Adapting Classic Scheduling Heuristics for Online Execution under Uncertainty

Jason Chamorro
Loyola Marymount University
Los Angeles, USA
jchamor1@lion.lmu.edu

Gabriel Twigg-Ho
Swinburne University of Technology
Melbourne, Australia
103597673@student.swin.edu.au

Jared Coleman
Loyola Marymount University
Los Angeles, USA
jared.coleman@lmu.edu

Tainã Coleman
San Diego Supercomputer Center (SDSC)
San Diego, USA
t1coleman@sdsc.edu

Bhaskar Krishnamachari
University of Southern California (USC)
Los Angeles, USA
bkrishna@usc.edu

Mohammadali Khodabandehlou
University of Southern California (USC)
Los Angeles, USA
mk40089@usc.edu

Abstract

We present an automated framework for online task scheduling on heterogeneous distributed systems, building on a modular parametric scheduler that enables dynamic scheduling decisions based on evolving execution states. Inspired by classical list-scheduling strategies such as HEFT and CPoP, our online scheduler simulates real-time task scheduling using only partial task graph knowledge. We evaluate our online scheduler variants against both their traditional offline baselines and a naive online strategy using a large-scale benchmark suite of real-world scientific workflows. Experimental results across different estimation methods and compute-to-communication ratio (CCR) settings show that our adaptive online schedulers consistently outperform the naive approach, achieving performance within approximately 3–5% of an ideal offline scheduler that has full future knowledge (compared to the approximately 10% overhead for the naive baseline).

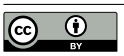
CCS Concepts

• **Computing methodologies** → **Distributed computing methodologies**.

Keywords

Scheduling, Online, Heterogeneous Networks, Scientific Workflows

ACM Reference Format:
Jason Chamorro, Gabriel Twigg-Ho, Jared Coleman, Tainã Coleman, Bhaskar Krishnamachari, and Mohammadali Khodabandehlou. 2025. Adapting Classic Scheduling Heuristics for Online Execution under Uncertainty. In *Workshops of the International Conference for High Performance Computing, Networking, Storage and Analysis (SC Workshops '25), November 16–21, 2025, St Louis, MO, USA.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3731599.3767581



SC Workshops '25, St Louis, MO, USA

ACM ISBN 979-8-4007-1871-7/25/11
https://doi.org/10.1145/3731599.3767581

1 Introduction

Scheduling tasks with precedence constraints onto heterogeneous distributed computing resources is a well-studied but computationally hard problem. In practice, heuristics like HEFT and CPoP offer strong offline performance assuming perfect knowledge of all task execution times [15]. However, these assumptions are often invalid in real-world settings such as edge computing, sensor-driven pipelines, or scientific workflows where task costs are uncertain and only revealed at runtime.

To address this, we propose an automated framework for online scheduling that dynamically adapts existing list-scheduling algorithms (like HEFT and CPoP) to handle runtime uncertainty. Our design extends a previously proposed parametric list-scheduling algorithm [3] to support online scheduling. This enables integration of classic scheduling heuristics into an online context with minimal code duplication.

Our contributions are as follows:

- We introduce a novel online scheduling framework that extends a parametric list-scheduling engine, enabling automated adaptation of classical offline algorithms (e.g., HEFT, CPoP) to online settings. This produces a large family of online scheduling variants with minimal future implementation effort.
- We propose a benchmarking methodology for online scheduling that evaluates performance on real-world scientific workflows using stochastic task cost models and varying computation-to-communication ratios (CCRs), with trace-driven task graphs from WfCommons [5].
- We provide empirical evidence that online adaptation significantly improves performance compared to naive baselines, often improving performance by over 15% and achieving results much closer to the idealized offline schedulers, demonstrating that online feedback and reactivity can make a substantial difference in realistic workflow execution scenarios.

The rest of the paper is organized as follows. Section 2 formally defines the problem and notation used throughout the paper. Section 3 reviews relevant literature on task graph scheduling, online

and stochastic scheduling, and scientific workflow systems. Section 4 describes our parametric scheduling framework and its extension to online execution, including our feedback-driven simulation model and baseline strategies. Section 5 presents the experimental setup, including workflow datasets, estimation strategies, and evaluation metrics. Section 6 reports the results of our empirical evaluation, comparing online and naive schedulers across estimation methods and heuristics. Finally, Section 7 concludes with a summary of contributions and directions for future work.

2 Problem Definition & Notation

We study the problem of scheduling tasks with precedence constraints onto a heterogeneous distributed system under stochastic task graph and network configurations, with the goal of minimizing expected makespan. The runtime of the scheduling algorithm is considered negligible and is not included in the makespan calculation.

Let the task graph be denoted by $G = (T, D)$, where T is the set of tasks and $D \subseteq T \times T$ is the set of directed edges representing precedence constraints: if $(t, t') \in D$, then t' cannot start until t has completed execution and communicated its output to the node on which task t' will execute on. Each task $t \in T$ has a stochastic cost $c(t) \sim \mathcal{D}_t$. Each dependency $(t, t') \in D$ has a stochastic data size $d(t, t') \sim \mathcal{D}_{t,t'}$ representing the amount of data to be transferred from t to t'.

The compute network is represented by a graph $N = (V, E)$, where V is the set of compute nodes and $E \subseteq V \times V$ is the set of communication links. Each node $v \in V$ has a stochastic processing speed $s(v) \sim \mathcal{S}_v$, and each communication link $(v, v') \in E$ has a stochastic bandwidth $s(v, v') \sim \mathcal{S}_{v,v'}$.

Let $\theta \sim \mathcal{D} \times \mathcal{S}$ denote a realization of all stochastic quantities, where $\mathcal{D}$ represents the distributions over task and communication costs, and $\mathcal{S}$ represents the distributions over processing speeds and bandwidths. Under a fixed realization θ, we use the notation $c_\theta(t)$, $d_\theta(t, t')$, $s_\theta(v)$, and $s_\theta(v, v')$ to refer to the realized (sampled) values of task cost, data size, processing speed, and bandwidth, respectively.

Schedules

A *schedule* S is a set of tuples of the form (t, v, r), where $t \in T$ is a task, $v \in V$ is the compute node assigned to t, and $r \in \mathbb{R}^+$ is the task's start time. Schedule feasibility is defined with respect to a particular realization $\theta \sim \mathcal{D} \times \mathcal{S}$. That is, a schedule S is considered *valid under a realization* θ if it satisfies the following constraints:

- **Uniqueness:** Each task is scheduled exactly once:

$$\forall t \in T,\ \exists!\ (t, v, r) \in S.$$

- **Precedence and communication:** For every dependency $(t, t') \in D$, the successor task must start only after the predecessor has completed execution and its output has been transferred:

$$\text{For all } (t, v, r),\ (t', v', r') \in S,\ (t, t') \in D: \quad r + \frac{c_\theta(t)}{s_\theta(v)} + \frac{d_\theta(t, t')}{s_\theta(v, v')} \leq r'.$$

- **No overlap:** Tasks assigned to the same node must not overlap in time:

$$\text{For all } (t_1, v, r_1),\ (t_2, v, r_2) \in S,\ t_1 \neq t_2: \quad \text{either } r_1 + \frac{c_\theta(t_1)}{s_\theta(v)} \leq r_2 \quad \text{or } r_2 + \frac{c_\theta(t_2)}{s_\theta(v)} \leq r_1.$$

Given a realization θ, the *makespan* of a schedule S is defined as the time at which the final task completes under the realized execution and communication times:

$$m_\theta(S) = \max_{(t,v,r) \in S} \left(r + \frac{c_\theta(t)}{s_\theta(v)} \right).$$

In general, our goal is to design scheduling algorithms that minimize the expected makespan:

$$\min_S\ \mathbb{E}_{\theta \sim \mathcal{D} \times \mathcal{S}} \left[m_\theta(S) \right].$$

Scheduling Under Uncertainty

Because computing the optimal expected schedule is intractable, heuristic strategies are used in practice. Two dominant strategies adapt classical deterministic heuristics to the stochastic setting:

(1) **Determinization-based scheduling:** Replace each random variable with a fixed point estimate (e.g., mean or conservative quantile), then apply a deterministic scheduling algorithm (such as HEFT or CPoP). The resulting schedule is planned and committed in advance.
(2) **Online scheduling:** Postpone scheduling decisions until a task is *ready* (i.e., when all its dependencies have completed). Then, a heuristic is used based on the current state and estimated durations. This allows adaptation to observed execution behavior.

Our approach combines both strategies: we use estimates to produce a tentative schedule, but re-schedule as tasks become ready and actual durations are revealed.

To formalize this, we distinguish between *estimated schedules* and *realized schedules*. Estimated schedules are constructed using point estimates of the stochastic quantities. Specifically, we let $\hat{c}(t)$, $\hat{d}(t, t')$, $\hat{s}(v)$, and $\hat{s}(v, v')$ denote estimated values for task costs, data sizes, processing speeds, and link bandwidths, respectively. These estimates typically reflect expected values, quantiles, or historical observations.

An *estimated schedule* $\hat{S}$ is defined to be a valid schedule under the assumption that these estimates are correct, that is, under a hypothetical realization $\hat{\theta}$ such that $c_{\hat{\theta}}(t) = \hat{c}(t)$ and similarly for all other parameters. Formally, $\hat{S} = \{(t, v_t, \hat{r}(t))\}$ assigns each task $t \in T$ to a node $v_t \in V$ and estimated start time $\hat{r}(t)$, such that:

$$\hat{r}(t') \geq \hat{r}(t) + \frac{\hat{c}(t)}{\hat{s}(v_t)} + \frac{\hat{d}(t, t')}{\hat{s}(v_t, v_{t'})} \quad \forall\, (t, t') \in D,$$

$$\hat{r}(t_{i+1}) \geq \hat{r}(t_i) + \frac{\hat{c}(t_i)}{\hat{s}(v_{t_i})} \quad \forall \text{ consecutive tasks } t_i, t_{i+1} \text{ on the same node.}$$

While estimated schedules are useful for planning, actual execution may deviate due to randomness in task costs and system behavior. To capture this, we define the corresponding *realized schedule* S_θ for a specific sampled realization $\theta \sim \mathcal{D} \times \mathcal{S}$. Each task

t is executed on the same node v_t as in $\hat{S}$, but its realized start time $r_\theta(t)$ is determined using the true realized values $c_\theta(t)$, $d_\theta(t, t')$, $s_\theta(v)$, and $s_\theta(v, v')$. The start time of each task can be computed recursively as:

$$r_\theta(t) = \max \left\{ \max_{(t',t)\in D} \left[r_\theta(t') + \frac{d_\theta(t', t)}{s_\theta(v_{t'}, v_t)} \right],\ \text{prev}_\theta(v_t, t) \right\}$$

Here, $\text{prev}_\theta(v_t, t)$ denotes the end time of the most recent task scheduled on node v_t before t, or 0 if t is the first task scheduled on that node. This mapping from estimated to realized schedule enables us to evaluate a scheduling strategy's performance under uncertainty. In our experiments, we generate estimated schedules using heuristic strategies, then simulate their execution by sampling multiple realizations θ and computing the resulting realized makespans $m_\theta(S)$. This lets us assess both average-case performance and variance across scenarios.

Figure 1 shows a simple example used to illustrate the difference between estimated and realized schedules. The left panel depicts the compute network, consisting of two identical nodes connected by symmetric links. The middle and right panels show the task graph under estimated and actual durations, respectively. Each task is assigned both an estimated duration (used for planning) and an actual duration (used during execution). Edges denote precedence constraints.

Figure 2 shows the corresponding schedule under the estimated durations (top) and how the same schedule plays out when actual durations are realized (bottom). The estimated schedule is feasible under the point estimates but can result in suboptimal or invalid timing when the true execution times differ. This example highlights a key challenge in stochastic scheduling: while point estimates enable scheduling using well-known algorithms, they may produce execution plans that perform poorly in practice if variance is not accounted for. Realized schedules provide a way to assess the true performance of a scheduling policy under sampled conditions.

3 Related Work

Task scheduling for heterogeneous distributed systems has long been a central research problem. Classical strategies such as MCP [8], HEFT, and CPoP [15] produce efficient static schedules for task graphs when execution times are known in advance. These list-based heuristics prioritize tasks based on metrics like upward rank or critical path length and assign them to resources to minimize the earliest finish time. While widely used due to their simplicity and generality, these algorithms rely on full knowledge of the task graph and assume deterministic task graph costs and network speeds. These assumptions rarely hold, however, in real-world systems like scientific workflow or edge computing environments.

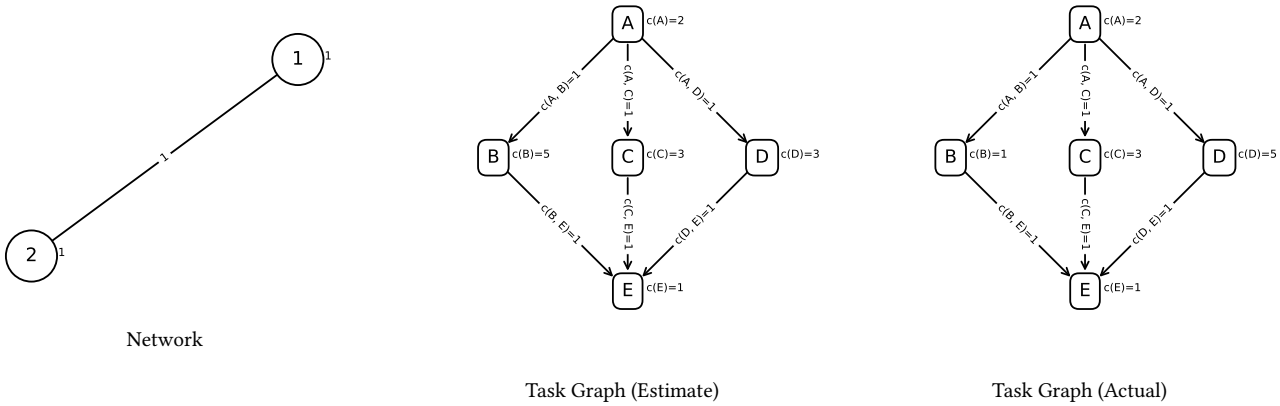
To account for partial observability and dynamic task readiness, many online scheduling algorithms have been proposed. These include extensions of list-based heuristics that either defer task assignment until runtime or adapt the initial schedule for optimization, such as aHEFT [21] and DRHEFT [22], and more complex ones such as DLS [17] and the adaptive resource allocation and scheduling mechanisms in the Pegasus workflow management system [9]. These strategies operate by either creating an initial schedule that can be adapted during execution according to the resource availability, task success rate, etc, or by scheduling tasks as they become ready during execution, typically using static priority functions and estimated task costs to guide decisions. A different class of methods, which we refer to as *naive online* schedulers, attempts to account for runtime variability by incorporating stochastic duration estimates into a fully planned schedule before execution begins. For instance, SHEFT [13] modifies HEFT to use both the mean and variance of task costs. Other approaches attempt to optimize expected makespan or minimize deadline violation probability [14, 19]. Although these schedulers acknowledge uncertainty in task costs by reasoning about distributions, they commit to an entire execution plan up front, without any mechanism for revising the schedule in response to runtime observations.

Other approaches have focused on developing more sophisticated estimators to handle runtime uncertainty. For instance, [2] proposes a method based on Cumulative Distribution Functions (CDFs) to generate a probabilistic makespan estimate for an entire workflow. While this method provides a robust upfront plan it is designed for static scheduling. In contrast our work focuses on the online scheduling framework itself which is less dependent on the perfection of a single initial estimate and relies instead on reactivity during runtime. The challenge of scheduling with imperfect information is also well documented. A study by [7] showed that in a high throughput context with multiple competing workflows, macro-level decisions (e.g. which workflow to prioritize) can have a greater influence on performance than the intricate micro-level decisions of scheduling tasks within a single workflow. Since our work focuses on optimizing the runtime and execution of individual workflows we address this foundational "micro-level" problem.

While our work focuses on scheduling to minimize makespan, a parallel line of research has looked into predicting resource requirements. For example, [1] describes a method for dynamically predicting a task's memory consumption to reduce wastage. Similarly, [18], [16] propose feedback loop systems for predicting the needed resources like CPU cores and memory. This research is complementary to our own as while these methods aim to determine how much of a resource a task needs our framework focuses on optimally scheduling tasks on those resources to determine when and where they should run.

Scientific workflows introduce additional challenges for scheduling. These workflows often comprise large, complex DAGs representing applications in bioinformatics, astrophysics, and other data-intensive domains. Numerous scheduling strategies have been developed for workflows running on cloud infrastructure [6, 10, 11], many of which focus on optimizing cost, energy, or multi-objective trade-offs under cloud-specific constraints. However, these methods tend to assume static input and often prioritize infrastructure-level concerns (e.g., VM pricing). In high-performance computing (HPC) contexts, workflows are typically executed through batch systems like Slurm [20]. Systems such as WoAS [12] introduce workflow-aware scheduling layers that improve resource allocation by grouping and reordering tasks.

In evaluating scheduling strategies, recent work has emphasized the need for standardized, extensible benchmarking. The SAGA framework [3] provides an open-source library for systematically comparing scheduling algorithms across diverse workloads and



Network

Task Graph (Estimate)

Task Graph (Actual)

Figure 1: Network and task graph structure. Left: symmetric two-node network (estimated and actual). Middle: estimated task costs. Right: actual task costs.

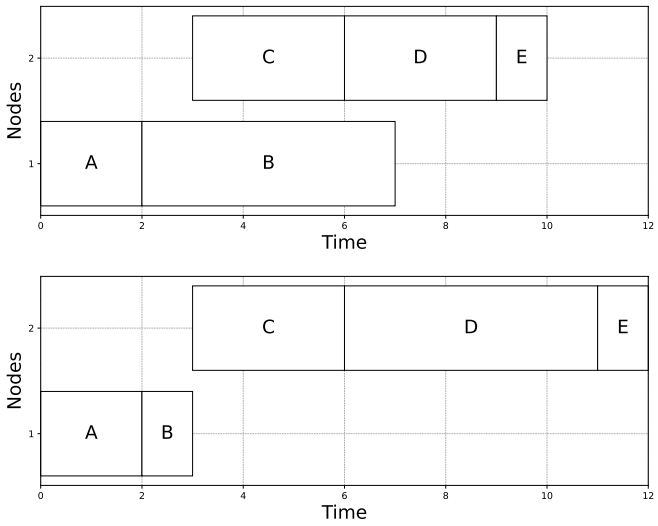


Figure 2: Estimated schedule (top) and realized schedule under actual durations (bottom). Even small misestimates can lead to overlapping tasks or idle gaps.

datasets. It enables large-scale, reproducible evaluation of algorithmic performance using a shared interface, which we extend in this work to support adaptive online scheduling. Complementary to this, the WfCommons suite [5] offers realistic workflow traces with empirical execution-time distributions from applications such as BLAST, BWA, and Montage. It also includes WfChef [4], a generator that produces synthetic workflows with structure and statistical properties drawn from real-world traces. Together, these tools support more comprehensive, reproducible evaluation of scheduling strategies across diverse and realistic workloads.

4 Methodology

In this section, we describe our online scheduling framework, which extends a modular parametric scheduler to support dynamic, feedback-driven execution under runtime uncertainty.

4.1 Parametric Scheduling Framework

Our system builds on a parametric scheduler introduced in prior work [3] that decomposes list-scheduling heuristics into four modular components:

- **Priority Function**: Determines the global order in which tasks are considered (e.g., upward rank for HEFT or critical path-based ranking for CPoP).
- **Comparison Function**: Chooses the best node for a given task based on some cost metric (e.g., earliest finish time, start time, or execution speed).
- **Insertion Strategy**: Dictates whether tasks are appended at the end of a schedule (append-only) or inserted into gaps (insertion-based).
- **Critical Path Reservation**: If enabled, reserves the fastest node for critical-path tasks, a strategy inspired by CPoP.

In prior work, we showed that these four components are sufficient to recompose many classical list-scheduling algorithms. For example, by selecting an upward rank priority function, earliest finish time comparison, the insertion-based strategy, no critical path reservation, the resulting algorithm is equivalent to HEFT. If instead we use the CPoP ranking function and enable critical-path reservation, then the resulting algorithm is equivalent to CPoP. This modular structure allows for novel combinations of these components, yielding 36 unique scheduling algorithms, many of which had not been previously explored.

This flexible design makes it easy to explore a rich space of scheduling strategies while maintaining a shared codebase. We extended the same parametric approach with an online adapter, enabling all 36 algorithms to operate in an incremental, feedback-driven setting that reflects real-world runtime uncertainty. In this

paper, we focus our analysis on two of the most well-known of these algorithms: HEFT and CPoP.

4.2 Online Execution Model

To extend our parametric schedulers into an online setting, we wrap them in a feedback-driven loop that simulates runtime execution using stochastic task cost estimates. At each iteration, the scheduler produces a complete estimated schedule for all remaining tasks, then commits only the subset that starts before the next observable event: a task completion. Once a task completes, the scheduler updates the execution state and re-estimates the remaining schedule, potentially re-scheduling tasks that have not started yet. This process is formalized in Algorithm 1.

Algorithm 1 Online Parametric Scheduling Loop

Initialize schedule $\mathcal{S}_{\text{actual}} \leftarrow \emptyset$
Initialize set of completed tasks $F \leftarrow \emptyset$
Initialize current time $t \leftarrow 0$
while not all tasks are completed **do**
  Generate full estimated schedule $\hat{S}$ using parametric scheduler with estimated durations and S_{actual}
  Map $\hat{S}$ to actual schedule $\mathcal{S}_{\text{hyp}}$ using sampled durations
  Let $\mathcal{T}_{\text{unfinished}} \leftarrow$ all tasks in $\mathcal{S}_{\text{hyp}}$ not in F
  Let $T_{\text{next}} \leftarrow \arg\min_{T \in \mathcal{T}_{\text{unfinished}}} \text{end}(T)$
  Advance time: $t \leftarrow \text{end}(T_{\text{next}})$
  Add T_{next} to F
  Update $\mathcal{S}_{\text{actual}}$ with all tasks in $\mathcal{S}_{\text{hyp}}$ with start time $\leq t$
end while
return Final schedule $\mathcal{S}_{\text{actual}}$

This procedure simulates an online environment in which task costs are not known a priori but instead sampled at runtime from distributions. Importantly, we do not make hard scheduling commitments in advance; instead, we replan the entire schedule after each task finishes, while preserving prior decisions for tasks that have already started. This approach makes minimal assumptions about uncertainty models and allows for the direct reuse of offline heuristics like HEFT and CPoP by simply replacing their static duration inputs with stochastic estimates.

The algorithm proceeds as follows. We begin with an empty actual schedule and zero tasks completed. At each iteration of the loop:

(1) **Estimate the Remaining Schedule:** We invoke a parametric offline scheduler (e.g., HEFT or CPoP) using the current state of the system and estimated weights. This scheduler produces a full schedule ($\hat{S}$) for *all* tasks, including those that have not yet started.
(2) **Get Schedule with Actual Durations:** Since we are simulating an online setting, we do not use these estimates directly. Instead, we convert the estimated schedule into a "hypothetical actual" schedule by replacing estimated durations with their sampled true durations (i.e., following the method described in Section 2). This allows us to determine which task *actually* finishes next.
(3) **Advance Time to Next Completion:** Among all tasks in the hypothetical schedule that have not yet been completed, we find the one that is scheduled to finish the soonest and move the simulation clock forward to that task's end time. This simulates the passage of time and marks the next observable event in the system.
(4) **Lock In Committed Tasks:** We commit all tasks in the hypothetical schedule whose start times are less than or equal to the current time. These tasks are considered to have started execution and can no longer be rescheduled (i.e., they remain fixed in all future iterations' estimated schedules $\hat{S}$).
(5) **Repeat:** The process repeats until all tasks have completed. Each new schedule builds upon the current execution state, incorporates updated feedback, and respects the actual durations of previously started tasks.

This feedback-driven approach allows our framework to retain the structure and logic of offline heuristics, while adapting them dynamically to runtime conditions. Critically, the algorithm never revisits or reschedules tasks that have already started execution. This ensures that the simulation is faithful to many real-world constraints, where tasks, once launched, are committed to a specific node and cannot be interrupted or migrated.

4.3 Naive Online Baseline

To evaluate the benefit of our proposed online re-scheduling strategy, we define a naive baseline that serves as a natural point of comparison. This baseline uses a parametric scheduler (e.g., HEFT or CPoP) with stochastic duration estimates to generate a complete schedule, but commits *all tasks at once*, without revisiting or adapting the schedule as execution progresses.

Conceptually, this simulates an overconfident planner: it acknowledges uncertainty at the outset (by using stochastic estimates) but fails to incorporate any feedback from actual task completions. Once the schedule is generated, it is treated as fixed, regardless of how reality unfolds. This approach mirrors what is done in many practical settings and prior research. For instance, the SHEFT algorithm [13] computes a modified version of HEFT using the mean and variance of task costs but still produces a fully committed schedule based on those estimates. Similarly, in most "deterministic" scheduling literature, task runtimes are assumed to be known in advance even though, in practice, they are usually just derived from profiling data or coarse-grained models.

Thus, our naive baseline can be seen as an abstraction of these widespread strategies: one-shot, estimate-driven planning that does not account for dynamic feedback. By contrasting it with our adaptive online method, we can quantify the performance gains enabled by runtime reactivity.

5 Experimental Setup

We evaluate our scheduling strategies using workflow DAGs from the WfCommons suite [5], which provides real execution traces from nine scientific applications, including BLAST, BWA, Epigenomics, Montage, and others. For each task and edge type, the observed samples are fitted to a family of common probability distributions (e.g., normal, log-normal, exponential, gamma, etc.), and the best-fitting distribution is selected based on statistical criteria. These fitted distributions serve as stochastic models from which

we draw independent and identically distributed (i.i.d.) samples for task costs and data sizes during simulation.

For compute node speeds, we do not fit parametric distributions. Instead, we use empirical distributions derived from real execution traces collected in prior studies. At runtime, node speeds are sampled i.i.d. from these empirical datasets. This approach preserves observed heterogeneity and variance in compute resources. Network bandwidths, on the other hand, are assumed to be homogeneous across all links in the system. Rather than sampling from a distribution, we scale the effective bandwidths to control the computation-to-communication ratio (CCR), a standard parameter used to simulate compute-bound or communication-bound environments (more detail on this below). WfCommons also includes WfChef [4], a synthetic workflow generator which we use to produce new, in-distribution workflow instances that preserve the structural and statistical properties of the original applications.

Estimators. Our scheduling algorithms require point estimates of task costs as inputs. We evaluate two such estimators:

- **Mean**: Uses the empirical mean $\mathbb{E}[X]$ of each task's cost or network speed distribution.
- **SHEFT-inspired**: Based on the formulation proposed in the SHEFT algorithm [13], this estimator adjusts task costs upward and network speeds downward based on their variances. Given a random variable $X \sim \mathcal{D}$, the re-weighted estimate $\hat{X}$ is defined as:
  - For task graph weights (e.g., costs or data size), we adjust *upward* to reflect more conservative assumptions:

$$\hat{X} = \begin{cases} \mathbb{E}[X] + \sqrt{\mathrm{Var}(X)} & \text{if } \frac{\mathrm{Var}(X)}{\mathbb{E}[X]} \leq 1 \\ \mathbb{E}[X] + \frac{\mathbb{E}[X]}{\sqrt{\mathrm{Var}(X)}} & \text{otherwise} \end{cases}$$

  - For network speeds (e.g., processing speed $s(v)$ or bandwidth $s(v, v')$), we adjust *downward* to reflect more conservative assumptions:

$$\hat{X} = \begin{cases} \mathbb{E}[X] - \sqrt{\mathrm{Var}(X)} & \text{if } \frac{\mathrm{Var}(X)}{\mathbb{E}[X]} \leq 1 \\ \mathbb{E}[X] - \frac{\mathbb{E}[X]}{\sqrt{\mathrm{Var}(X)}} & \text{otherwise} \end{cases}$$

  This reflects the intuition that high-variance tasks should be treated more cautiously by overestimating their cost, while potentially unreliable network speeds should be conservatively underestimated to avoid overly optimistic scheduling decisions. Note that SHEFT, as originally proposed, focused solely on task costs. Our extension generalizes its logic to handle both computational and communication uncertainty.

Schedulers. We benchmark two base heuristics, HEFT and CPoP, under three different execution modes:

(1) **Offline**: Assumes full knowledge of actual task and network weights ahead of time. This serves as a reasonable bound on achievable performance.
(2) **Online**: Our proposed re-scheduling strategy, which repeatedly generates full schedules using the current estimator and adapts as tasks complete and actual durations are revealed.
(3) **Naive Online**: A single-pass scheduler that uses an estimator to generate a fixed schedule up front and commits to it entirely. This simulates a common practical approach that accounts for uncertainty during planning but does not incorporate runtime feedback.

Configurations. For each workflow, we explore a range of system and modeling parameters:

- **Computation-to-communication Ratios (CCR):** We vary CCR values from 0.2 to 5.0 to simulate compute-bound and communication-bound workflows. The CCR is defined as:

$$\mathrm{CCR} = \frac{\text{mean task cost}}{\text{mean node speed}} \Big/ \frac{\text{mean data size}}{\text{mean link speed}}$$

  This represents the ratio of average computation time to average communication time per task. A low CCR (e.g., 0.2) corresponds to communication-heavy scenarios where data movement dominates execution time, while a high CCR (e.g., 5.0) indicates compute-heavy settings.
- **Monte Carlo Sampling:** Each configuration (workflow, CCR, estimation method, scheduler) is simulated over 100 independent Monte Carlo samples. In each run, actual task costs are drawn from the empirical runtime distributions provided by WfCommons.

To evaluate performance across different schedulers and estimators, we use the **makespan** as our primary metric. Since absolute makespan values depend heavily on the specific workflow and system parameters, we report *normalized* results using the *makespan ratio*:

$$\mathrm{MR}_{\mathrm{Online}} = \frac{m_{\mathrm{Online}}}{m_{\mathrm{Offline}}}, \qquad \mathrm{MR}_{\mathrm{Naive}} = \frac{m_{\mathrm{Naive}}}{m_{\mathrm{Offline}}}$$

where m_{Online} and m_{Naive} are the makespans achieved by the Online and Naive Online schedulers, respectively, and m_{Offline} is the makespan of the corresponding Offline scheduler that has full knowledge of actual task costs and compute node speeds. A ratio closer to 1 indicates performance closer to that of the idealized offline scheduler.

In many prior studies, makespan ratios are defined relative to the *best* makespan achieved by any of the evaluated algorithms, treating it as a proxy for the optimal schedule (since the true optimal is generally intractable to compute). However, we do not take that approach here. Instead, we normalize all results against the same heuristic applied in an oracle-like offline setting (i.e., with full knowledge of actual task and network costs). This allows us to isolate the impact of dynamic re-scheduling alone, without conflating it with differences in the base heuristic itself. In other words, our goal is not to compare across scheduling families, but to understand how online reactivity improves performance relative to the same algorithm with perfect foresight.

6 Results

Table 1 summarizes the normalized makespan ratios for both our online scheduler and the naive online baseline across all combinations of estimation method and base scheduling heuristic (HEFT or CPoP). Values are averaged over 100 Monte Carlo runs per configuration, with standard deviation shown to reflect runtime variability.

Figure 3 and Figure 4 show the distribution of makespan ratios across varying computation-to-communication ratios (CCR), broken down by heuristic (HEFT or CPoP) and estimation strategy (mean or SHEFT). Several consistent patterns emerge:

Table 1: Makespan Ratio statistics.

Estimator	Scheduler	MR_{Online}	MR_{Naive}
SHEFT	CPoP	1.056 ± 0.116	1.105 ± 0.140
SHEFT	HEFT	1.045 ± 0.116	1.108 ± 0.140
mean	CPoP	1.046 ± 0.102	1.102 ± 0.138
mean	HEFT	1.036 ± 0.100	1.108 ± 0.143

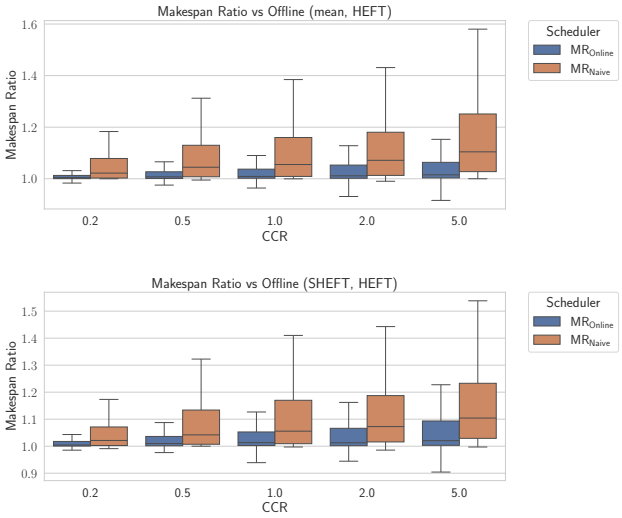


Figure 3: Normalized makespan ratios using HEFT across CCR values. Top: mean-based estimates; Bottom: SHEFT-style variance-aware estimates. Lower is better.

- **Online scheduling consistently outperforms naive scheduling.** Across all combinations, the online scheduler achieves lower average makespan ratios than the naive online baseline, confirming the benefit of progressive re-scheduling. The relative improvement is typically in the range of 15-20%.
- **SHEFT-based estimates slightly underperform compared to mean-based estimates**, both for the Online and Naive Online schedulers. This is somewhat surprising, given that SHEFT was designed as a more conservative estimator to account for task runtime variance. This result suggests that conservatism in estimation does not always yield better outcomes and that, for certain applications, simpler mean-based estimators may provide better overall performance.
- **Observed variance is substantially reduced in the online setting for most cases.** While the standard deviations in Table 1 appear only slightly better for the online schedulers, the box plots in Figures 3 and 4 reveal a much clearer picture: the interquartile ranges are significantly tighter for the online scheduler. This suggests that the online strategy not only improves average makespan, but also leads to more stable and predictable performance in the majority of cases. The apparent similarity in standard deviations is likely due to outliers in the results, which skew the mean-based variability metric.

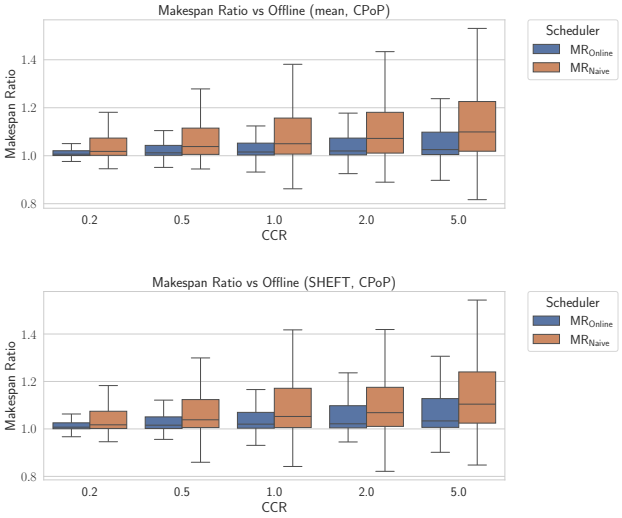


Figure 4: Normalized makespan ratios using CPoP across CCR values. Top: mean-based estimates; Bottom: SHEFT-style variance-aware estimates. Lower is better.

These results show that even a lightweight online extension of classic heuristics like HEFT and CPoP can yield substantial performance and robustness benefits in uncertain environments.

6.1 Workflow-Specific Results

While our aggregate results demonstrate that the online scheduler consistently outperforms the naive baseline in terms of both expected makespan and variance, a more detailed breakdown reveals that performance is workflow-dependent. Figures 5 through 8 present normalized makespan ratios grouped by individual workflows. Online scheduling generally improves performance across most workflows. For the majority of applications, the online scheduler yields lower median makespan ratios and narrower interquartile ranges compared to the naive baseline. This confirms that dynamic re-scheduling can significantly mitigate the impact of runtime uncertainty in a variety of scientific workloads.

Montage appears to be an exception, though. Interestingly, this is one of the few workflows where the online scheduler performs *worse* than the naive variant across most estimation methods and base heuristics. We hypothesize that this may be due to the very high variance of some task costs in Montage compared to other workflows. In particular, the real execution trace data shows that mBgModel tasks in this workflow have a runtime of anywhere from 11 seconds to 24 days (with an average runtime of 1.3 hours)!

These results highlight that while online reactivity generally leads to better performance, its benefits are not uniform across all application types.

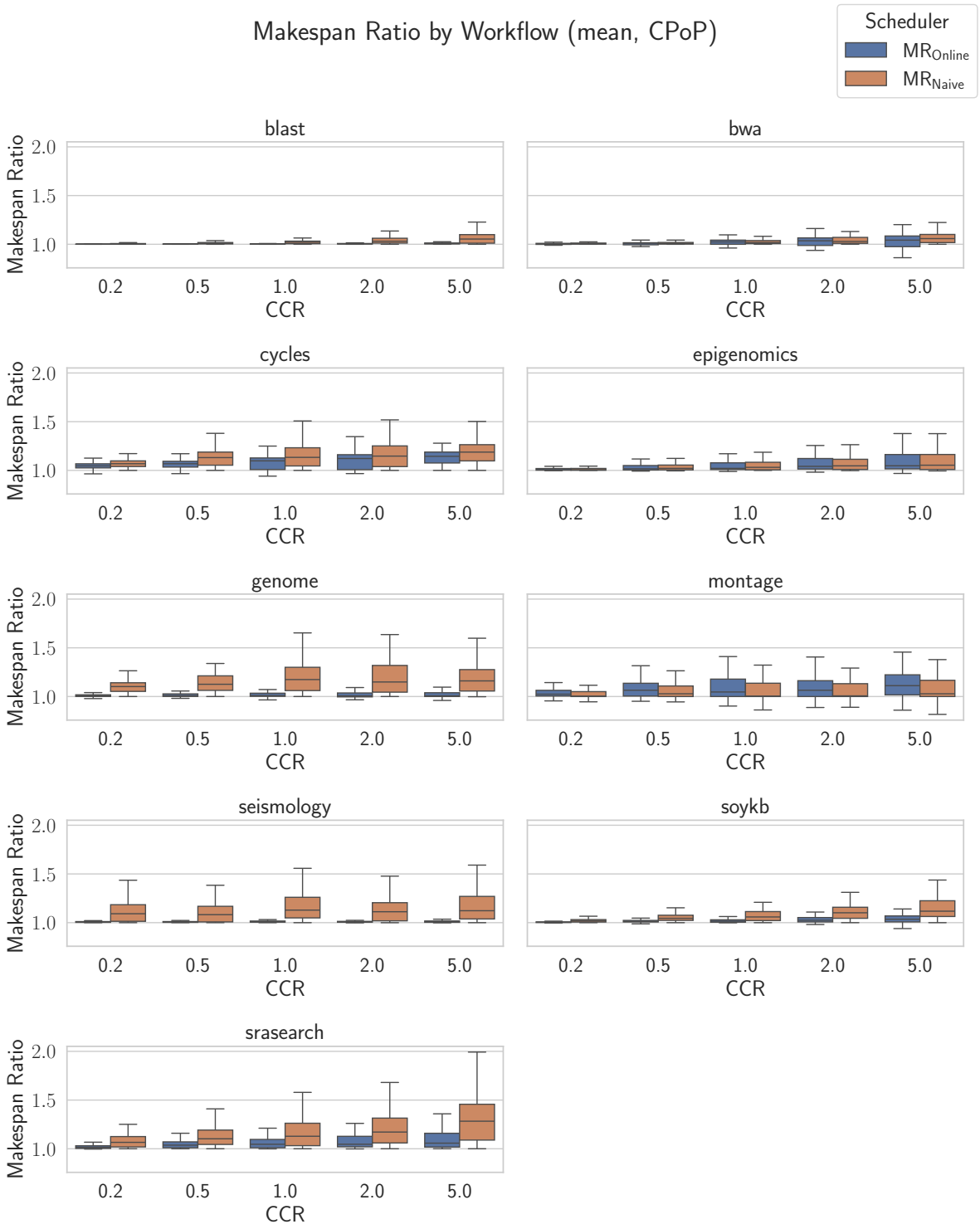


Figure 5: Normalized makespan ratios by workflow using the mean estimator with the CPoP scheduling heuristic. Online scheduling (blue) often improves both median performance and spread relative to the naive baseline (orange), although results vary by workflow. Notably, Montage performs worse with online re-scheduling.

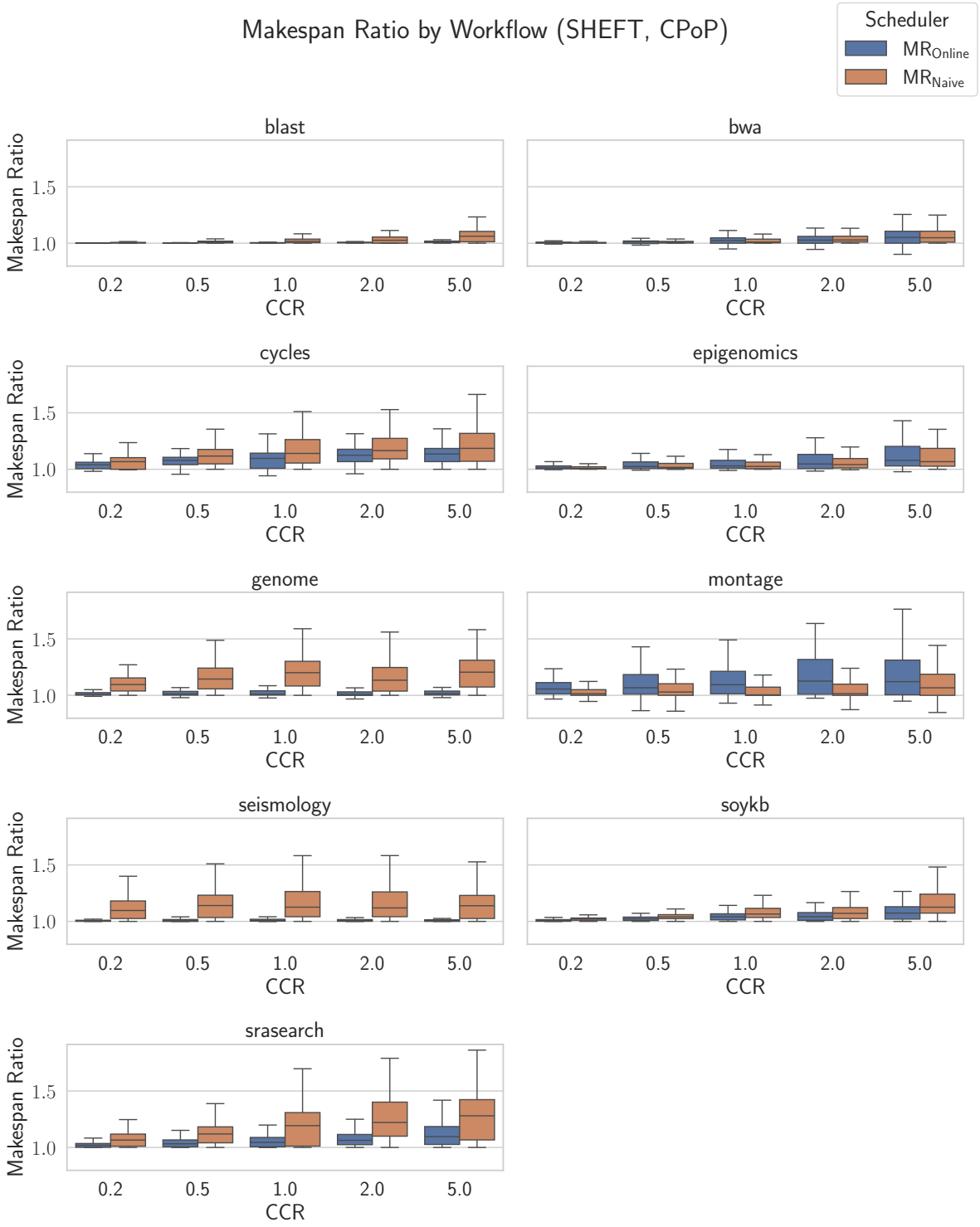


Figure 6: Normalized makespan ratios by workflow using the SHEFT estimator with the CPoP scheduling heuristic. Online scheduling (blue) often improves both median performance and spread relative to the naive baseline (orange), although results vary by workflow. Notably, Montage performs worse with online re-scheduling.

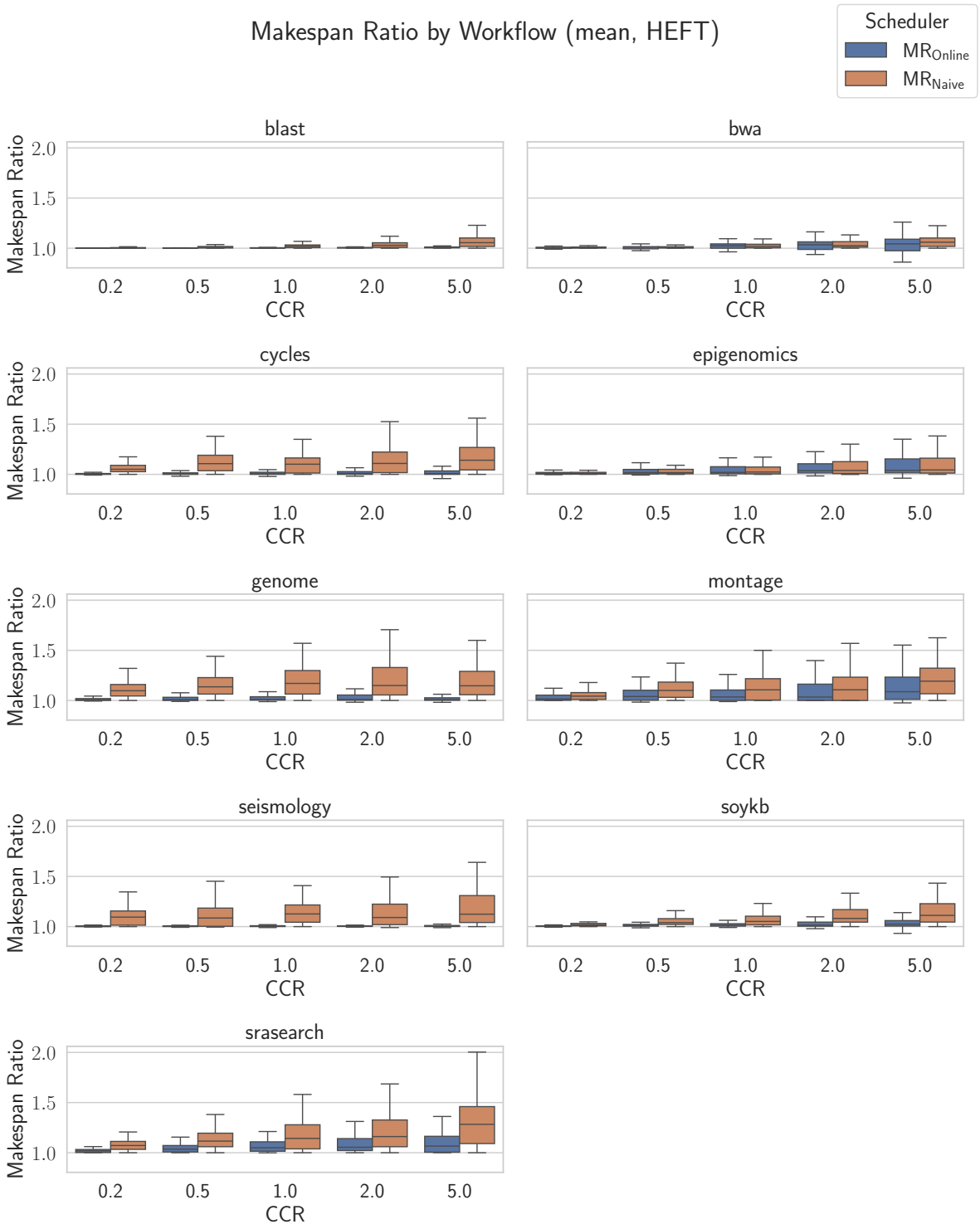


Figure 7: Normalized makespan ratios by workflow using the mean estimator with the HEFT scheduling heuristic. Online scheduling (blue) often improves both median performance and spread relative to the naive baseline (orange), although results vary by workflow. Notably, Montage performs worse with online re-scheduling.

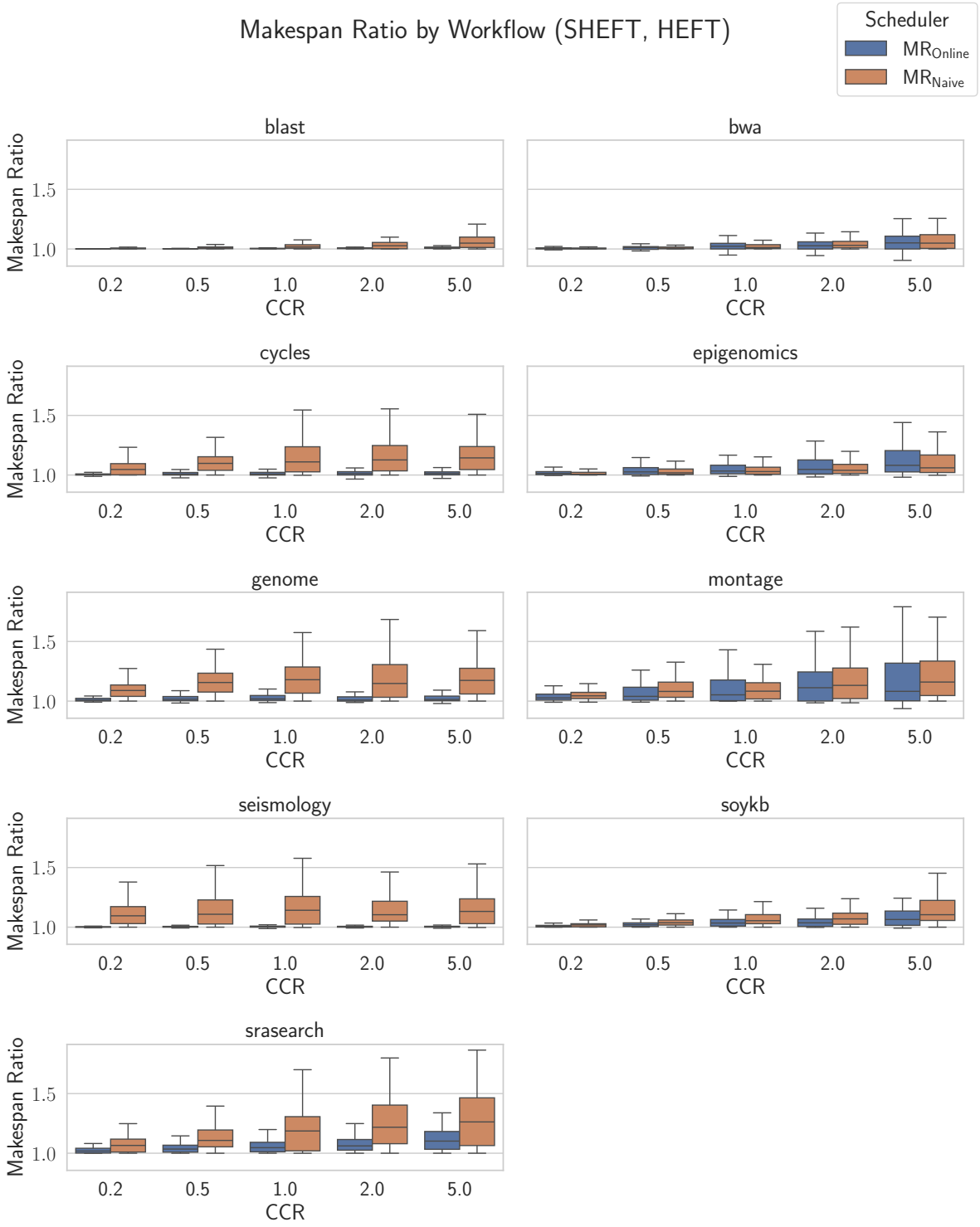


Figure 8: Normalized makespan ratios by workflow using the SHEFT estimator with the HEFT scheduling heuristic. Online scheduling (blue) often improves both median performance and spread relative to the naive baseline (orange), although results vary by workflow. Notably, Montage performs worse with online re-scheduling.

7 Conclusion

We presented an automated framework for online task scheduling in heterogeneous distributed systems, enabling classical list-scheduling heuristics such as HEFT and CPoP to operate effectively in real-time environments with runtime uncertainty. By extending a modular parametric scheduler with online execution semantics and feedback-driven adaptation, our system automatically adapts a broad family of offline schedulers to work in online contexts. Empirical evaluation on large-scale, real-world scientific workflows showed that our adaptive schedulers consistently outperform naive baselines, achieving performance within 3–5% of an ideal offline scheduler (compared to an approximately 10% degradation for naive approaches). In addition to improving mean makespan, our approach significantly reduces variance, making execution more predictable under uncertainty.

There are several promising directions for future work. One natural extension is to incorporate dynamic estimation models that adapt to observed execution behavior over time, allowing the scheduler to refine its predictions based on feedback. Another avenue is the integration of reinforcement learning into the parametric scheduling framework, potentially enabling hybrid strategies that learn when and how to reconfigure scheduling parameters. Although our current design focuses on minimizing makespan, many real-world systems must also consider energy efficiency, resource cost, or other multi-objective trade-offs. Extending the framework to accommodate such goals is a valuable next step. Also, while our current method re-plans the entire schedule after every task completion, it may be beneficial to explore more selective or window-based re-scheduling strategies to reduce computational overhead without sacrificing adaptability. Future work should also analyze the computational overhead of re-scheduling and explore strategies to mitigate it through the identification of key re-scheduling opportunities. Furthermore a robust evaluation requires benchmarking and turning our focus toward other online schedulers and employing more realistic interdependent task cost models. Finally, validating the framework in real-world workflow systems or distributed computing platforms would provide critical insights into its robustness under practical deployment conditions, such as queueing delays, heterogeneous load, and data locality constraints.

Overall, our results suggest that even lightweight online adaptations of classic heuristics can yield substantial performance benefits in dynamic environments. We hope this framework provides a foundation for further research on adaptive, modular scheduling in modern heterogeneous computing ecosystems.

Acknowledgments

This material is based upon work supported by the National Science Foundation under Award No. 2451267. This work was supported in part by Army Research Laboratory under Cooperative Agreement W911NF-17-2-0196.

References

[1] Jonathan Bader, Ansgar Lößer, Lauritz Thamsen, Björn Scheuermann, and Odej Kao. 2024. Ks+: Predicting workflow task memory usage over time. In *2024 IEEE 20th International Conference on e-Science (e-Science)*. IEEE, 1–7.

[2] Artem M Chirkin, Adam SZ Belloum, Sergey V Kovalchuk, Marc X Makkes, Mikhail A Melnik, Alexander A Visheratin, and Denis A Nasonov. 2017. Execution time estimation for workflow scheduling. *Future generation computer systems* 75 (2017), 376–387.

[3] Jared Coleman, Ravi Vivek Agrawal, Ebrahim Hirani, and Bhaskar Krishnamachari. 2025. Parameterized Task Graph Scheduling Algorithm for Comparing Algorithmic Components. In *Proceedings of the 29th Workshop on Job Scheduling Strategies for Parallel Processing (JSSPP)*. arXiv:2403.07112 [cs.DC] https://arxiv.org/abs/2403.07112 To appear.

[4] Taina Coleman, Henri Casanova, and Rafael Ferreira Da Silva. 2021. Wfchef: Automated generation of accurate scientific workflow generators. In *2021 IEEE 17th International Conference on eScience (eScience)*. IEEE, 159–168.

[5] Tainã Coleman, Henri Casanova, Loïc Pottier, Manav Kaushik, Ewa Deelman, and Rafael Ferreira da Silva. 2022. Wfcommons: A framework for enabling scientific workflow research and development. *Future generation computer systems* 128 (2022), 16–27.

[6] Mehdi Hosseinzadeh, Marwan Yassin Ghafour, Hawkar Kamaran Hama, Bay Vo, and Afsane Khoshnevis. 2020. Multi-objective task and workflow scheduling approaches in cloud computing: a comprehensive review. *Journal of Grid Computing* 18, 3 (2020), 327–356.

[7] Alexey Ilyushkin and Dick Epema. 2018. The impact of task runtime estimate accuracy on scheduling workloads of workflows. In *2018 18th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID)*. IEEE, 331–341.

[8] Yu-Kwong Kwok and I. Ahmad. 1996. Dynamic critical-path scheduling: an effective technique for allocating task graphs to multiprocessors. *IEEE Transactions on Parallel and Distributed Systems* 7, 5 (1996), 506–521. doi:10.1109/71.503776

[9] Kevin Lee, Norman W Paton, Rizos Sakellariou, Ewa Deelman, Alvaro AA Fernandes, and Gaurang Mehta. 2009. Adaptive workflow processing and execution in pegasus. *Concurrency and Computation: Practice and Experience* 21, 16 (2009), 1965–1981.

[10] Suraj Pandey, Linlin Wu, Siddeswara Mayura Guru, and Rajkumar Buyya. 2010. A particle swarm optimization-based heuristic for scheduling workflow applications in cloud computing environments. In *2010 24th IEEE international conference on advanced information networking and applications*. IEEE, 400–407.

[11] Limei Peng, Ahmad R. Dhaini, and Pin-Han Ho. 2018. Toward integrated Cloud–Fog networks for efficient IoT provisioning: Key challenges and solutions. *Future Generation Computer Systems* 88 (2018), 606–613. doi:10.1016/j.future.2018.05.015

[12] Gonzalo P Rodrigo, Erik Elmroth, Per-Olov Östberg, and Lavanya Ramakrishnan. 2017. Enabling workflow-aware scheduling on hpc systems. In *Proceedings of the 26th International Symposium on High-Performance Parallel and Distributed Computing*. 3–14.

[13] Xiaoyong Tang, Kenli Li, Guiping Liao, Kui Fang, and Fan Wu. 2011. A stochastic scheduling algorithm for precedence constrained tasks on Grid. *Future Generation Computer Systems* 27, 8 (2011), 1083–1091. doi:10.1016/j.future.2011.04.007

[14] T.-S. Tia, Z. Deng, M. Shankar, M. Storch, J. Sun, L.-C. Wu, and J.W.-S. Liu. 1995. Probabilistic performance guarantee for real-time tasks with varying computation times. In *Proceedings Real-Time Technology and Applications Symposium*. 164–173. doi:10.1109/RTTAS.1995.516213

[15] H. Topcuoglu, S. Hariri, and Min-You Wu. 2002. Performance-effective and low-complexity task scheduling for heterogeneous computing. *IEEE Transactions on Parallel and Distributed Systems* 13, 3 (2002), 260–274. doi:10.1109/71.993206

[16] Benjamin Tovar, Rafael Ferreira Da Silva, Gideon Juve, Ewa Deelman, William Allcock, Douglas Thain, and Miron Livny. 2017. A job sizing strategy for high-throughput scientific workflows. *IEEE Transactions on Parallel and Distributed Systems* 29, 2 (2017), 240–253.

[17] Wei Wang, Guosun Zeng, Daizhong Tang, and Jing Yao. 2012. Cloud-DLS: Dynamic trusted scheduling for Cloud computing. *Expert systems with applications* 39, 3 (2012), 2321–2329.

[18] Carl Witt, Dennis Wagner, and Ulf Leser. 2019. Feedback-based resource allocation for batch scheduling of scientific workflows. In *2019 International Conference on High Performance Computing & Simulation (HPCS)*. IEEE, 761–768.

[19] Lingyun Yang, Jennifer M Schopf, and Ian Foster. 2003. Conservative scheduling: Using predicted variance to improve scheduling decisions in dynamic environments. In *Proceedings of the 2003 ACM/IEEE conference on Supercomputing*. 31.

[20] Andy B. Yoo, Morris A. Jette, and Mark Grondona. 2003. SLURM: Simple Linux Utility for Resource Management. In *Job Scheduling Strategies for Parallel Processing*, Dror Feitelson, Larry Rudolph, and Uwe Schwiegelshohn (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 44–60.

[21] Zhifeng Yu and Weisong Shi. 2007. An Adaptive Rescheduling Strategy for Grid Workflow Applications. In *2007 IEEE International Parallel and Distributed Processing Symposium*. 1–8. doi:10.1109/IPDPS.2007.370305

[22] Junlong Zhou, Mingyue Zhang, Jin Sun, Tian Wang, Xiumin Zhou, and Shiyan Hu. 2022. DRHEFT: Deadline-Constrained Reliability-Aware HEFT Algorithm for Real-Time Heterogeneous MPSoC Systems. *IEEE Transactions on Reliability* 71, 1 (2022), 178–189. doi:10.1109/TR.2020.2981419