and having a large confidence interval.

Bayesian bandits use the concepts in Bayesian statistics and assume that an unknown quantity is sampled from a known distribution (Slivkins, 2019). The expected reward is maximized over the distribution, referred to as a belief model. Before selecting an arm in round t, we refer to the prior distribution of the belief. After obtaining the reward, the belief model is updated, and we refer to the posterior distribution.

The posterior can be used as a prior in the next round. An algorithm for arm selection in a Bayesian bandit is Thompson sampling. In each round, every arm is assigned a probability of selection equal to the probability that the arm is optimal, given the history of previous rounds. An equivalent formulation is the following: at each round, a reward is sampled from the prior distribution over the expected reward of each arm. The arm corresponding to the highest reward is selected.

In some cases, the posterior distribution can be derived as a closed-form expression from a conjugate prior of the same family. However, in many cases, the posterior belief model is approximated (Bietti et al., 2021). One such approximation method is Bagging or Online Bootstrap Thompson Sampling (Bietti et al., 2021; Oza and Russell, 2001). In this approach, several bags or replicates estimate the mean reward of the arms from observations in the past. Each replicate is updated with a new observation with a certain probability, resulting in the bags having different histories and mean reward estimates. In this way, the set of bags estimates the belief distribution of the reward mean.

In Contextual Multi-Armed Bandit (CMAB) problems, some feature vector or context is observed prior to the decision, and reward distributions are different for different contexts. For example, in a recommender system, the context could be user demographic information.

In a restless bandit setting (Whittle, 1988), the reward distributions associated with the arms, including non-selected arms, may change across rounds. The objective is to maximize the average reward over an infinite horizon.

This paper considers the traditional MAB setting. There is no context information prior to the arm choice, and each arm's reward distribution remains fixed, but our knowledge about them evolves.

3 Related Work

3.1 Task Models With Precedence Constraints

Modeling precedence constraints with DAGs is common, both for precedence constraints between different tasks (Graham, 1969) and for precedence constraints within the same task (Baruah et al., 2012; Saifullah et al., 2013). In Graham's list scheduling (Graham, 1969), a ready task is selected for processing each time a processor is idle. A ready task is a task with no precedence constraints or fulfilled precedence constraints. Graham's list scheduling does not produce a sustainable schedule (Burns et al., 2008), but bounds for the response time differences are presented (Graham, 1969).

In the parallel synchronous task model (Saifullah et al., 2013), a task's jobs consist of sequential segments, each containing threads that can run in parallel.

Even richer DAG-based models have been developed and studied. In the conditional parallel DAG task model (Melani et al., 2015), parallel nodes are combined with nodes representing if-then-else clauses. The multi-DAG model (Fonseca et al., 2015) models different execution flows as separate DAGs.

Papadopoulos et al. (2022) is the work most closely related to this paper, and we describe the necessary content in Sections 4 and 5.2.

3.2 Bandit Scheduling

Yu et al. (2018) have proposed using Restless Bandits for stochastic deadline scheduling in a data center. Here, arms represent positions in the job queue, and selecting an arm is equivalent to processing the job at that position in the queue on one processor. The problem is shown to be indexable. Whittle's index policy is not optimal, but the gap-to-optimality is bounded. Chen et al. (2022) used a measure of information freshness in a restless bandit setting to determine which states to update. In a network setting, Raghunathan et al. (2008) selected packets for broadcasting with a similar restless bandit approach. Borkar et al. (2017) have applied a restless bandit approach in which a queue is selected for packet transmission on a channel, considering costs modeling the delay constraints and transmission energy consumption. To the best of our knowledge, bandit scheduling has not been applied to DAG or synchronous task models. Most of the approaches in the literature use arms to represent packets or jobs. This requires a restless bandit approach as the states of all packets/ jobs evolve, whether or not they are transmitted/ processed. In our work, an arm represents the number of cores initially assigned to a task. This enables a more simplistic MAB problem while the considered task model is more complex.

3.3 Energy-Aware Scheduling

In modern processors, the two main approaches to control energy consumption are Dynamic Voltage and Frequency Scaling (DVFS), in which the voltage and CPU frequency are lowered at times, and Dynamic Power Management (DPM), in which a number of different processor idle states (C-states[1]) are used (Bambagini et al., 2016). Deeper idle states save more power by turning off additional components in the CPU, but they require higher wake-up latencies. As noted in Le Sueur and Heiser (2010), the potential advantages of DVFS over DPM are decreasing. One reason is that reduced transistor sizes lead to an increased proportion of leakage currents, as these are a quantum phenomenon (Bambagini et al., 2016).

Bambagini et al. (2016) survey and discuss work on energy-aware scheduling for real-time systems. Sheikh and Pasha (2018) presented a survey on energy-efficient multicore scheduling for hard real-time systems. Xie et al. (2021) surveyed low-energy parallel scheduling algorithms focused on DVFS techniques. Additional works that focused recently on the use of DVFS and big.LITTLE architectures for real-time tasks can be found in Mascitti et al. (2021). The former work introduces BL-CBS, an extension of the Adaptive Partitioned EDF scheduler in Abeni and Cucinotta (2020), where each real-time task CBS server is dynamically placed on the most energy-convenient CPU, chosen among big or LITTLE ones, considering the impact of possible frequency changes, needed to preserve schedulability, on the power consumption of the whole affected island. The technique is further extended in Mascitti and Cucinotta (2021) to schedule real-time DAG tasks.

The power P_{gate} of a single active gate is described as a function of the probability of gate switching α, the loading capacitance C_L, the supply voltage V, the clock

[1] C0 is actually the name of the fully operational state of the CPU, when it is executing instructions, C1 is the first software-only idle state, whereas C2, C3, etc... are the deep idle states. For more details, see: https://doc.opensuse.org/documentation/leap/archive/42.2/tuning/html/book.sle.tuning/cha.tuning.power.html.

frequency f, the short circuit current I_{sc} and the leakage current I_{leak} (Bambagini et al., 2016; Chandrakasan et al., 1992; Sheikh and Pasha, 2018)[2]:

$$P_{gate} = \alpha \cdot C_L \cdot V^2 \cdot f + V \cdot I_{sc} + V \cdot I_{leak} \tag{2}$$

An important part of the power savings of DVFS is the concurrent reduction in the supply voltage enabled by a reduced clock frequency (Le Sueur and Heiser, 2010). In some work (Pillai and Shin, 2001; Aydin et al., 2001; Bambagini et al., 2016), the dynamic power component is assumed to be $P(f) = \beta \cdot f^{\delta}$, $2 \leq \delta \leq 3$. With today's low core voltages, the voltage-scaling window is reduced, resulting in a smaller benefit of the dynamic component (Le Sueur and Heiser, 2010). A lower clock frequency also implies a longer computation time, limiting when a low-power idle state can be used.

Schöne et al. (2015) describe the per-core C-states of x86 processors, and Package C-states that can be entered if all the cores in a socket are in a sleep state. In these Package C-states, power demand is reduced further, for example, by partially disabling the last-level cache. Recent work by Antoniou et al. (2024) have proposed an alternative architecture to significantly reduce the wake-up latency of sleep states while retaining most of the power savings.

Sleep state arbiters have been developed to minimize data center response times and latency while saving energy. Examples include feedback control (Zhan et al., 2016) and machine learning (Sharafzadeh et al., 2019). In Chou et al. (2019), sleep states are coordinated with request delays and voltage frequency scaling to reduce tail latency and energy consumption. Request tail latency is estimated using random variables.

Despite the reduced gains in DVFS techniques, these and hybrid methods combining DVFS and DPM are common in the literature. The energy model presented in Section 6 uses DVFS only as a complement to DPM for cores in sleep states. This model can be incorporated into the framework with virtual deadlines for DAGs developed in Papadopoulos et al. (2022) with minimal modification.

4 System Model and Notation

The main notation used in the paper is listed in Table 1.

4.1 Task Model

We consider a periodic task τ, generating a sequence of jobs J_i, $i \in \mathbb{N}$. Job arrivals are separated by the period p, and we refer to the arrival of J_i as round i. The task, referred to as a stochastic parallel synchronous task, has the internal structure of a parallel synchronous task (Saifullah et al., 2013). That is, τ is composed of s sequential computation segments. There are u_j threads in the j-th segment of τ, and the total number of threads is U. The task structure is illustrated in Fig. 1.

Each segment ends in a synchronization point, so all threads in one segment must be completed before the next segment can begin its execution. In our task model, the execution time e_{ijk} of the k-th thread in job i's j-th segment is the outcome of an

[2] In Bambagini et al. (2016) α also multiplies the second term.

Table 1 Notation used in this paper.

Symbol	Description
τ, J_i	Task, job (at round) i of the task.
p	Task period.
s	Number of segments in a parallel synchronous τ.
u_j	Number of threads in the jth segment of τ.
U	Total number of threads in τ.
$\mathcal{E}_{jk}$	Execution time random variable of the kth thread in the jth segment of τ.
e_{ijk}	Execution time of thread k in the jth segment of J_i.
L, W	The deterministic worst-case span, and work of τ.
r_i	The response time of J_i.
w_i	The total work of J_i.
D	The relative deadline of τ.
M	The maximum number of cores available for τ.
m_i	The number of cores assigned to J_i at its arrival.
Ω	The number of valid m.
$V(m)$	The virtual deadline of τ associated with m.
$S_i(t)$, $Q_i(t)$	The partial schedule and profile at time t after arrival.
n_s, n_{cs}	Number of sockets, and number of cores per socket.
P_X	The power consumption of a core in power state X.
Δ_{XY}	Transition latency in transitioning from power state X to Y and back to X.
$P_{\Delta XY}$	The average power consumption during Δ_{XY}.
c_{sx}, c_{rx}	The cores out of x cores that make up full sockets, and the remaining cores.
E_i	The modeled energy consumption from arrival to deadline of J_i.
$E^{\uparrow}$, $E^{\downarrow}$	The maximum and minimum possible energy consumption of a job.
T	Horizon (a specified number of rounds).
ρ	Reward.
B	Bag (bootstrap replicate) to estimate arm mean rewards.
κ	Number of bags.

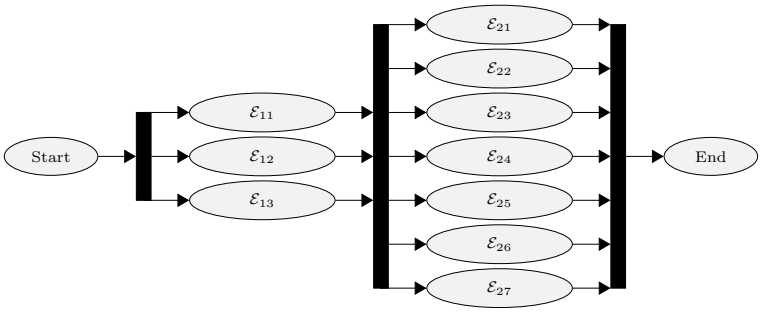

Fig. 1 A stochastic parallel synchronous task example.

i.i.d. random variable $\mathcal{E}_{jk}$ with bounded support. The work w_i of job i is the sum of the execution times of all threads.

The main difference with respect to the model in Saifullah et al. (2013) is the specification of thread execution times. In Saifullah et al. (2013) the execution times of threads in a segment are specified by the worst case execution requirement of the task segment. In our model, execution times are outcomes of random variables. We assume that a fixed but unknown probability distribution specifies the execution requirement

for a thread in a segment; different threads in the same segment may have different probability distributions.

The worst-case span L and work W of a DAG task are defined in Papadopoulos et al. (2022), and we define them here for the stochastic parallel synchronous task. The worst-case span L of τ is a deterministic conservative estimate of the sum of the longest thread execution times from each segment as formalized in Eq. (3)

$$L \geq \sum_{j=1}^{s} \max_{k=1,\ldots,u_j} e_{ijk}, \forall i \tag{3}$$

W is the worst-case work of τ, a deterministic conservative estimate of the worst-case total computation time of all threads, as stated in Eq. (4).

$$W \geq \sum_{j=1}^{s} \sum_{k=1}^{u_j} e_{ijk}, \forall i \tag{4}$$

The assumption of bounded support for the execution time distribution of the k-th thread of the j-th segment $\mathcal{E}_{jk}$ of τ is necessary to assume that τ has deterministic worst-case work and span.

Let r_i denote the response time of J_i. The response time for any job should not exceed the relative deadline D. The deadline is implicit, $D = p$.

4.2 Scheduling

As in Papadopoulos et al. (2022), jobs are scheduled on up to M identical cores. At the arrival, J_i is assigned m_i cores. Depending on m, a virtual deadline V is calculated according to Eq. (5).

$$V(m_i) = \left\lfloor \frac{M \cdot (D - L) - (W - L)}{M - m_i} \right\rfloor, m_i < M \tag{5}$$

If J_i is not completed at the time of $V(m_i)$, all M cores are assigned to J_i at this point. As shown in Papadopoulos et al. (2022) this guarantees that J_i will meet its deadline.

When a job J_i is completed, the resulting response time r_i and total computation time w_i are reported, and this information is available to determine subsequent core assignments.

The guarantee that jobs meet their deadlines is based on the bound on the response time $r(m)$ of a DAG task job assigned m cores until completion in Eq. (6).

$$r(m) \leq L + \frac{W - L}{m} \tag{6}$$

This relies on the fact that the mean work of one of the cores and not on the critical path is at most $\frac{W-L}{m}$ (Melani et al., 2015), assuming no interference from other tasks,

a constrained relative deadline ($D \leq p$) and a work-conserving schedule. $\frac{W-L}{m}$ is the latest possible point when one core is available to process the work on the critical path.

The threads within a task are scheduled according to list scheduling (Graham, 1969); that is, the threads are ordered in a list. Threads ready to execute will be run in the order they are listed. We assume that the list starts with the threads of the first segment, followed by the threads of the second, third, and so on. This is without loss of generality - assume that in a list ordering LO, thread a of segment H precedes thread b of segment H', $H' < H$. Since thread a cannot start execution until all threads of H' have completed execution, it follows that moving thread a just after the last thread of segment H' in LO will give the same schedule.

We provide the definitions below for the *partial schedule*, *profile*, and the *ahead* relation of profiles of a stochastic parallel synchronous task with threads scheduled according to list scheduling.

Definition 1. *The partial schedule $S_i(t)$ of job J_i at time t, $0 \leq t \leq D$, specifies for each thread the total time the thread has been processed up to time t.*

The profile $Q_i(t)$ of a partial schedule $S_i(t)$ is the list $Q_i(t) = (q_1(t), q_2(t), \ldots, q_U(t))$ where $q_l(t)$, $l = 1, 2, \ldots, U$, represents the remaining processing time at time t after the job's arrival of thread l from J_i, with threads ordered according to the list scheduling priority.

Since the threads' processing times are unknown, the remaining thread processing times are unknown. Next, we introduce an ordering relation among profiles that corresponds to different partial schedules of the job at different points in time.

Definition 2. *The profile $Q''_i(t'') = (q''_1(t''), q''_2(t''), \ldots, q''_U(t''))$ is ahead of profile $Q'_i(t') = (q'_1(t'), q'_2(t'), \ldots, q'_U(t'))$ if $q''_a(t'') \leq q'_a(t'), \forall a$. We denote this as $Q''_i(t'') \leq Q'_i(t')$.*

We note that the ahead relation is transitive. If $Q''_i(t'') \leq Q'_i(t')$ and $Q'''_i(t''') \leq Q''_i(t'')$ then $Q'''_i(t''') \leq Q'_i(t')$.

We also note that if at time t' the partial schedule S'_i executes a thread from segment H', and at time t'' the partial schedule S''_i executes a thread from segment H'' that follows H', then $Q''_i(t'')$ is ahead of $Q'_i(t')$.

5 Resource Management

In this section, the problem formulation is outlined in Section 5.1. A motivating example is presented in Section 5.2, along with descriptions of methods from Papadopoulos et al. (2022). The proposed partial feedback MAB approach is outlined in Section 5.3. In Section 5.4 we return to the motivating example.

5.1 Problem Formulation

We want to choose the number of cores m to initially assign to a stochastic parallel synchronous task τ as described in Section 4, that minimizes a regret related to the task's response time distribution for the arms. In other words, we want to choose the arm that maximizes the total reward over a specified time horizon, where the reward of a job depends on the response time, execution requirement, and core assignment. The exact structure of the task is unknown to the scheduler, only the worst-case work

W and span L are known. The scheduler observes the response time and total work of jobs after their execution. The deadline is guaranteed to be met, given that M cores are assigned to the task at the virtual deadline.

5.2 Motivating Example and Methods From Related Work

In Papadopoulos et al. (2022), the objective was to find the ideal initial assignment of cores, resulting in a response time as close as possible to the virtual deadline without exceeding it. We can reformulate this to the probabilistic case - assume the objective is to find the initial assignment of cores, resulting in the average response time as close as possible to the virtual deadline without exceeding it. Let us outline and apply two of the methods evaluated in Papadopoulos et al. (2022), namely the *binary search* and the *binary-exponential search*, to a task as defined in Section 4. We will discuss why these methods are unsuitable for the stochastic parallel synchronous task model.

The general structure of the algorithms is outlined in Algorithm 1. After some initialization, at each round a core allocation is selected. The next job is run with the selected allocation, and the resulting response time and work are observed. Some update is performed based on the observations and the selected allocation.

Algorithm 1: General structure for core allocation over a horizon.

Input: Horizon T
Output: Allocations $(m_1, \cdots, m_T)$, response times $(r_1, \cdots, r_T)$, work $(w_1, \cdots, w_T)$

```
Function CoreAllocatorHorizon(T):
    Init()
    for i ∈ 1 : T do
        m_i ← CoreAllocator()
        r_i, w_i ← RunTask(m_i)
        Update(m_i, r_i, w_i)
```

Example 1. *Our example task has $s = 2$ segments, the first segment has $u_1 = 8$ threads, and the second has $u_2 = 4$ threads. The number of cores m to assign is in the range $[1, M], M = 10$. The scheduler uses a relative deadline $D = 16$, worst-case work $W = 52$, and span $L = 8$ to calculate the virtual deadlines.*

Example 2. *In a deterministic version of Example 1, the threads in the first and second segments have the lengths $(2, 2, 2, 5, 5, 2, 2, 2)$ and $(1, 1, 3, 3)$ in scheduling order.*

Example 3. *In a stochastic version of Example 1, the threads in the first segment have a length of 2 with 0.75 probability and 5 with 0.25 probability. The threads in the second segment have a length of 1 with 0.5 probability and 3 with 0.5 probability.*

5.2.1 Binary Search for Selecting m

We recall the binary search algorithm from Papadopoulos et al. (2022), intended for tasks with unknown and constant typical workload. It maintains an interval $(lo, hi], lo < hi$. For a deterministic task, the interval contains the ideal m resulting in a

Algorithm 2: Binary search initialization.

Input: Maximum number of cores M
Output: $(lo, hi]$ interval
Function `InitBS(`M`):`
    $lo \leftarrow 0$
    $hi \leftarrow M$

response time as close as possible to the virtual deadline without exceeding it. Initially $lo = 0$ and $hi = M$ - Algorithm 2 is the Init-function in Algorithm 1. The core allocation-function is Algorithm 3, $m_1 = \lceil \frac{lo+hi}{2} \rceil$. In the following rounds, m_{i+1} is determined and the $(lo, hi]$ interval is updated from m_i and r_i, where i represents the round when J_i arrives. The algorithm for updating the interval is outlined in Algorithm 4. Note that although lo starts at 0, $m_i > 0, \forall i$ because of the ceiling operation and that hi is only assigned to previous values of m.

Algorithm 3: Binary search for selection of m at round i.

Input: $(lo, hi]$ interval
Output: Cores m_i
Function `CoreAllocatorBS(`r_i, m_i, lo, hi`):`
    $m_i \leftarrow \lceil \frac{lo+hi}{2} \rceil$

Algorithm 4: Binary search update of $(lo, hi]$ interval.

Input: Response time r_i, cores m_i, $(lo, hi]$ interval
Output: Updated $(lo, hi]$ interval
Function `UpdateBS(`r_i, m_i, lo, hi`):`
    if $r_i > V(m_i)$ **then**
        $lo \leftarrow m_i$
    if $r_i < V(m_i)$ **then**
        $hi \leftarrow m_i$

We apply the binary search allocation to the deterministic task in Example 2. The allocation m and the interval are shown in Fig. 3, and the scheduling for the different m are shown in the first row of Fig. 2. The binary search starts with the interval $(lo = 0, hi = 10]$ and an initial $m_1 = 5$ corresponding to $V(5) = 7$ calculated from Eq. (5). In the first segment, three threads with length 2 and two with length 5 are scheduled, and the three remaining threads are scheduled at time 2, resulting in the completion time of 5 for the first segment. The response time $r_1 = 8$ is longer than the virtual deadline, and lo is updated to 5. The next $m_2 = \lceil (5 + 10)/2 \rceil = 8$, corresponding to $V(8) = 18$. The response time is 10, below $V(8)$, so hi is updated to

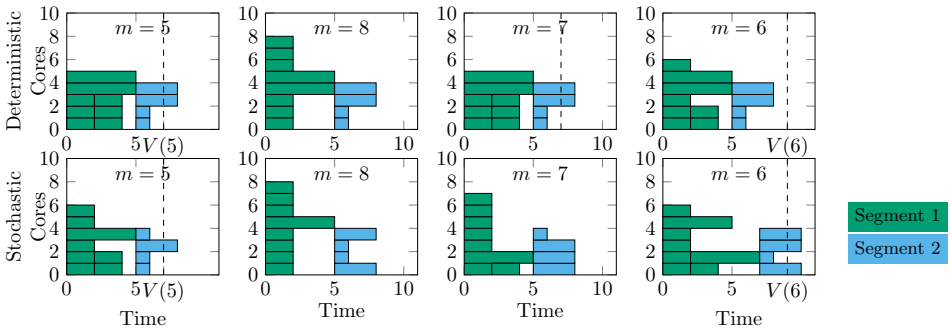

Fig. 2 Scheduling in the motivating example.

8, and the next $m_3 = 7$. Here we have $V(7) = 12$ and $r_3 = 8$, and a new $hi = 7$. This gives $m_4 = 6$, $r_4 = 8$, $V(6) = 9$, leading to $hi = 6$. From this stage, we will remain at $m = \lceil (5 + 6)/2 \rceil = 6$, the lowest m resulting in $r < V(m)$.

Next, we apply the binary search allocation to the stochastic task in Example 3. The evolution of the range and m selection is shown in Fig. 3 and the scheduling in the second row of Fig. 2. We first select $m_1 = 5$. At this time, the threads in the first segment have lengths $(2, 2, 2, 5, 2, 2, 2, 2)$, and in the second $(1, 1, 3, 1)$. The first segment has a completion time of 5, as all three threads in the first segment scheduled at 2 have length 2. The response time is therefore 8, longer than $V(5) = 7$, so lo is replaced with 5, and the new $m_2 = 8$. In this job instantiation, the threads in the first segment have lengths $(2, 2, 2, 2, 5, 2, 2, 2)$, , and the threads in the second segment have lengths $(3, 1, 1, 3)$, resulting in $r_2 = 8$. $V(8) = 18 > 8$, and $hi = 8$. Next, $m_3 = 7$ and the thread lengths of the first and second segment are $(2, 5, 2, 2, 2, 2, 2, 2)$ and $(3, 3, 3, 1)$ respectively. This results in a response time of $r_3 = 8 < V(7) = 12$, and $hi = 7$. In the next round, $m_4 = 6$, and the threads in the first segment have the lengths $(2, 2, 2, 2, 5, 2, 2, 5)$, and in the second segment thread lengths are $(3, 1, 3, 3)$. Now there is one thread of length 5 starting at 2, so the first segment completes at 7 and the response time is $10 > V(6) = 9$. leading to $lo = 6$. From now on, we will always select $m = 7$, although $m = 6$ is the lowest allocation where the average response time is lower than the virtual deadline. The binary search allocation is not suitable when the thread execution times may vary between jobs, such as in the stochastic parallel synchronous task.

5.2.2 Binary-Exponential Search for Selecting m

The *binary-exponential search*(BES) from Papadopoulos et al. (2022) enables adjustment of the interval $(lo, hi]$ from changes to the response times. For example, if the algorithm has converged, so that $m_i = hi$, but the response time is longer than the virtual deadline, $r_i > V(hi)$, it seems that we have converged to the wrong value. In this case hi will be increased by 2. If the response time in the next round is still longer than $V(hi)$, hi is increased by 4, with an exponential increase of hi or decrease of lo if response times are misaligned with the virtual deadlines of the (lo, hi] interval. In

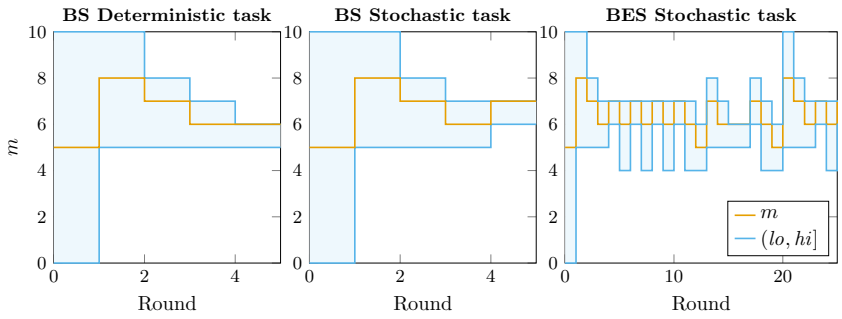

Fig. 3 The $(lo, hi]$ interval and the core allocation m in the motivating example task.

Algorithm 5: Binary-exponential search initialization.

Input: Maximum number of cores M
Output: $(lo, hi]$ interval, inc, dec

```
1 Function InitBES(M):
2     lo ← 0
3     hi ← M
4     inc ← 2
5     dec ← 2
```

addition to the $(lo, hi]$ interval, the BES maintains the variables inc and dec, initialized to 2 as outlined in Algorithm 5. These are used to enable an exponential increase of hi or decrease of lo if response times are misaligned with the virtual deadlines of the $(lo, hi]$ interval. The core allocation part is the same as for the *binary search* (Algorithm 3), and the modified update function is outlined in Algorithm 6.

Applying the BES algorithm to Example 3 leads to the interval being updated frequently, as illustrated in Fig. 3. The algorithm will select m near the ideal allocation, but the random variations in the response times will cause repeated updates of the search interval. The BES algorithm is constructed for core assignment for a task that changes characteristics at certain points in time, but where the structure and thread execution times remain the same between these points. Therefore, it is not well suited for finding m that is good over time for a stochastic parallel synchronous task.

5.3 Partial Feedback Bayesian MAB

An MAB algorithm selects the initial number of cores m_i assigned to J_i. Each m that is valid represents an arm. The task must be schedulable for each valid arm, and a necessary schedulability condition is that the virtual deadline according to Eq. (5) is non-negative. There may be further restrictions on valid m; the energy model presented in Section 6 and used in the evaluation is one such example. The number of valid m is denoted by Ω.

The MAB models a current belief about each arm's reward and response time. The arms' response time distributions all depend on the properties of the parallel

Algorithm 6: Binary-exponential search update of $(lo, hi]$ interval.

Input: Response time r_i, cores m_i, $(lo, hi]$ interval, interval update terms inc, dec
Output: Updated $(lo, hi]$ interval and terms inc, dec

```
Function UpdateBES(r_i, m_i, lo, hi, inc, dec):
    if r_i > V(m_i) then
        if m_i = hi or r_i > V(hi) then
            hi ← min(hi + inc, M)
            inc ← inc · 2
        lo ← m_i
    if r_i < V(m_i) then
        if r_i < V(m_i − 1) then
            if m_i = lo + 1 or r_i < V(lo) then
                lo ← max(lo − dec, 0)
                dec ← dec · 2
        hi ← m_i
    if hi − lo ≤ 1 then
        inc ← 2
        dec ← 2
```

synchronous task. Generally, we expect a higher initial number of cores to result in a shorter response time. However, this is not always the case, as was already pointed out in Papadopoulos et al. (2022). In Section 5.3.3 we will discuss this further and provide response time bounds for different initial core allocations. The response time bounds are used in partial feedback to derive information about the reward of one arm from observations of another arm. The response time bounds are derived using the stochastic parallel synchronous task model, but without assumptions on the number of segments or threads. An MAB approach without response time bounds may be applied to a general DAG task.

The MAB maintains κ bags (bootstrap replicates), each estimating the mean reward for every arm. In Thompson sampling, an arm is selected with a probability equal to the probability that it is the best arm (that it has the highest mean reward) based on the history. We have chosen a bag implementation of the MAB, as it is applicable for general reward distributions and straightforward to implement and explain. There are many options for MAB implementations (Slivkins, 2019; Bietti et al., 2021), and many of these may perform better but may restrict the reward model or require additional steps, for example optimization. There are caveats (Rubin, 1981; Russo et al., 2018) with and benefits (Eckles and Kaptein, 2019) of the bootstrap method. For example, a low number of bags tends to make the algorithm greedy (Eckles and Kaptein, 2019).

Algorithm 7: MAB initialization.

Input: set of valid arms *validArms*, *horizon*, number of bags κ
Output: Initialized bags, valid arms.

```
1 Function InitMAB(validArms, κ):
2     bags ← κ empty bags
3     minArm ← min(validArms)
4     maxArm ← max(validArms)
```

5.3.1 Notation Related to the Partial Feedback MAB Algorithms

$\mathcal{U}(a, b)$ denotes a uniform probability distribution on the range $[a, b]$. We use the same notation for the continuous uniform distribution on $[0, 1]$ and the discrete uniform distribution on an integer range. $Poisson(\lambda)$ refers to the Poisson probability distribution with expectation λ.

In pseudocode, we use the function `SampleFrom`. If the argument is a probability distribution, `SampleFrom` returns a sample generated from the probability distribution. If the argument is a set, `SampleFrom` returns an element drawn uniformly at random from the set.

For estimating rewards, each bag B keeps the sum of rewards obtained for an arm m, in $B.rewSum[m]$. The number of rewards observed for the arm is stored in $B.numObs[m]$. To provide estimates for other arms, all observed response times and work for m in B are stored in the arrays $B.rt[m]$ and $B.work[m]$ respectively. The nearest lower and higher arm with observations are stored in $B.nearLoMap[m]$ and $B.nearHiMap[m]$, respectively.

5.3.2 Main Algorithms for MAB Initialization, Core Assignment and Update

The overall MAB algorithm has the general structure as Algorithm 1. In Algorithm 7, the initialization of the bags is performed. In each round, one of the bags is selected at random, and the arm m_i is chosen as the arm with the highest mean reward according to this bag, as outlined in Algorithm 8. The estimate of mean rewards in a bag is described in detail in Section 5.3.4. We schedule J_i with m_i core assignment, then retrieve the response time r_i and total work w_i. In the update algorithm Algorithm 9 the reward ρ_i is calculated. For each bag B, we generate $nUpdatesBag$ as an outcome of $Poisson(1)$, and update B with $nUpdatesBag$ copies of m_i, r_i, w_i, and ρ_i. This means that the expected number of observations in each bag after round i is i, but the bags will contain different parts of the history. A bag used to select m_i at round i is equally likely as any other bag to be updated with the resulting observation in one or more copies. Since the bags contain randomly selected parts of the history, the proportion of bags where an arm has the highest mean reward estimate is an approximation of the probability that the arm has the highest mean reward given the history. In this way, selecting a bag at random and taking the best arm given the bag's observations is an approximation of selecting an arm with the probability that it is the best given the history.

Algorithm 8: MAB core allocation at round i.

Input: set of κ bags $bags$, min and max valid arm $minArm$, $maxArm$.
Output: Core allocation m_i

```
Function CoreAllocatorMAB(bags, minArm, maxArm):
    B ← SampleFrom(bags)
    if B.empty() then
        m_i ← SampleFrom(U(minArm, maxArm))
    else
        m_i ← HighestEstMeanRew(B, minArm, maxArm)
```

Algorithm 9: MAB update at round i.

Input: set of κ bags $bags$, arm m_i, observed response time, work r_i, w_i
Output: Updated bags.

```
Function UpdateMAB(bags, m_i, r_i, w_i):
    ρ_i ← Reward(r_i, w_i, m_i)
    for B ∈ bags do
        nUpdatesBag ← SampleFrom(Poisson(1))
        for k ∈ 1 : nUpdatesBag do
            B.rewSum[m_i] ← B.rewSum[m_i] + ρ_i
            Add (B.rt[m_i], r_i)
            Add (B.work[m_i], w_i)
            B.numObs[m_i] ← B.numObs[m_i] + 1
```

5.3.3 Response Time Bounds

In this section, we provide upper and lower response time bounds for a job scheduled with different number of cores initially. These are used in the partial feedback MAB, to obtain reward estimates for unexplored arms by using observations in explored arms, as outlined in Section 5.3.4. This is done with a counterfactual reasoning: for a job scheduled with m_a assignment and observed response time and work, what would the response time have been if it was instead scheduled with m_b assignment? In this way, we can avoid unnecessary exploration of non-optimal arms.

First, we derive a bound for the response time difference due to a lower initial number of cores having a smaller virtual deadline and M assignment at an earlier point. Next, we derive a bound on the response time difference resulting from different initial core allocations, based on Graham (1969). We also provide response time bounds based on the job's work, since that does not change with the choice of allocation.

For example, consider a parallel synchronous task with two segments, each segment containing four threads. For simplicity, we assume that the computation time required for each thread is deterministic. In Fig. 4 we illustrate the scheduling or this task with initial core assignment of $m = 2$ to the left or $m = 3$ to the right. From Eq. (5) we conclude that $V(2) < V(3)$. As observed in Fig. 4, this can lead to a longer response time with a higher number of cores assigned initially, because all $M = 4$ cores are

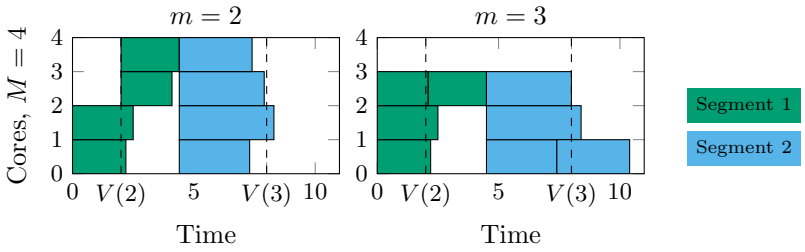

Fig. 4 A higher initial m resulting in a longer response time.

assigned at a later point in time. The response time difference is bounded by the difference of the virtual deadlines.

We will show in the following that Eqs. (7)–(9) hold, under the scheduling described in Section 4.2.

$$m_1 < m_2 \quad \Rightarrow \quad r_i(m_2) \leq r_i(m_1) + V(m_2) - V(m_1) \tag{7}$$

$$((r_i(m_1) < V(m_1)) \vee (r_i(m_2) < V(m_1))) \wedge (m_1 < m_2) \quad \Rightarrow \quad r_i(m_2) \leq r_i(m_1) \tag{8}$$

$$(r_i(m_2) > V(m_1)) \wedge (m_1 < m_2) \quad \Rightarrow \quad r_i(m_1) \geq V(m_1) \tag{9}$$

Eq. (7) implies that the response time with a higher initial assignment m_2 is at most $(V(m_2) - V(m_1))$ longer than the response time with a lower initial assignment m_1. In the example of Fig. 4, the response time with the scheduling on the right side can be at most $V(3) - V(2)$ longer than the response time with the scheduling on the left side.

Eq. (8) implies that if the response time with a lower initial assignment m_1 is shorter than the virtual deadline $V(m_1)$, then the response time with a higher initial assignment m_2 is at most the same as with m_1. Compared to Eq. (7), Eq. (8) provides a tighter upper bound for r_i with the counterfactual m_2 assignment, for the case when $r_i(m_1) < V(m_1)$. Eq. (8) also implies that if the response time with a higher initial assignment m_2 is shorter than the virtual deadline of a lower assignment $V(m_1)$, then the response time with a lower initial assignment m_1 is at least the same as with m_2. This gives a tighter lower bound than Eq. (7) for r_i with the counterfactual m_1 assignment, for the case when $r_i(m_2) < V(m_1)$.

Eq. (9) implies that if the response time with a higher initial assignment m_2 is longer than the virtual deadline of a lower assignment $V(m_1)$, then the response time with m_1 must be at least $V(m_1)$. Compared to Eq. (7), Eq. (9) provides a tighter lower bound for r_i with the counterfactual m_1 assignment, for the case when $V(m_1) < r_i(m_2) < V(m_2)$.

In the following, recall the definition of partial schedule S_i and profile Q_i of a job J_i in Definition 1, and the ahead relation of profiles from Definition 2.

Assumption 1. *The task model is as outlined in Section 4.*

Assumption 2. *Scheduling is with list scheduling, with the same list order for all arms and not modified during execution.*

Assumption 3. *A job J_i is scheduled with two different schedules, S_i' and S_i'', resulting in two different profiles Q_i' and Q_i''. At time t_0, profile $Q_i''(t_0)$ is ahead of $Q_i'(t_0)$, $Q_i''(t_0) \leq Q_i'(t_0)$. From time t_0 to time $t_1 > t_0$, S_i' assigns m_1 cores to J_i, and S_i'' assigns m_2 cores, $m_1 < m_2$.*

Assumption 4. *A job J_i is scheduled with two initial core assignments m_1 and m_2, $m_1 < m_2$. At the corresponding $V(m_1)$ and $V(m_2)$, respectively, the job is assigned M cores.*

Let us consider a single job J_i under different schedules.

Assume that job J_i starts execution at time 0 with m cores, resulting in a partial schedule $S_i(t)$ and profile $Q_i(t)$ at time t. Let $(e_1, e_2, \ldots, e_U)$ denote the execution times of each thread of J_i ordered in the list scheduling order of the threads.

Since thread processing times are unknown, the remaining processing times in the profile are also unknown. However the following observations on the profile $Q_i(t)$ hold:

- When a thread completes then its remaining processing time is 0: if thread a has completed execution then $q_a(t) = 0$;
- Since threads are processed using list scheduling the following holds: if $q_a(t) < e_a$ then all threads that precede a in list scheduling have started execution and, therefore, $q_b(t) < e_b$ for all $b, b < a$;
- If at time t partial schedule $S_i(t)$ executes a thread of segment H and thread a belongs to a segment that precedes H then a has completed execution, and, therefore, $q_a(t) = 0$;
- If at time t thread a has been processed for at least the same time interval in schedule $S_i(t)'$ compared to the schedule $S_i''(t)$ then $q_a'(t) \leq q_a''(t)$.

We now compare two different partial schedules $S_i'(t)$ and $S_i''(t)$ of J_i with different number of cores. Lemma 1 below will be applied in Proposition 1 to the part of the task execution taking place before $V(2)$ for both core allocations illustrated in Fig. 4.

Let $Q'(t) = (q_1'(t), q_2'(t), \ldots, q_U'(t))$ and $Q''(t) = (q_1''(t), q_2''(t), \ldots, q_U''(t))$ denote the profiles representing the remaining processing times when m_1 and m_2, $m_1 < m_2$, cores are used in in $S'(t)$ and $S''(t)$ respectively.

Lemma 1. *We use Assumptions 1–3. At any time $t_0 \leq t \leq t_1$, $Q_i'(t) \geq Q_i''(t)$, i.e. $Q_i''(t)$ is ahead of $Q_i'(t)$.*

Proof. The proof is by induction. The statement is true at time $t = t_0$.

Assume that the statement is true at time $t-1$ and let A be the set of threads processed at t in schedule $S_i'(t)$ (that uses m_1 cores). If all threads in A are also processed at t in schedule $S_i''(t)$, then the claim is true at t by the inductive hypothesis.

Now assume that there exists a thread a, $a \in A$, that is not executed at t in S_i''; if thread a has already completed execution we have $q_a''(t) = 0$ and the claim holds at time t.

If thread a has not completed execution by time $t-1$ in S_i'', the segment executed at t in S_i'' must be the segment to which a belongs because of the assumption that $Q'(t-1) \geq Q''(t-1)$. Then, at least m_2 threads should precede a in the list ordering that has not been completed. By the inductive hypothesis, these threads are also unfinished at time $t-1$ in the schedule that uses m_1 cores; since $m_2 > m_1$ this

contradicts the assumption that a is executed at t when m_1 cores are available and the claim is true at time t. This completes the proof of the inductive step. □

Proposition 1. *We use Assumptions 1, 2 and 4. At any time* $0 \leq t \leq V(m_1)$, $Q'_i(t) \geq Q''_i(t)$, *i.e.* $Q''_i(t)$ *is ahead of* $Q'_i(t)$.

Proof. Let $t_0 = 0$ and $t_1 = V(m_1)$. Then Proposition 1 follows from Lemma 1 because $Q'_i(0)$ and $Q''_i(0)$ are identical. □

Eqs. (8) and (9) follow directly from Proposition 1.

We now consider two partial schedules $S'_i(t')$ and $S''_i(t'')$ obtained using different number of cores and execution for different time intervals.

We are interested in comparing the completion times of $S'_i(t')$ and $S''_i(t'')$ when M of cores are used to complete $S'_i(t')$ ($S''_i(t'')$) after time t' (t''). Proposition 2 will be applied to the part of the task execution taking place after $V(2)$ for $m = 2$ and after $V(3)$ for $m = 3$ in Fig. 4. $S'_i(t')$ ($S''_i(t'')$) uses M cores after time t' (t''). Let r'_i (r''_i) be the completion time of J_i under schedule $S'_i(t')$ ($S''_i(t'')$). The following Proposition shows that, if $Q''_i(t'')$, the profile of $S''_i(t'')$ at time t'', is ahead of $Q'_i(t')$ then $r''_i - t'' \leq r'_i - t'$, i.e. the time required to complete S''_i is not greater than the time necessary to complete S'_i when the same number of cores M is used.

Proposition 2. *We use Assumptions 1, 2 and 4. If* $t' \geq V(m_1)$, $t'' \geq V(m_2)$ *and* $Q''_i(t'')$, *the profile of* $S''_i(t'')$, *is ahead of* $Q'_i(t')$, *the profile of* $S'_i(t')$, *then* $r''_i - t'' \leq r'_i - t'$.

Proof. Assume that $t' \leq t''$; (the proof for $t'' < t'$ is analogous).

It is sufficient to prove that, at any time instant t, $t \geq t'$, $Q'_i(t) \geq Q''_i(t + (t'' - t'))$. We prove it by induction. The statement is true at $t = t'$. We now assume it is true for $t - 1$, and we will prove that it holds at t.

Let A', A'' be the set of threads that are processed at t in schedule $S'_i(t)$ and at $t + (t'' - t')$ in $S''_i(t + (t'' - t'))$ respectively, and let I'' be the segment to which threads in A'' belong. If $A' = A''$, then the proposition is true at time t by the inductive hypothesis.

Now, assume a thread a, $a \in A' - A''$ exists. If a has already completed execution in $S''_i(t - 1 + (t'' - t'))$ at $t - 1 + (t'' - t')$ then its remaining processing time is 0 and the proposition holds at time t. We now observe that if $q''_a(t - 1 + (t'' - t')) > 0$, a must belong to segment I''. In fact, if a belongs to a segment that precedes I'' then l has completed execution in $S''_i(t - 1 + (t'' - t'))$ and this implies that $q''_a(t - 1 + (t'' - t')) = 0$. Since $Q'_i(t - 1) \geq Q''_i(t - 1 + (t'' - t'))$ a cannot belong to a segment that succeeds I''.

If a has not completed execution by time $t - 1 + (t'' - t')$, it follows that there are M uncompleted threads in segment I'' at time $t + (t'' - t')$ that precede a in list ordering. By the inductive hypothesis, these threads have not completed execution in schedule $S'_i(t - 1)$, thus contradicting the assumption that a is scheduled at t in $S'_i(t)$.

This concludes the inductive step and the proof. □

Now, we are ready to prove Theorem 1, claiming that a potential increase in a job's response time when assigning a higher number of cores initially is bounded by the difference of the virtual deadlines.

Theorem 1. *We use Assumptions 1, 2 and 4. Then* $r_i(m_2) \leq r_i(m_1) + V(m_2) - V(m_1)$.

Proof. Let $S'_i(V(m_1))$ ($S''_i(V(m_1))$) be the partial schedule at time $V(m_1)$ when m_1 (m_2) cores are used and let $Q'_i(V(m_1))$ ($Q''_i(V(m_1))$) be the profile at time $V(m_1)$. Since $m_2 > m_1$ Proposition 1 implies that $Q'_i(V(m_1)) \geq Q''(V_i(m_1))$.

We now observe that as $V(m_2) > V(m_1)$, $Q''_i(V(m_1)) \geq Q''_i(V(m_2))$.

By transitivity, it follows that

$$Q'_i(V(m_1)) \geq Q''_i(V(m_1)) \geq Q''_i(V(m_2))$$

By Proposition 2 $r_i(m_2) - V(m_2) \leq r_i(m_1) - V(m_1)$ follows. □

If we have a response time of a job with one m allocation, Eqs. (7)–(9) provide bounds for one end or the range of possible response times with another allocation. For the other end of the range, we will show that Eqs. (10)–(13) hold.

$$(r_i(m_2) \leq V(m_1)) \vee (r_i(m_1) \leq V(m_1)) \Rightarrow r_i(m_2) \geq r_i(m_1) \cdot \frac{m_1}{m_2 + m_1 - 1} \tag{10}$$

$$r_i(m_1) > V(m_1) \Rightarrow r_i(m_2) \geq V(m_1) \cdot \frac{m_1}{m_2 + m_1 - 1} \tag{11}$$

$$r_i(m_1) > V(m_1) \cdot (m_1 + m_2 - 1)/m_1 \Rightarrow r_i(m_2) > V(m_1) \tag{12}$$

$$r_i(m_2) > V(m_1) \Rightarrow r_i(m_2) \geq r_i(m_1) - V(m_1) \cdot \frac{m_2 - 1}{m_1} \tag{13}$$

Eq. (10) provides a lower bound for r_i with the counterfactual higher initial assignment m_2, when we observed $r_i(m_1) \leq V(m_1)$. It also provides an upper bound for r_i with the counterfactual lower initial assignment m_1, when we observed $r_i(m_2) \leq V(m_1)$.

Eq. (11) provides a lower bound for r_i with the counterfactual higher initial assignment m_2, when we observed $r_i(m_1) > V(m_1)$. Compared to Eq. (11), Eq. (12) provides a tighter lower bound for r_i with the counterfactual higher initial assignment m_2, when we observed $r_i(m_1) > V(m_1) \cdot (m_1 + m_2 - 1)/m_1$.

Eq. (13) provides an upper bound for r_i with the counterfactual lower initial assignment m_1, when we observed $r(m_2) > V(m_1)$.

Eq. (10) is simply a reformulation of Theorem 1 from Graham (1969), which proves a bound for the response time ratio for a DAG scheduled with list ordering and m_1 or m_2 cores. Observing $r_i(m_1) \leq V(m_1)$, Eq. (10) bounds the potential response time reduction with a larger number of cores m_2. From Eq. (8) we have $r_i(m_2) \leq r_i(m_1)$, and the conditions from Graham (1969) are fulfilled up until $r_i(m_2)$. Observing $r_i(m_2) \leq V(m_1)$, Eq. (10) bounds the potential response time increase with a lower number of cores m_1. This bound still holds if M cores are assigned at $V(m_1)$. Eq. (11) follows because according to Theorem 1 from Graham (1969), the part of J_i completed at $V(m_1)$ with m_1 cores assigned can be completed earliest at $V(m_1) \cdot \frac{m_1}{m_2+m_1-1}$ with m_2 cores assigned.

We denote the response time of a job J with allocation m up until t from its arrival, thereafter M, with $r(J, m, t)$.

Theorem 2. *We use Assumptions 1, 2 and 4. Then Eq. (12) holds.*

Proof. The response time if scheduling J_i with m_1 cores until completion, $r(J_i, m_1, \infty)$, is compared to $r_i(m_1)$. Because the profiles are identical at $V(m_1)$, we use $t_0 = V(m_1)$ in Lemma 1, that gives $r(J_i, m_1, \infty) \geq r_i(m_1)$.

We compare $r(J_i, m_1, \infty)$ to $r(J_i, m_2, \infty)$, and Theorem 1 from Graham (1969) gives that $r(J_i, m_2, \infty) \cdot \frac{m_1+m_2-1}{m_1} \geq r(J_i, m_1, \infty) \geq r_i(m_1)$.

Inserting $r_i(m_1) > V(m_1) \cdot \frac{m_1+m_2-1}{m_1}$ gives $r(J_i, m_2, \infty) > V(m_1)$. At $V(m_1)$ profiles are identical for scheduling with m_2 until completion or until $V(m_2)$, so $r_i(m_2) > V(m_1)$ □

Theorem 3. *We use Assumptions 1, 2 and 4. Then Eq. (13) holds.*

Proof. The job J_i is split into two part-jobs denoted J_a and J_b. J_a is the threads and parts of threads completed at $V(m_1)$ when the job is scheduled upon m_2 cores. J_b is the remaining parts of threads and threads at this time.

Scheduling J_i with m_2 cores up until $V(m_2)$, and then with M cores is equivalent to scheduling J_a with m_2 cores, and immediately schedule J_b with m_2 cores up until $V(m_2) - V(m_1)$ and thereafter M. Therefore, we have:

$$r_i(m_2) = r(J_a, m_2, V(m_2)) + r(J_b, m_2, V(m_2) - V(m_1))$$

Clearly $r(J_a, m_2, V(m_2)) = V(m_1)$, so $r_i(m_2) = V(m_1) + r(J_b, m_2, V(m_2) - V(m_1))$.

From Proposition 1, $r(J_b, m_1, 0) \leq r(J_b, m_2, V(m_2) - V(m_1))$, giving:

$$r_i(m_2) \geq V(m_1) + r(J_b, m_1, 0)$$

Since the split is not done with m_1 assignment, we have:

$$r_i(m_1) \leq r(J_a, m_1, V(m_1)) + r(J_b, m_1, 0)$$

Combining these gives $r_i(m_2) \geq V(m_1) + r_i(m_1) - r(J_a, m_1, V(m_1))$.

We compare scheduling of J_a with m_1 cores until completion and scheduling it with m_1 cores until $V(m_1)$ and thereafter M cores. The profiles are identical at $V(m_1)$, and from Lemma 1 with $t_0 = V(m_1)$, $r(J_a, m_1, V(m_1)) \leq r(J_a, m_1, \infty)$.

Theorem 1 from Graham (1969) gives $r(J_a, m_1, \infty) \leq V(m_1)\frac{m_2+m_1-1}{m_1}$. Combining these results we have $r_i(m_2) \geq V(m_1) + r_i(m_1) - V(m_1)\frac{m_2+m_1-1}{m_1}$. □

Furthermore, we use the total computation time w_i of a given a job J_i to derive the following response time bounds that hold for all m. Eq. (14) states that the response time cannot be longer than the total computation time (the response time when J_i is scheduled on a single core). Eq. (15) states that the response time cannot be shorter than the time it takes to complete w_i if all assigned cores are busy from start to completion.

$$r_i \leq w_i, \forall m \tag{14}$$

$$r_i \geq \begin{cases} \frac{w_i}{m} & w_i \leq V(m) \cdot m \\ V(m) + \frac{w_i - V(m) \cdot m}{M} & w_i > V(m) \cdot m \end{cases} \tag{15}$$

5.3.4 Bag Mean Reward Estimates

The arm with the highest mean reward in a bag B is returned from the algorithm outlined in Algorithm 10. First, all valid arms are traversed from lowest to highest, to find the nearest lower arm with observations for every valid arm, if it exists. Second, the arms are traversed in the reverse order to find the nearest higher arm, if it exists. Then, a mean reward estimate is retrieved for each valid arm, and the arm with the highest estimate is returned.

The algorithm for estimating the mean reward of an arm m in a bag B is outlined in Algorithm 11. If there are observations from this arm in B, we simply take the mean observed reward in the arm as our estimate. However, if there are no observations, an observation from another arm is used to obtain the reward estimate. The closest lower or higher arm with observations in B is used as the source arm, with probability in relation to the number of observations each of these arms have. We know that in Algorithm 11 the B contains at least one observation for one arm, due to the check on Line 3 in Algorithm 8.

The reward estimate for a target arm from an observation in a source arm is obtained according to Algorithm 12 if the source arm m is lower than the target arm's. If the source arm has a higher m than the target arm, the reward estimate is obtained as described in Algorithm 13. In both these algorithms, an observation is randomly sampled from the source arm, and the response time and total work are retrieved. Now, we consider the possible response time range for this observation under the counterfactual scenario that the job was scheduled with the target arm, although it was in fact scheduled with the source arm. A response time range with the target arm is retrieved using the bounds derived in Section 5.3.3. A factor is uniformly sampled in [0,1], and used to determine where in the range the response time estimate for the target arm goes. A reward is calculated with m, the estimated response time and the sampled total work, and used as the mean reward estimate.

In Algorithm 12, the lower end of the range is obtained from Eqs. (10)–(12) and (15).The higher end of the range is obtained from Eqs. (7), (8) and (14). Comments are added in the pseudocode to relate lines to equations.

In Algorithm 13, the lower end of the target arm response time range is obtained from Eqs. (7)–(9) and (15). The higher end of the range is obtained from Eqs. (10), (13) and (14). Comments in the algorithm connect lines to equations.

5.3.5 Time and Space Complexity and Scalability

Time complexity of the core allocation: At each round, the MAB algorithm randomly chooses one bag. The estimated mean reward for each valid m is computed according to Algorithm 10 and Algorithm 11. If the arm has observations, the estimate is simply a division, with constant time complexity. Otherwise, the response time and work from another arm is used for the estimate (Algorithm 12 or Algorithm 13). These

Algorithm 10: Find the arm with the highest estimated mean reward in bag B.

Input: Bag B, min and max valid arm $minArm$, $maxArm$
Output: Valid arm with highest estimated mean reward for B

Function `HighestEstMeanRew`(B, $minArm$, $maxArm$)**:**
  $nearLo \leftarrow -1$
  $B.nearLoMap \leftarrow$ the empty map
  for m *in* $minArm : maxArm$ **do**
    if $B.numObs[m] > 0$ **then**
      $nearLo \leftarrow m$
    if $nearLo \geq 0$ **then**
      $B.nearLoMap[m] \leftarrow nearLo$
  $nearHi \leftarrow -1$
  $B.nearHiMap \leftarrow$ the empty map
  for m *in* $maxArm : minArm$ **do**
    if $B.numObs[m] > 0$ **then**
      $nearHi \leftarrow m$
    if $nearHi \geq 0$ **then**
      $B.nearHiMap[m] \leftarrow nearHi$
  $\rho_{high} \leftarrow$ `EstMeanRew`$(B, minArm)$
  $m_{highRew} \leftarrow minArm$
  for m *in* $minArm + 1 : maxArm$ **do**
    $\rho \leftarrow$ `EstMeanRew`(B, m)
    if $\rho > \rho_{high}$ **then**
      $\rho_{high} \leftarrow \rho$
      $m_{highRew} \leftarrow m$
  return $m_{highRew}$

consist of conditions and arithmetic operations, resulting in constant time. Finding the arm to sample from in Algorithm 11 has constant time complexity, leading to linear complexity with respect to Ω for the core selection process Algorithm 10.

Time complexity of the update: At each round, a number of copies of the reward along with the selected arm, observed work, and response time are added to the bags as outlined in Algorithm 9. Each update involves additions for the reward and number of observations, and array additions for the work and response time, all of these requiring constant time. The average number of additions at one round equals the number of bags, so the time complexity for the update is linear with respect to κ.

MAB time complexity: The time complexity for the core allocation and update in a round is $\mathcal{O}(\Omega + \kappa)$.

MAB space complexity: The set of observations in a round (reward, response time, and work) is stored in some bags. On average, each set of observations is stored in κ bags. For an horizon T, this leads to the space complexity $\mathcal{O}(\kappa \cdot T)$. For long horizons, it may be desirable to provide a maximum number of observations to keep in each bag's arms. In this case, with a maximum array size of L, the space required

Algorithm 11: Estimate mean reward of an arm m in bag B.

Input: Bag B, arm m
Output: Estimated mean reward $\rho(m)$ for B

Function `EstMeanRew`(B, m)**:**
  if $B.numObs[m] > 0$ **then**
    return $\frac{B.rewSum[m]}{B.numObs[m]}$
  $sumObs \leftarrow 0$
  if $\exists B.nearLo[m]$ **then**
    $loArm \leftarrow B.nearLo[m]$
    $sumObs \leftarrow sumObs + B.numObs[loArm]$
  if $\exists B.nearHi[m]$ **then**
    $hiArm \leftarrow B.nearHi[m]$
    $sumObs \leftarrow sumObs + B.numObs[hiArm]$
  $useLoArmProb \leftarrow 0$
  if $\exists B.nearLo[m]$ **then**
    $useLoArmProb \leftarrow \frac{B.numObs[loArm]}{sumObs}$
  $s \leftarrow$ `SampleFrom`$(\mathcal{U}(0,1))$
  if $s \leq useLoArmProb$ **then**
    return `SampleRewardFromLower`$(B, loArm, m)$
  else
    return `SampleRewardFromHigher`$(B, hiArm, m)$

is $\mathcal{O}(\kappa \cdot \Omega \cdot L)$. A MAB algorithm not using the response time bounds will require $\mathcal{O}(\kappa \cdot \Omega)$ space for the reward estimates.

Scalability: The approach considers one task, as is the case also in Papadopoulos et al. (2022). The cores available to the task may be a subset of cores on the system, so that M is lower than the total number of cores. Other tasks can be scheduled on the remaining cores. Further, there are two ways to schedule other tasks on the M available for τ. First, if the algorithm is restricted to a choice of cores $m^{\downarrow} \leq m \leq m^{\uparrow}$, then $M - m^{\uparrow}$ cores are available to other tasks until $V(m^{\downarrow})$. Second, if τ has a constrained deadline, $D < p$, cores are available to other tasks after D. Both these approaches may affect the reward function - the energy model described in Section 6 would need to consider the other tasks.

The number of threads in the task does not directly influence the time required for application of the partial feedback MAB algorithm. However, it is likely that a larger number of cores will be used for a task with more threads, and that the number of valid cores will also be greater. The number of valid cores affects the time and space complexity of the approach.

5.4 Returning To The Motivating Example

Let us return to the motivating example in Section 5.2 and compare the BES algorithm with our proposed MAB approach. For this purpose, we construct a reward function ρ as in Eq. (16). For a selected arm m and the observed response time r, $\rho = 1$ if the

Algorithm 12: Estimate reward in target arm from lower source arm sample.

Input: Bag B, source arm $srcArm$, target arm $tgtArm$
Output: Estimated reward $\rho(tgtArm)$ for B

Function `SampleRewardFromLower`(B, $srcArm$, $tgtArm$):
  $sampleIdx \leftarrow$ `GetRandomIndex`$(B, srcArm)$
  $rt \leftarrow B.rt[srcArm][sampleIdx]$
  $w \leftarrow B.work[srcArm][sampleIdx]$
  /* Eq. (10) */
  $loRange \leftarrow rt \frac{srcArm}{srcArm+tgtArm-1}$
  /* Eq. (12) */
  if $loRange > V(srcArm)$ **then**
    $loRange \leftarrow rt - V(srcArm) \cdot \frac{tgtArm-1}{srcArm}$
  else
    /* Eq. (11) */
    if $rt > V(srcArm)$ **then**
      $loRange \leftarrow V(srcArm) \cdot \frac{srcArm}{srcArm+tgtArm-1}$
  /* Eq. (15) */
  $loRangeWork \leftarrow \frac{w}{tgtArm}$
  if $loRangeWork > V(tgtArm)$ **then**
    $loRangeWork \leftarrow V(tgtArm) + \frac{w-V(tgtArm)\cdot tgtArm}{M}$
  if $loRange < loRangeWork$ **then**
    $loRange \leftarrow loRangeWork$
  /* Eq. (7) */
  $hiRange \leftarrow rt + V(tgtArm) - V(srcArm)$
  /* Eq. (8) */
  if $rt < V(srcArm)$ **then**
    $hiRange \leftarrow rt$
  /* Eq. (14) */
  if $hiRange > w$ **then**
    $hiRange \leftarrow w$
  $u \leftarrow$ `SampleFrom`$(\mathcal{U}(0,1))$
  $rtTgt \leftarrow loRange + u \cdot (hiRange - loRange)$
  return `Reward`$(rtTgt, w, tgtArm)$

response time is between $V(m-1)$ (or 0 for $m=1$) and $V(m)$. Otherwise $\rho = 0$.

$$\rho(r,m) = \begin{cases} 1 & r \leq V(1), m = 1 \\ 1 & V(m-1) < r \leq V(m), m > 1 \\ 0 & otherwise \end{cases} \tag{16}$$

We use the proposed MAB approach with this reward function, $\kappa = 50$ bags, and the stochastic task described in Section 5.2. The resulting core allocations over 500 rounds compared to using the BES are shown in Fig. 5. In the first round, $m = 3$ is

Algorithm 13: Estimate reward in target arm from higher source arm sample.

Input: Bag B, source arm $srcArm$, target arm $tgtArm$
Output: Estimated reward $\rho(tgtArm)$ for B

Function `SampleRewardFromHigher`(B, $srcArm$, $tgtArm$):
  $sampleIdx \leftarrow$ `GetRandomIndex`$(B, srcArm)$
  $rt \leftarrow B.rt[srcArm][sampleIdx]$
  $w \leftarrow B.work[srcArm][sampleIdx]$
  /* Eq. (7) */
  $loRange \leftarrow rt + V(tgtArm) - V(srcArm)$
  /* Eq. (8) */
  if $rt < V(tgtArm)$ **then**
    $loRange \leftarrow rt$
  else
    /* Eq. (9) */
    if $rt < V(srcArm)$ **then**
      $loRange \leftarrow V(tgtArm)$
  /* Eq. (15) */
  $loRangeWork \leftarrow \frac{w}{tgtArm}$
  if $loRangeWork > V(tgtArm)$ **then**
    $loRangeWork \leftarrow V(tgtArm) + \frac{w - V(tgtArm) \cdot tgtArm}{M}$
  /* Eq. (10) */
  $hiRange \leftarrow rt \cdot \frac{tgtArm + srcArm - 1}{tgtArm}$
  /* Eq. (13) */
  if $rt > V(tgtArm)$ **then**
    $hiRange \leftarrow V(tgtArm) \cdot \frac{srcArm - 1}{tgtArm} + rt$
  /* Eq. (14) */
  if $hiRange > w$ **then**
    $hiRange \leftarrow w$
  $u \leftarrow$ `SampleFrom`$(\mathcal{U}(0, 1))$
  $rtTgt \leftarrow loRange + u \cdot (hiRange - loRange)$
  return `Reward`$(rtTgt, w, tgtArm)$

randomly selected. In the next 10 rounds, arms in the range $5 - 8$ are chosen, with $m = 5$ selected 5 times, $m = 6$ selected 3 times, and $m = 7$ and $m = 8$ both selected once. After this point, $m = 6$ is chosen most of the time, and exploration is less frequent the more rounds we add. The response times and rewards for all m are calculated for each job, and the rewards are summed per m to find the highest mean reward. The arm $m = 6$ corresponds to the highest mean reward of 0.554. In Fig. 6, the rewards of the BES and MAB algorithms are shown. The reward in Eq. (16) is binary, so we show the average reward over each 20-round interval. The mean reward of $m = 6$ (the highest mean reward) is displayed in magenta. Another common way to evaluate MAB algorithms is the regret, Eq. (1). The difference between consistently choosing the arm with the highest expected reward and the algorithm choice is calculated over an

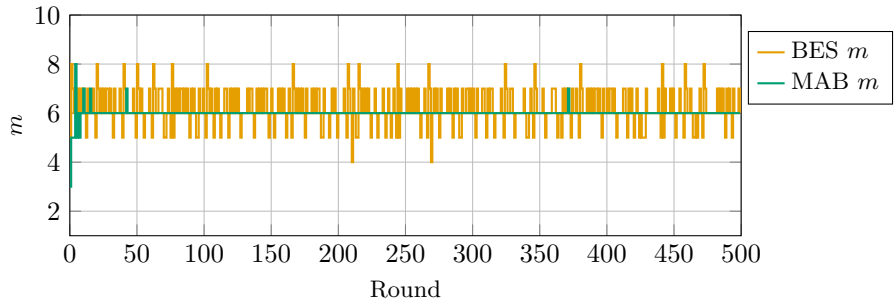

Fig. 5 Core allocation for the BES and MAB in the motivating example.

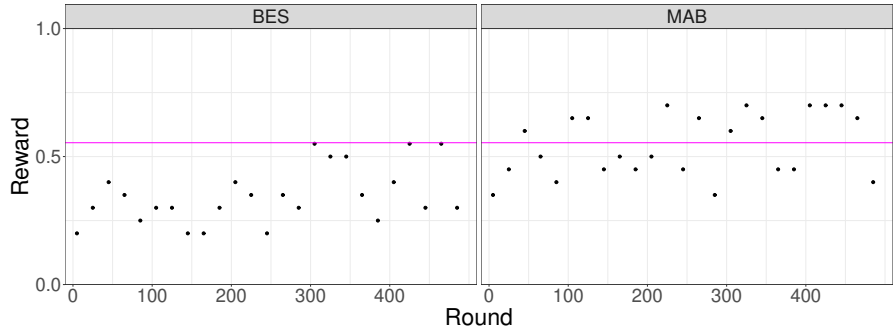

Fig. 6 Average rewards over each 20-round interval for the BES and MAB in the motivating example.

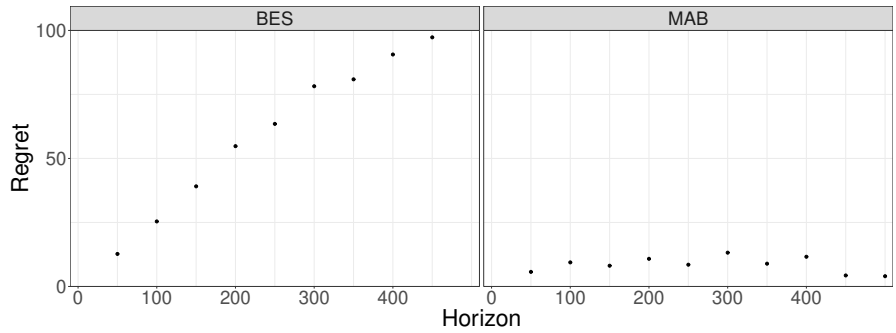

Fig. 7 Regrets over different horizons for the BES and MAB in the motivating example.

interval or horizon (T in Eq. (1)). The regret for different horizons is shown in Fig. 7. Here, it is clear that the BES regret grows linearly with the horizon, while the MAB regret grows much slower once the algorithm has learned which arm is likely the best.

6 Energy Consumption for Reward Function

In this section, we show how to use the MAB approach to find initial core allocation m that minimizes the energy consumption over time. This is done by using an energy model that estimates energy consumption of jobs within the reward function. We emphasize that the MAB approach can also be used with other optimization goals.

The energy model is based on an existing microarchitecture with sleep states and the task model and scheduling from Section 4. In Section 6.1, we discuss how the sleep state latency affects the schedulability condition.

Consider a job arriving when m cores are in the running (C0) or halt (C1) state, and $M - m$ cores are in a deep sleep state. The wake-up latency of the sleep state is denoted as Δ_{RS}. If the parallel synchronous task has not completed at $V - \Delta_{RS}$, the $M - m$ sleeping cores need to be woken up. This model is a simplification: in a real system, the choice of m needs to be done Δ_{RS} prior to the job arrival, to ensure that cores are woken up if needed. This implies that the arm selection should be performed prior to this point, when the observations from the latest task invocation may be only partially complete (i.e., either the latest task completed and its execution time is known, or it is still running and its execution time is known with a small uncertainty bounded by Δ_{RS}). This is ignored in what follows, as the effect on the MAB performance is minor. Sleep states for a full socket can save power by disabling the last-level cache. This could cause cold misses and longer response times. However, we would have the same concern when assigning all cores to the task at the virtual deadline. One of the assumptions stated in Papadopoulos et al. (2022) is that the task is compute-bound. The total execution time of the threads in a compute-bound task is related to the amount of performed computations, in contrast to a memory-bound task, where the execution time is related to the amount of accessed memory. As stated in Papadopoulos et al. (2022), this assumption is required for finding an optimal core assignment, but not for ensuring that the deadline is met.

The energy consumption is modeled according to the Sandy Bridge-EP (Xeon E5-2670) microarchitecture as described in Schöne et al. (2015), chosen because the power savings of the different states were documented here. Cores are distributed on n_s sockets with $n_{cs} = 8$ cores per socket. We let $M = n_s \cdot n_{cs}$, only this task is scheduled on n_s CPU sockets. Each core is in the running, halt, or sleeping state, corresponding to the CC0, CC1 and CC6 states. If all cores sharing a socket are in the sleep state, they enter the package sleep state, corresponding to PC6 in Schöne et al. (2015). We assume that the scheduler can control when a core goes to and leaves the sleeping state. Linux allows for enabling or disabling individual sleep states either directly, or by specifying the per-core latency tolerance[3]. System-wide latency tolerance is set to allow for the use of deep sleep states. The scheduler restricts the use of deep sleep states for cores allocated to the task by temporarily setting a lower per-core latency tolerance. We do not require the cores to run at the highest possible frequency when executing the work of τ, but at a fixed frequency taken into consideration when determining W, L and schedulability.

[3] https://wiki.linuxfoundation.org/realtime/documentation/howto/applications/cpuidle

P_R	P_H	P_S	P_{SS}	Δ_{RS}	$P_{\Delta RS}$
$7W$	$4W$	$2W$	$0.25W$	$40\mu s$	$7W$

Table 2 Core state power consumption and transition latency in the energy model.

The power consumption of a core in the running state is P_R. In the halt state, the power is P_H. In the sleep state, the power is P_S, and in the package sleep state the power is P_{SS}. Transitions between the halt and running state are instantaneous in the model, although a delay of less than $2\mu s$ was seen in Schöne et al. (2015). Transitions to and from the sleep state are associated with a latency of Δ_{RS}, both for entering and exiting the sleep state. The average power requirement during this latency period is $P_{\Delta RS}$, and no work is processed during this time. The power consumption values for the different states are presented in Table 2, along with the transition latency. The power consumption values are inferred from the power savings compared to the running state presented in Schöne et al. (2015), except for the average power consumption during the transition to the sleep state, $P_{\Delta RS}$, as it is not presented in Schöne et al. (2015). Therefore, we estimate it as follows. The residency times, that is the minimum time spent in the sleep state leading to power savings, are documented in the Linux drivers[4]. Based on these and the latencies from Schöne et al. (2015), we estimate $P_{\Delta RS} = 7W$, equal to the power consumption in the running state.

Now we can describe how we model the power consumption over time from the arrival of a job J_i to its deadline, given the response time r_i and the total work w_i. Depending on the relation of D, r_i and V we outline 6 cases, illustrated in Fig. 8. The left column, Eq. (17), is the case where the job is completed sufficiently early, so it is advantageous to temporarily move the m cores into the sleep state and wake them up again before the next job arrival. This is illustrated by all M cores colored blue with P_{SS} consumption for a portion of the time on the left column of Fig. 8. The right column, Eq. (18), is the case where it is more advantageous to keep the m cores in the halt state.

The condition for the first column is $D - r_i > \Delta_{RS} \frac{P_{\Delta RS} - P_S}{P_H - P_S}$

For each column in Fig. 8 (for Eqs. (17) and (18)), we have three cases, represented by the rows in Fig. 8 and the a), b), and c) cases of Eqs. (17) and (18). The first row, a), is the case when the job is completed in time, so we don't need to wake up the $M - m$ cores, $r_i < V - \Delta_{RS}$. Waking them up is illustrated as the red rectangles with $M - m$ cores having $P_{\Delta RS}$ consumption, in the second and third rows of Fig. 8. The second row, b), is when we wake them up, but it turns out they were not needed. $V - \Delta_{RS} \leq r_i \leq V$. The third row, c), is when all M cores are used for completion of the job, $r_i > V$. This is illustrated as all M colored orange, with P_R or P_H, for an interval in the third row of Fig. 8.

The full transition time Δ_{RS} is modeled at the time of going from the sleep state to the running state, although a smaller part of this is at the time of entering the sleep state.

The energy consumption $E_i(r_i, w_i, m)$ from the arrival until the deadline of J_i is calculated from the selected m, the response time r_i, and the total work w_i. The

[4] https://github.com/torvalds/linux/blob/master/drivers/idle/intel_idle.c

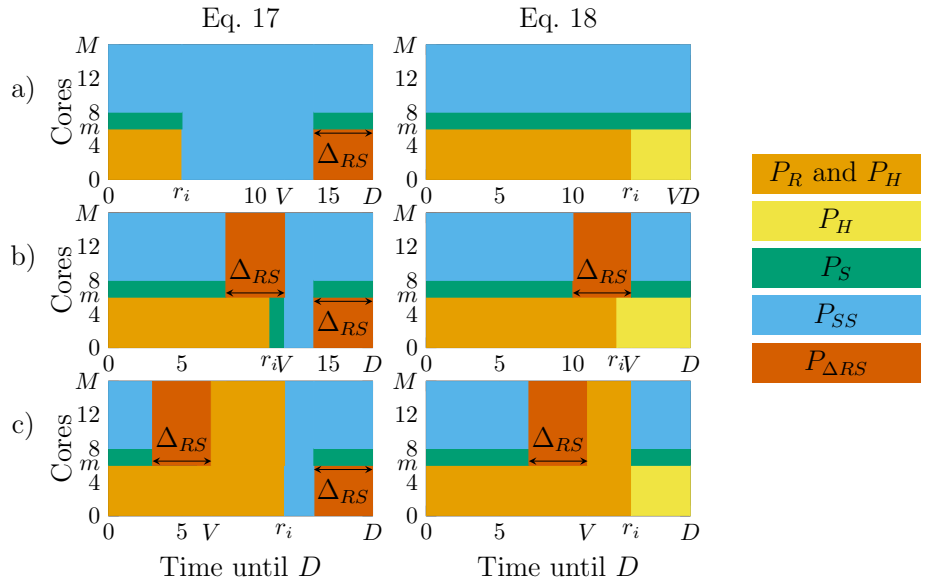

Fig. 8 Energy model illustration, $n_s = 2$, $n_{cs} = 8$, $M = 16$, $m = 6$.

direct energy consumption from the computation is $w_i \cdot P_R$ appearing in all cases of Eqs. (17) and (18). The rest of the energy consumption comes from halt and sleep states and transitions between states. A core is in the halt state when it is assigned to the task by the scheduler, but all threads in the current segment are executing in other cores. The total time of cores in the halt state from an orange area in Fig. 8 is the size of the area minus w_i, found in all cases of Eqs. (17) and (18). A core is also in the halt state when there is no benefit in moving to the sleep state after the job has completed, shown as yellow areas in the right column of Fig. 8, and appearing in Eq. (18). The energy consumption while transitioning to and from the sleep state is determined by the red areas in Fig. 8, and found in Eqs. (17), (18b) and (18c). The energy consumption in the sleep state is shown as blue areas (full sleeping sockets) or green areas in Fig. 8, and found in all cases of Eqs. (17) and (18). Let $c_{s(M-m)}$ denote the number of cores in $M - m$ that make up full sockets, $c_{s(M-m)} = n_{cs} \cdot \lfloor \frac{M-m}{n_{cs}} \rfloor$. Analogously $c_{sm} = n_{cs} \cdot \lfloor \frac{m}{n_{cs}} \rfloor$ denotes the number of cores in m that make up full sockets. Let $c_{r(M-m)} = M - m - c_{s(M-m)}$ and $c_{rm} = m - c_{sm}$ denote the remaining cores in $M - m$ and m that share a socket with non-sleeping cores.

The energy consumption is derived by multiplying the colored areas in Fig. 8 by the corresponding power consumption. For the orange areas, the total work is multiplied by P_R, and the orange area minus the total work is multiplied by P_H. The resulting equations are presented in Eqs. (17) and (18):

$$E_i = w_i \cdot P_R + (r_i \cdot m - w_i) \cdot P_H + \Delta_{RS} \cdot m \cdot P_{\Delta RS} + (r_i + \Delta_{RS}) \cdot (c_{s(M-m)} \cdot P_{SS} + c_{r(M-m)} \cdot P_S) + (D - r_i - \Delta_{RS}) \cdot M \cdot P_{SS} \quad (17a)$$

$$E_i = w_i \cdot P_R + (r_i \cdot m - w_i) \cdot P_H + (V - r_i)(c_{sm} \cdot P_{SS} + c_{rm} \cdot P_S) + \quad (17b)$$

$$
\begin{aligned}
& V \cdot (c_{s(M-m)} \cdot P_{SS} + c_{r(M-m)} \cdot P_S) + \Delta_{RS} \cdot M \cdot P_{\Delta RS} + (D - V - \Delta_{RS}) \cdot M \cdot P_{SS} \\
E_i = & w_i \cdot P_R + (r_i \cdot m + (r_i - V) \cdot (M - m) - w_i) \cdot P_H + \qquad (17c) \\
& V \cdot (c_{s(M-m)} \cdot P_{SS} + c_{r(M-m)} \cdot P_S) + \Delta_{RS} \cdot M \cdot P_{\Delta RS} + (D - r_i - \Delta_{RS}) \cdot M \cdot P_{SS}
\end{aligned}
$$

$$
\begin{aligned}
E_i = & w_i \cdot P_R + (r_i \cdot m - w_i) \cdot P_H + (D - r_i) \cdot m \cdot P_H + \qquad (18a) \\
& D \cdot (c_{s(M-m)} \cdot P_{SS} + c_{r(M-m)} \cdot P_S) \\
E_i = & w_i \cdot P_R + (r_i \cdot m - w_i) \cdot P_H + (D - r_i) \cdot m \cdot P_H + \qquad (18b) \\
& + (D - \Delta_{RS})(c_{s(M-m)} \cdot P_{SS} + c_{r(M-m)} \cdot P_S) + \Delta_{RS} \cdot (M - m) \cdot P_{\Delta RS} \\
E_i = & w_i \cdot P_R + (r_i \cdot m + (r_i - V) \cdot (M - m) - w_i) \cdot P_H + \qquad (18c) \\
& (V - \Delta_{RS} + D - r_i) \cdot (c_{s(M-m)} \cdot P_{SS} + c_{r(M-m)} \cdot P_S) + \Delta_{RS} \cdot (M - m) \cdot P_{\Delta RS}
\end{aligned}
$$

We note that the highest possible energy consumption is $E^{\uparrow} = W \cdot P_R + M \cdot \Delta_{RS} \cdot P_{\Delta RS} + ((D - \Delta_{RS}) \cdot M - W) \cdot P_H$ and the lowest is $E^{\downarrow} = M \cdot D \cdot P_{SS}$. We use this to construct a reward function where rewards are in the interval $[0, 1]$, according to Eq. (19). This function provides the reward 1 for $E^{\downarrow}$, with decreasing reward as the energy consumption increases. $E^{\uparrow}$ leads to reward 0. Compared to the binary reward function from the motivating example (Section 5.4), Eq. (19) allows for more fine-grained optimization.

$$
\rho(r_i, w_i, m) = \frac{E^{\uparrow} - E_i(r_i, w_i, m)}{E^{\uparrow} - E^{\downarrow}} \qquad (19)
$$

6.1 Schedulability Condition Considering Sleep State Latency

The $M - m$ cores that we need to use at the virtual deadline V may be in the sleep state, and we need to wake them up at $V - \Delta_{RS}$ to ensure they are available on time. This means that with this energy model, a non-negative $V(m)$ according to Eq. (5) is not sufficient for schedulability, but valid m must fulfill the condition in Eq. (20).

$$
V(m) = \frac{M \cdot (D - L) - (W - L)}{M - m} \geq \Delta_{RS}, m < M \qquad (20)
$$

7 Evaluation

The main goal of the evaluation is to assess the ability of the proposed MAB algorithm to find the highest-reward arm quickly, resulting in a low regret. We focus on the algorithm's ability to find the best arm, given a reward function depending on the selected arm, and the job's work and response time. The secondary goal is to evaluate how the approach scales to a larger number of cores and threads. By using simulation, a number of different task structures can be evaluated. In the simulation approach, we know that tasks comply with the stochastic parallel synchronous task model. The worst-case span and work are known. It also allows for comparison with the best arm, as the exact knowledge about the task structure allows us to simulate the reward for

Algorithm 14: NB_MAB core allocation at round i.

Input: set of κ bags $bags$, min and max valid arm $minArm$, $maxArm$.
Output: Core allocation m_i

```
Function CoreAllocatorNB(bags, minArm, maxArm):
    B ← SampleFrom(bags)
    if B.empty() then
        m_i ← SampleFrom(U(minArm, maxArm))
    else
        sampleArmsSet ← (m) s.t. B.numObs[m] = 0
        if ∃m ∉ sampleArmsSet then
            m_maxEst ← arg max_{m∉sampleArmsSet} EstMeanRew(B, m)
            sampleArmsSet ← sampleArmsSet ∪ (m_maxEst)
        m_i ← SampleFrom(sampleArmsSet)
```

all possible arm choices. The main limitations of the simulation approach are that we don't show the method's applicability to a real-world use case, and that we don't verify that the energy model is an accurate representation of the energy consumption. The evaluation[5] is performed with task structures selected to highlight factors that affect the algorithm's performance. .

7.1 Evaluated Algorithms

A comparison is performed between:

- **B_MAB**: The MAB algorithm outlined in Section 5.3.
- **BES**: The BES described in Section 5.2.2, adapted to use only valid m with the schedulability condition in Section 6.1.
- **GREEDY**: A greedy method selecting the lowest possible number of full sockets that fulfils the schedulability condition in Section 6.1.
- **NB_MAB**: A bandit feedback MAB algorithm that does not use the response time bounds to share information between arms.

We have not included a comparison with any DAG scheduling approach that requires knowledge about the DAG structure.

Both MABs use $\kappa = 50$, and reward functions based on the energy model described in Section 6, and specified in Eq. (19). In **NB_MAB**, arms without observations in a bag are selected with equal probability to the arm with the highest mean reward. That is, Algorithm 14 is used in place of Algorithm 8. The best arm with observations in the bag is added to a set with all arms without observations. An arm is selected at random from this set. Because Algorithm 11 is never called for an arm and bag without observations, the response time bounds are not used.

[5] Code used in the evaluation is available at https://github.com/annafriebe/ResourceManagementStochasticPSTasksBandits.

The **BES** in the evaluation is adapted to use only valid m according to the schedulability condition Eq. (20), and aim for the lowest m that results in response times below $V(m) - \Delta_{RS}$ instead of below $V(m)$.

The **GREEDY** method selects the lowest possible number of full sockets that fulfills the schedulability condition Eq. (20).

7.2 Task Structures and Energy Model

The task structures selected for the main evaluation are listed in Table 3. Task structures 1, 3, and 5 have the same number of segments but varying degrees of parallelism within the segments. Task structures 2, 4, and 6 have the same degree of parallelism but with varying numbers of segments. Task structure 7 has four low parallelism segments and two high parallelism segments as the second and fifth segments. Task structure 8 is similar to task structure 7 but with both high parallelism segments at the end.

TS	Description	s	u	TS	Description	s	u
1	Lo-u	5	(5, 5, 5, 5, 5)	2	Lo-s	4	(5, 10, 5, 10)
3	Mid-u	5	(10, 10, 10, 10, 10)	4	Mid-s	6	(5, 10, 5, 10, 5, 10)
5	Hi-u	5	(15, 15, 15, 15, 15)	6	Hi-s	8	(5, 10, 5, 10, 5, 10, 5, 10)
7	Hi-u-in	6	(6, 16, 6, 6, 16, 6)	8	Hi-u-end	6	(6, 6, 6, 6, 16, 16)

Table 3 The task structures in the main evaluation.

The task structures in Table 3 are evaluated with the energy model from Section 6 and Eq. (19), with $n_s = 2$ and $n_{cs} = 8$, resulting in $M = 16$.

To evaluate the scalability of the approach, three task structures with different thread parallelism are evaluated with three energy models with different numbers of sockets. The task structures are listed in Table 4, and they are evaluated with three versions of the energy model from Section 6 and Eq. (19). All energy models have $n_{cs} = 8$. The first is the same as in the main evaluation, $n_s = 2$ and $M = 16$, the second has $n_s = 4$ and $M = 32$, and the third has $n_s = 8$ and $M = 64$.

TS	Description	s	u
LS-1	LS-10	5	(10, 10, 10, 10, 10)
LS-2	LS-20	5	(20, 20, 20, 20, 20)
LS-3	LS-40	5	(40, 40, 40, 40, 40)

Table 4 The task structures used in the scalability evaluation.

The execution time e_{ijk} of thread k in segment j of J_i is generated from a beta distribution, $Beta(\alpha = 2, \beta = 5)$, scaled and translated according to a setting β_γ and a translation β_Δ. e_{ijk} is sampled from $\mathcal{E}_{jk} \sim \beta_{\Delta jk} \cdot (1 + \beta_\gamma \cdot Beta(\alpha = 2, \beta = 5))$. The setting β_γ is varied in the experiments to explore the effect of thread execution times varying to different degrees between different jobs in the same task realization. $\beta_{\Delta jk}$ is drawn from a uniform distribution with the same width ($50\mu s$) for each thread, $\mathcal{U}(\beta_{\Delta\downarrow}, \beta_{\Delta\downarrow} + 50)$. The starting point of the uniform distribution $\beta_{\Delta\downarrow}$ is calculated

according to Eq. (21) so that the expected value of thread execution times is the same ($150\mu s$) for all β_γ settings. This is to separate the effects of higher work and span from the effects of higher variance in the thread execution times within a realized sequence. The experiments generate tasks with $\beta_\gamma \in (0.1, 0.2, 0.4, 0.8, 1.6)$.

$$\beta_{\Delta\downarrow} = \left\lfloor \frac{150}{1 + \frac{2 \cdot \beta_\gamma}{7}} - \frac{50}{2} \right\rfloor \tag{21}$$

7.3 Performing the Evaluation

A task is generated for each task structure and β_γ setting. For each task, the worst case work W and span L are calculated as $W = \sum_{j=1}^{s} u_j \cdot (\beta_{\Delta\downarrow} + 50) \cdot (1 + \beta_\gamma)$ and $L = s \cdot (\beta_{\Delta\downarrow} + 50) \cdot (1 + \beta_\gamma)$. For each task structure, the maximum worst-case work and span over all β_γ settings are used to calculate virtual deadlines for all tasks.

The performance of the algorithms depends on the set deadline. A deadline set tightly compared to the schedulability condition leads to short virtual deadlines. This causes the BES to allocate more cores and affects the reward function for the MAB. Deadlines are generated as $D = \left(\frac{W+L}{M} + L + \Delta_{RS}\right) \cdot d_s$ from the schedulability condition given in Section 6.1. The factor d_s is randomly drawn from a uniform distribution $\mathcal{U}(1.25, 2.5)$. Every task is run with 20 deadline configurations, and the runs contain 2000 rounds.

In the evaluation, all methods are evaluated in parallel for the same task realization. At the same time, the reward and energy consumption according to the energy model are calculated for each possible assignment of m of all arriving jobs. The arm resulting in the highest reward for a job is referred to as the clairvoyant best arm (**OPT_C**). At the end of each 2000-job realization, the arm with the highest reward sum is retrieved as the best average arm (**OPT**). The arm with the highest reward is the arm with the lowest energy consumption, as $E^\uparrow$ and $E^\downarrow$ in Eq. (19) are fixed for a single realization.

7.4 Results and Discussion

In this section, we present and discuss the results from the main evaluation in Section 7.4.1, followed by the results from the scalability evaluation in Section 7.4.2.

7.4.1 Main Evaluation Results and Discussion

In Fig. 9, the regrets over different horizon lengths are displayed for the **B_MAB**, **BES**, **GREEDY**, and **NB_MAB** methods. Different deadline settings d_s are shown in different colors. InFig. 10, the same data is shown but with coloring of the different configurations of β_γ that control the computation time variation between different threads.

It is clear that the regret of the **BES** and **GREEDY** methods grow linearly with the horizon, although the slope of the **BES** regret may change at points when the interval is updated. The regrets for the MAB algorithms grow much slower once the likely best arms are learned. There is no case where **B_MAB** has higher regret than 10 or **NB_MAB** has higher regret than 15. It is also clear that in some cases,

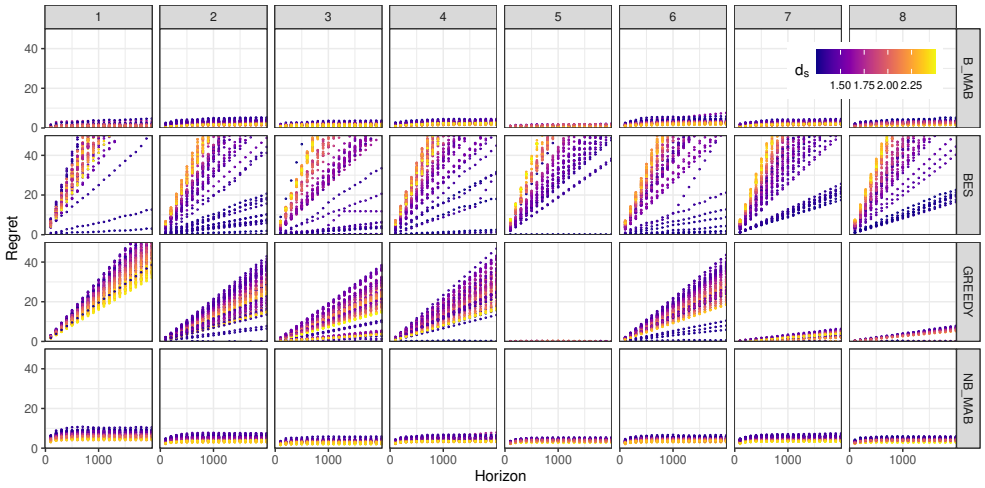

Fig. 9 Regrets for the methods over different horizon lengths for the task structures, with deadline configurations visualized.

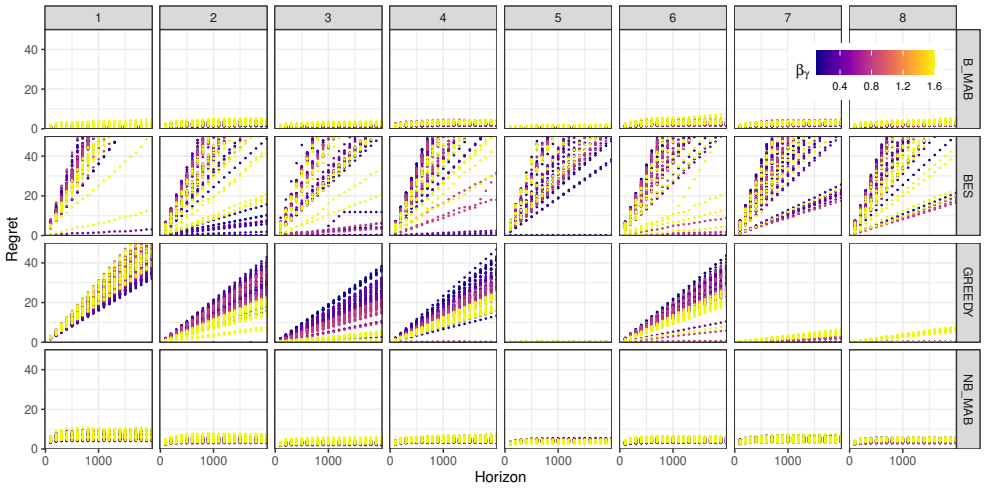

Fig. 10 Regrets for the methods over different horizon lengths for the task structures, with computation time variation configuration visualized.

BES and **GREEDY** have very low regret. These cases correlate with particular task structures and deadline factors, and for the **BES** case also low thread computation time variation. **BES** has lower regret for tighter deadlines, while the opposite is true for the MAB methods. **GREEDY** outperforms the other methods for task structure 5, with a large degree of parallelism. The MABs perform some initial exploration that comes with a cost to the regret, which is more pronounced for **NB_MAB**. Both MABs reliably find arms providing low regret in the long run for all tested task structures and deadline configurations.

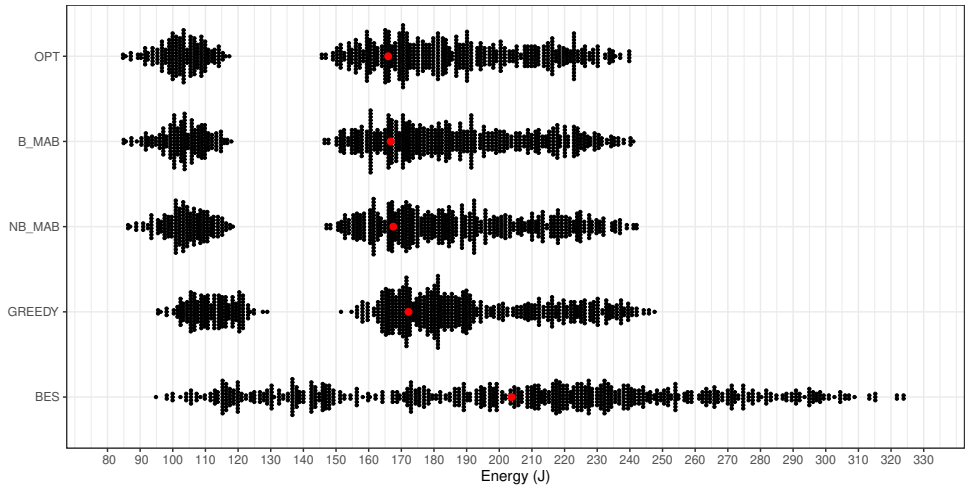

Fig. 11 Energy consumption at 2000 rounds over all realizations for the different methods compared to the best average arm (**OPT**) for each realization.

The average regrets at step 2000 over all tasks and configurations are 18.9 for **GREEDY**, 98.9 for **BES**, 4.5 for **NB_MAB** and 2.1 for **B_MAB**. Using the response time bounds resulted in regrets less than half of those of **NB_MAB**. Due to the retrieved response time bounds, arms that cannot be optimal are not explored, reducing the regret.

To interpret the performance in terms of energy consumption, the total energy consumption at 2000 rounds with the energy model in Section 6 is calculated for each realization and each of the methods and for the arm that is best on average for the realization. The results are visualized in Fig. 11, where each black dot represents a realization in a bin, where bins have width $1J$. The red dots are the average energy consumption over the realizations for each method, that are also shown in Table 5, along with the ratios with the consumption using the best-average arm. **B_MAB** is within 0.5% of the optimal consumption, **NB_MAB** is within 1%, **GREEDY** within 4%, and **BES** within 25%. Statistical tests are performed with the sign test (binomial test). The number of realizations where the energy consumption is higher for one method than another is compared to the binomial distribution of 800 (the number of realizations) tests with success probability 0.5, the expected distribution if one method would be equally good as the other. The results show that **B_MAB** has lower energy consumption than **NB_MAB**, **NB_MAB** lower than **GREEDY**, and **GREEDY** lower than **BES**, all with p-value $< 10^{-15}$. The 95%-confidence interval of the binomial test success probability is $[0.994, 1]$ when comparing **B_MAB** to **NB_MAB**, showing that for a specific realization, **B_MAB** almost always outperforms **NB_MAB** slightly due to reduced initial exploration. The binomial test success probability is $[0.633, 1]$ when comparing **B_MAB** to **GREEDY** and $[0.612, 1]$ when comparing **NB_MAB** to **GREEDY**. There are realizations where **GREEDY** is optimal or close to optimal and outperform the MAB methods. Comparing to **BES**, the binomial test success probability is $[0.968, 1]$ for **B_MAB** and $[0.965, 1]$ for **NB_MAB**.

	OPT	**B_MAB**	**NB_MAB**	**GREEDY**	**BES**
Mean energy consumption [J]	166.0	166.7	167.5	172.2	203.8
Ratio with **OPT**	1	1.004	1.009	1.037	1.228

Table 5 Average energy consumption at 2000 rounds over all realizations for the different methods compared to the best average arm (**OPT**) for each realization.

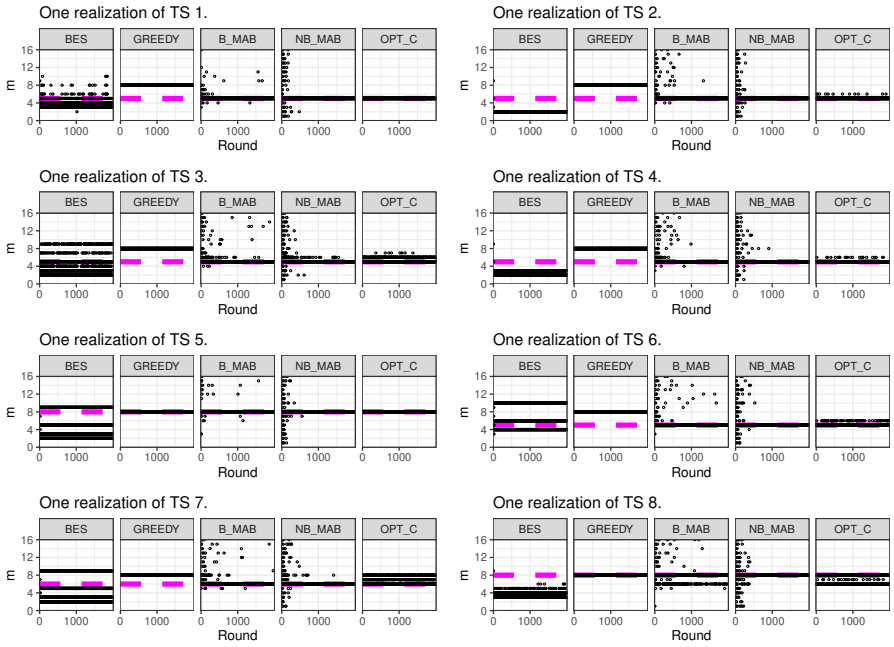

Fig. 12 Examples of arm selections for a few realizations along with the clairvoyant best arm for each job (**OPT_C**). The best average arm is dashed.

In Fig. 12, arm selections from one realization of each task structure are shown, along with the clairvoyant best choice **OPT_C** for each job. The average best arm **OPT** in the realization is shown as dashed magenta color. The **GREEDY** method finds the best allocation in the realizations shown for task structures 5 and 8. The **BES** method oscillates between different allocations. In the realizations for task structures 1, 3, and 6 these are near the optimal average choice. **B_MAB** has a lower amount of exploration than **NB_MAB**. For example the highest allocations in the realization of task structure 1 and the lowest allocations in the realizations of all task structures are almost never explored for **B_MAB**. Exploration for **B_MAB** occurs at later times, when **NB_MAB** no longer explores.

It is worth noting that there is a tradeoff between the simplicity of the **GREEDY** method and the performance of the MAB algorithms. Although there is a linear regret observed with the **GREEDY** method, the core assignment is done once, and then there is no energy required to compute it in further rounds. The other algorithms perform some computations at each round to determine the core assignment, resulting in

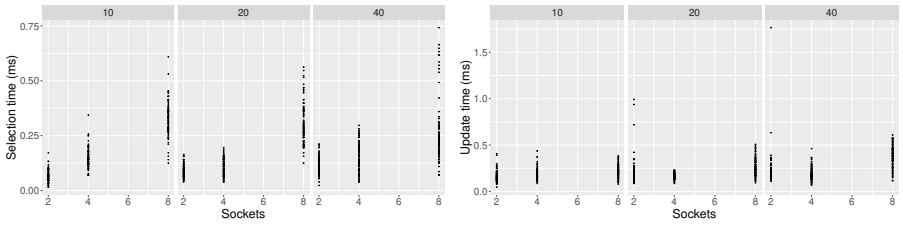

Fig. 13 Time for the partial MAB selection and update process with increasing number of sockets and cores.

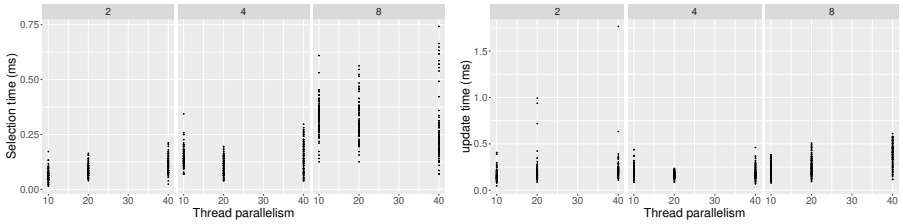

Fig. 14 Time for the partial MAB selection and update process with increasing number of threads.

energy consumption that is linear with the number of rounds. The expected reduction in energy consumption for executing the tasks needs to outweigh the energy consumed for the core assignment algorithms. In the evaluated task structures, the energy savings per round compared to **GREEDY** are in the order of mJ. With energy per instruction in the order of pJ to nJ (Vasilakis, 2015), the energy savings are likely to outweigh the energy costs for the core assignment algorithms.

7.4.2 Scalability evaluation results and discussion

In the scalability evaluation, the times for the partial feedback MAB selection and update procedures are logged. The algorithms are implemented in Python without optimization, so the exact times should not be emphasized, but trends in relation to the number of threads and cores are shown. In Fig. 13, the logged times are shown with increasing number of sockets and cores. Task structures are separated into facets labeled with the thread parallelism. As shown in the analysis in Section 5.3.5, the time for the selection increases linearly with the number of valid cores. In Fig. 14, the logged times are shown with increasing number of threads. Energy models with different numbers of sockets are separated into facets. The task parallelism does not affect the time required for partial feedback MAB selection or update.

The regrets over different horizon lengths with the evaluated methods are shown for the task structures in the scalability evaluation. In Fig. 15, regrets are shown for the 2-socket energy model, in Fig. 16 for the 4-socket energy model, and in Fig. 17 for the 8-socket energy model. From these figures, it is clear that the lower regrets with the partial feedback MAB approach remain, also for energy models with larger numbers of cores.

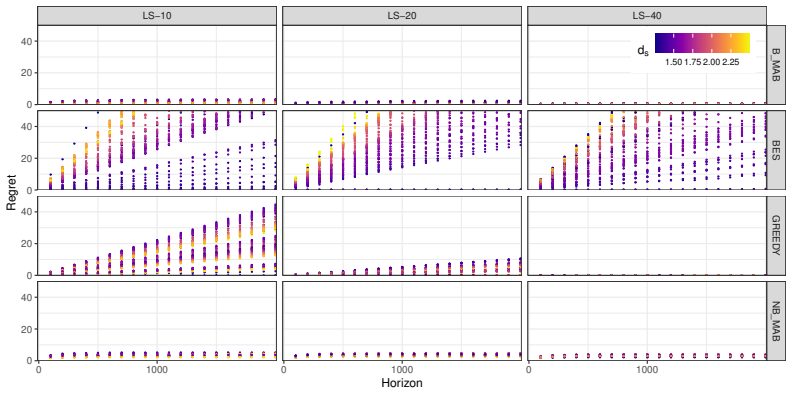

Fig. 15 Regrets for the methods over different horizon lengths for the scalability evaluation task structures with the 2-socket energy model.

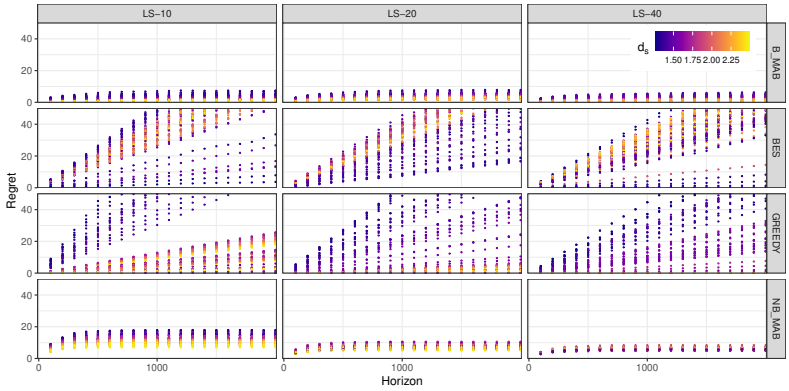

Fig. 16 Regrets for the methods over different horizon lengths for the scalability evaluation task structures with the 4-socket energy model.

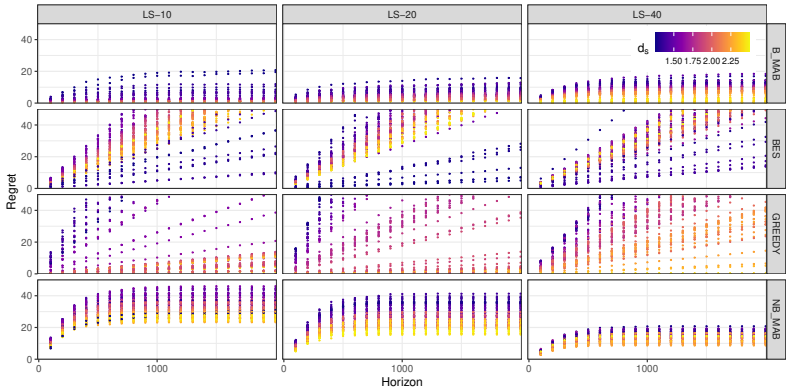

Fig. 17 Regrets for the methods over different horizon lengths for the scalability evaluation task structures with the 8-socket energy model.

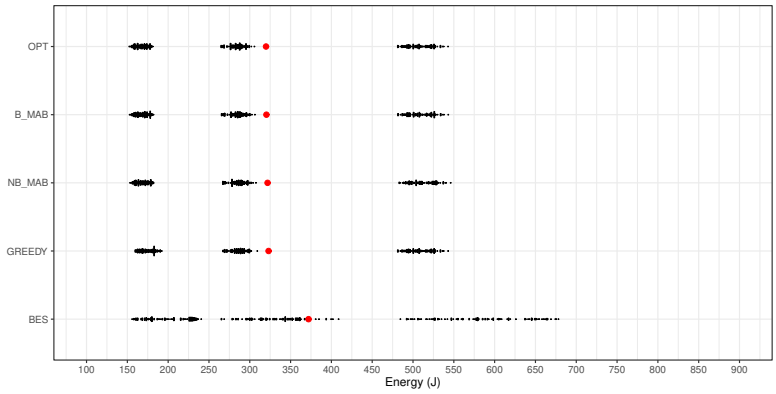

Fig. 18 Energy consumption at 2000 rounds for each realization of the scalability evaluation task structures with the different methods and the 2-socket energy model.

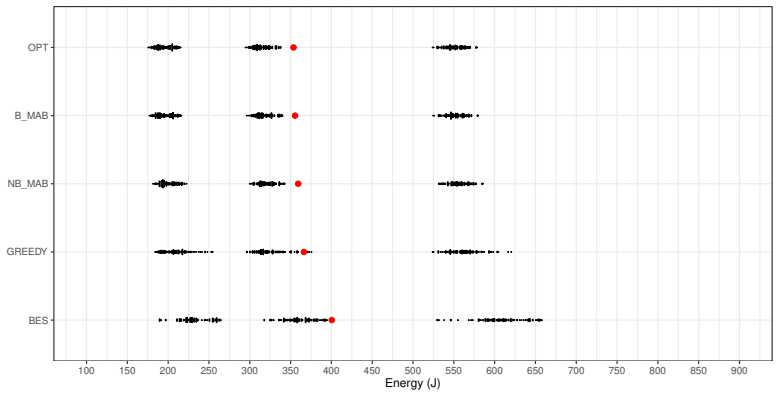

Fig. 19 Energy consumption at 2000 rounds for each realization of the scalability evaluation task structures with the different methods and the 4-socket energy model.

The total energy consumption at 2000 rounds with the energy model in Section 6 is calculated for each realization and shown to interpret the results in terms of energy consumption. In Fig. 18 the results are shown for the 2-socket energy model, in Fig. 19 for the 4-socket energy model, and in Fig. 20 for the 8-socket energy model. Comparing **B_MAB** to **GREEDY** for the evaluated task structures, we observe a greater reduction in energy consumption with an increasing number of cores. This is also the case when comparing **B_MAB** to **NB_MAB**. The results indicate that the increased computation required for a larger number of cores is likely to pay off by reducing the energy consumption.

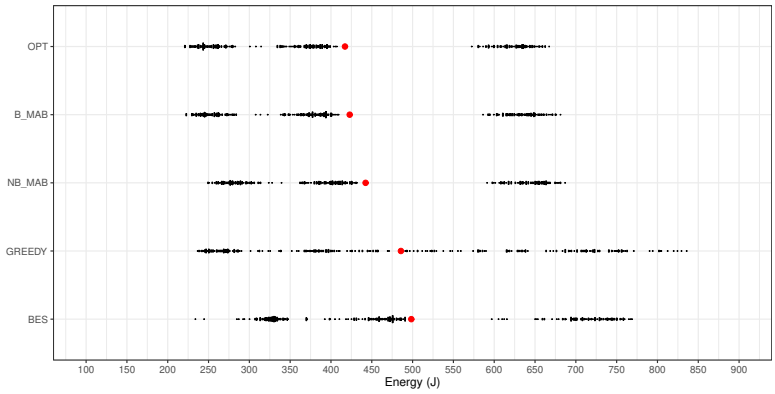

Fig. 20 Energy consumption at 2000 rounds for each realization of the scalability evaluation task structures with the different methods and the 8-socket energy model.

8 Conclusion And Future Work

This paper has integrated hard real-time constraints with an MAB resource management approach optimizing for the average case. Relying on previous work Papadopoulos et al. (2022) to ensure that deadlines are met, an MAB approach is evaluated for assigning a suitable number of cores to a Stochastic Parallel Synchronous Task. A partial feedback MAB approach is proposed, utilizing response time bounds to obtain information for unexplored arms. The MAB approach has two main advantages compared to the methods evaluated in Papadopoulos et al. (2022). First, the MAB algorithm considers all observations, compared to the most recent observation only. Second, the reward function is decoupled from the arm selection, resulting in a more versatile method. In the evaluation, the MAB approach is compared to the BES from Papadopoulos et al. (2022), to a greedy method, and to a bandit feedback MAB not using response time bounds, for eight selected task structures over different settings for thread execution time variance and deadlines. Both MABs reliably find arm choices with low regrets in the long term for all task structures and settings, while the BES and greedy methods have low regret for certain combinations of task structure and deadline configuration. Using the response time bounds in a partial feedback MAB decreases the amount of initial exploration needed compared to the bandit feedback MAB.

The findings above show that an MAB approach is useful for resource management with optimization for the average behavior, can be integrated in a hard real-time context, and that the use of response time bounds for partial feedback improves the performance.

In future work, MAB integration in other real-time use cases will be explored. It would be interesting to investigate the case where the reward dependence on the response, work, and arm choice is unknown, for example, a reward taken from a power measurement. For systems with changing reward distributions, such as the tasks switching between different DAG structures in Papadopoulos et al. (2022), CMAB or restless bandit approaches could be explored.

References

Antoniou, G., Bartolini, D., Volos, H., Kleanthous, M., Wang, Z., Kalaitzidis, K., Rollet, T., Li, Z., Mutlu, O., Sazeides, Y., et al.: Agile C-states: a core C-state architecture for latency critical applications optimizing both transition and cold-start latency. ACM Transactions on Architecture and Code Optimization (2024)

Abeni, L., Cucinotta, T.: Adaptive partitioning of real-time tasks on multiple processors. In: Proceedings of the 35th Annual ACM Symposium on Applied Computing. SAC '20, pp. 572–579. Association for Computing Machinery, New York, NY, USA (2020). https://doi.org/10.1145/3341105.3373937 . https://doi.org/10.1145/3341105.3373937

Aydin, H., Melhem, R., Mosse, D., Mejia-Alvarez, P.: Determining optimal processor speeds for periodic real-time tasks with different power characteristics. In: Proceedings 13th Euromicro Conference on Real-Time Systems, pp. 225–232 (2001). IEEE

Bietti, A., Agarwal, A., Langford, J.: A contextual bandit bake-off. Journal of Machine Learning Research **22**(133), 1–49 (2021)

Burns, A., Baruah, S.K., *et al.*: Sustainability in real-time scheduling. J. Comput. Sci. Eng. **2**(1), 74–97 (2008)

Baruah, S., Bonifaci, V., Marchetti-Spaccamela, A., Stougie, L., Wiese, A.: A generalized parallel task model for recurrent real-time processes. In: 2012 IEEE 33rd Real-Time Systems Symposium, pp. 63–72 (2012). IEEE

Borkar, V.S., Kasbekar, G.S., Pattathil, S., Shetty, P.Y.: Opportunistic scheduling as restless bandits. IEEE Transactions on Control of Network Systems **5**(4), 1952–1961 (2017)

Bambagini, M., Marinoni, M., Aydin, H., Buttazzo, G.: Energy-aware scheduling for real-time systems: A survey. ACM Transactions on Embedded Computing Systems (TECS) **15**(1), 1–34 (2016)

Bouneffouf, D., Rish, I., Aggarwal, C.: Survey on applications of multi-armed and contextual bandits. In: 2020 IEEE Congress on Evolutionary Computation (CEC), pp. 1–8 (2020). IEEE

Chou, C.-H., Bhuyan, L.N., Wong, D.: μDPM: Dynamic power management for the microsecond era. In: 2019 IEEE International Symposium on High Performance Computer Architecture (HPCA), pp. 120–132 (2019). IEEE

Chen, G., Liew, S.C., Shao, Y.: Uncertainty-of-information scheduling: A restless multiarmed bandit framework. IEEE Transactions on Information Theory **68**(9), 6151–6173 (2022)

Chandrakasan, A.P., Sheng, S., Brodersen, R.W.: Low-power CMOS digital design. IEICE Transactions on Electronics **75**(4), 371–382 (1992)

Eckles, D., Kaptein, M.: Bootstrap Thompson sampling and sequential decision problems in the behavioral sciences. Sage Open **9**(2), 2158244019851675 (2019)

Fonseca, J.C., Nélis, V., Raravi, G., Pinho, L.M.: A multi-DAG model for real-time parallel applications with conditional execution. In: Proceedings of the 30th Annual ACM Symposium on Applied Computing, pp. 1925–1932 (2015)

Graham, R.L.: Bounds on multiprocessing timing anomalies. SIAM journal on Applied Mathematics **17**(2), 416–429 (1969)

Le Sueur, E., Heiser, G.: Dynamic voltage and frequency scaling: The laws of diminishing returns. In: Proceedings of the 2010 International Conference on Power Aware Computing and Systems, pp. 1–8 (2010)

Melani, A., Bertogna, M., Bonifaci, V., Marchetti-Spaccamela, A., Buttazzo, G.C.: Response-time analysis of conditional DAG tasks in multiprocessor systems. In: 2015 27th Euromicro Conference on Real-Time Systems, pp. 211–221 (2015). https://doi.org/10.1109/ECRTS.2015.26

Mascitti, A., Cucinotta, T.: Dynamic partitioned scheduling of real-time DAG tasks on arm big.LITTLE architectures*. In: Proceedings of the 29th International Conference on Real-Time Networks and Systems. RTNS '21, pp. 1–11. Association for Computing Machinery, New York, NY, USA (2021). https://doi.org/10.1145/3453417.3453442 . https://doi.org/10.1145/3453417.3453442

Mascitti, A., Cucinotta, T., Marinoni, M., Abeni, L.: Dynamic partitioned scheduling of real-time tasks on arm big.LITTLE architectures. Journal of Systems and Software **173**, 110886 (2021) https://doi.org/10.1016/j.jss.2020.110886

Oza, N.C., Russell, S.J.: Online bagging and boosting. In: International Workshop on Artificial Intelligence and Statistics, pp. 229–236 (2001). PMLR

Papadopoulos, A.V., Agrawal, K., Bini, E., Baruah, S.: Feedback-based resource management for multi-threaded applications. Real-Time Systems, 1–34 (2022)

Pillai, P., Shin, K.G.: Real-time dynamic voltage scaling for low-power embedded operating systems. In: Proceedings of the Eighteenth ACM Symposium on Operating Systems Principles, pp. 89–102 (2001)

Raghunathan, V., Borkar, V., Cao, M., Kumar, P.R.: Index policies for real-time multicast scheduling for wireless broadcast systems. In: IEEE INFOCOM 2008-The 27th Conference on Computer Communications, pp. 1570–1578 (2008). IEEE

Rubin, D.B.: The Bayesian bootstrap. The annals of statistics, 130–134 (1981)

Russo, D.J., Van Roy, B., Kazerouni, A., Osband, I., Wen, Z., *et al.*: A tutorial on Thompson sampling. Foundations and Trends® in Machine Learning **11**(1), 1–96 (2018)

Sharafzadeh, E., Kohroudi, S.A.S., Asyabi, E., Sharifi, M.: Yawn: A CPU idle-state governor for datacenter applications. In: Proceedings of the 10th ACM SIGOPS Asia-Pacific Workshop on Systems, pp. 91–98 (2019)

Saifullah, A., Li, J., Agrawal, K., Lu, C., Gill, C.: Multi-core real-time scheduling for generalized parallel task models. Real-Time Systems **49**, 404–435 (2013)

Slivkins, A.: Introduction to multi-armed bandits. Foundations and Trends® in Machine Learning **12**(1-2), 1–286 (2019) https://doi.org/10.1561/2200000068

Schöne, R., Molka, D., Werner, M.: Wake-up latencies for processor idle states on current x86 processors. Computer Science-Research and Development **30**, 219–227 (2015)

Sheikh, S.Z., Pasha, M.A.: Energy-efficient multicore scheduling for hard real-time systems: A survey. ACM Transactions on Embedded Computing Systems (TECS) **17**(6), 1–26 (2018)

Vasilakis, E.: An instruction level energy characterization of ARM processors. Foundation of Research and Technology Hellas, Inst. of Computer Science, Tech. Rep. FORTH-ICS/TR-450 (2015)

Whittle, P.: Restless bandits: Activity allocation in a changing world. Journal of applied probability **25**(A), 287–298 (1988)

Xie, G., Xiao, X., Peng, H., Li, R., Li, K.: A survey of low-energy parallel scheduling algorithms. IEEE Transactions on Sustainable Computing **7**(1), 27–46 (2021)

Yu, Z., Xu, Y., Tong, L.: Deadline scheduling as restless bandits. IEEE Transactions on Automatic Control **63**(8), 2343–2358 (2018)

Zhan, X., Azimi, R., Kanev, S., Brooks, D., Reda, S.: CARB: A C-state power management arbiter for latency-critical workloads. IEEE Computer Architecture Letters **16**(1), 6–9 (2016)