Robustness–Efficiency Trade-offs in Learning-Aware Distributed Scheduling

Mehedi Hasan [1] **Aidan Peña** [1]

Abstract

Learning-aware schedulers promise improved efficiency for distributed machine learning workloads by exploiting task-specific convergence behavior. However, when deployed in resource-constrained edge environments, these systems must also contend with network congestion and energy limitations. In this work, we evaluate classical scheduling baselines alongside Learning-Aware Resource Allocation (LARA) and its integration with Utility-Aware Resource Allocation (UARA). While UARA-enhanced methods achieve superior energy efficiency, they suffer severe degradation in task completion under heavy load. Through detailed simulation and code-level analysis, we identify a normalization mismatch in the congestion-control mechanism that induces self-inflicted resource starvation. Our findings highlight a fundamental robustness–efficiency trade-off and underscore the importance of implementation-aligned system design.

1. Introduction

Distributed machine learning workloads are increasingly deployed in edge computing environments, where compute resources are limited, network conditions are volatile, and energy efficiency is critical. Unlike traditional batch jobs, ML training exhibits diminishing returns: early computation yields greater reductions in loss than later updates. This observation has motivated the development of learning-aware schedulers that allocate resources based on predicted training progress.

Learning-Aware Resource Allocation (LARA) leverages estimated learning curves to prioritize tasks expected to benefit most from additional computation. Under light load and idealized assumptions, such strategies outperform classical schedulers. However, real edge systems must also contend with shared network bandwidth, communication delays, and energy constraints.

To address these challenges, we integrate Utility-Aware Resource Allocation (UARA), a congestion-aware throttling mechanism designed to reduce energy consumption during periods of high network utilization. Unexpectedly, the combined UARA+LARA strategy performs significantly worse than simpler baselines under heavy load. This paper investigates why.

2. Literature Review

Classical schedulers such as Uniform allocation and Shortest-Deadline-First (SDF) remain widely used due to their predictability and robustness. Uniform scheduling ensures fairness but ignores urgency, while SDF prioritizes deadlines at the risk of starving long-running tasks.

Recent work has explored learning-aware scheduling for ML workloads. Wang et al. (2) introduce LARA, demonstrating that learning-curve estimation can significantly improve efficiency. Zhou (3) studies learnability under shared computational constraints, highlighting the interaction between scheduling and convergence.

Separately, Kim et al. (1) propose UARA for latency-sensitive mobile edge computing, focusing on congestion-aware resource throttling. However, the interaction between learning-aware estimation and utility-based congestion control remains underexplored.

3. Data, Model, and Compute Baselines

3.1. System Model

We simulate an edge cluster consisting of five compute nodes, each capable of processing 30 samples per timestep. Network bandwidth is capped at 100 samples per timestep, with an average transmission delay of three timesteps and stochastic jitter. Idle servers incur an energy cost, encouraging full utilization unless throttled.

3.2. Workload Model

Tasks arrive dynamically and must first transmit data before computation begins. Each task follows a noisy power-law learning curve:

$$\epsilon(s) = as^{-b} + \mathcal{N}(0, \sigma), \tag{1}$$

where s is the number of processed samples. Tasks have heterogeneous deadlines and initial learning parameters.

3.3. Baselines

We evaluate five scheduling strategies, ranging from simple classical approaches to learning- and network-aware methods:

1. **Uniform allocation:** Each task receives an equal share of compute and network resources, regardless of deadline or learning progress. This strategy ensures fairness and prevents starvation but ignores task urgency and potential efficiency gains.
2. **Shortest-Deadline-First (SDF):** Tasks are prioritized based on the proximity of their deadlines. This approach aims to maximize on-time task completion, but long-running tasks may experience starvation if many short-deadline tasks arrive simultaneously.
3. **Learning-Aware Resource Allocation (LARA):** LARA predicts task progress using online learning-curve estimation and dynamically allocates compute resources to tasks expected to benefit most from additional computation (2). It balances exploration (gathering progress information) and exploitation (optimizing resource use) to improve task completion efficiency.
4. **Utility-Aware Resource Allocation (UARA) + Uniform:** UARA manages network and server resources to reduce congestion and energy use (1). When combined with uniform compute allocation, it ensures fair distribution of compute resources while throttling task submissions and network usage under heavy load.
5. **UARA + LARA:** Integrates network-aware UARA with task-aware LARA, aiming to jointly optimize latency, energy, and task completion. UARA regulates bandwidth and device-server associations, while LARA allocates compute resources based on predicted learning progress. This combined approach seeks to leverage the strengths of both strategies but may be sensitive to misalignment in resource normalization under high load.

3.4. From Demo to Final Implementation

The original demo implemented a simple simulation of task scheduling with three strategies: Uniform, Shortest-Deadline-First (SDF), and a LARA-style scheduler. Tasks were few in number, and allocation ignored network effects and energy constraints, limiting realism.

Key limitations of the original demo included:

- **Simplified resource model:** Total server power was split statically, without per-server constraints or network delays.
- **Limited task diversity:** Task arrivals were deterministic, and workloads were small (typically 6 tasks).
- **No network modeling:** Transmission delays and congestion were ignored.
- **Basic energy accounting:** Energy usage per server or idle cost was not considered.
- **Minimal scalability:** The simulation horizon and task counts were small, with no quantitative analysis for larger workloads.

The final implementation addressed these limitations:

- **Network-aware modeling:** Tasks now experience transmission delays and stochastic jitter, and UARA regulates bandwidth and task-server associations.
- **Cluster-based resource allocation:** Multiple servers with per-server power caps enable realistic allocation and energy tracking.
- **Enhanced LARA scheduler:** Robust log-linear curve fitting with safeguards improves remaining-sample estimation under sparse updates.
- **Energy tracking and efficiency metrics:** Energy per server and overall efficiency (success per unit energy) are computed at each timestep.
- **Scalability and analysis framework:** Supports larger task sets (10–60 tasks) with plots and tables for detailed performance metrics.
- **Improved code structure:** Modular design, reproducible seeding, and dataclasses facilitate experimentation and extension.

These improvements make the simulation more realistic and extensible, revealing how network congestion and energy constraints interact with learning-aware scheduling to expose robustness–efficiency trade-offs.

4. Our Innovation

Our main contribution is a detailed study of how learning-aware and congestion-aware scheduling interact in realistic edge environments. By gradually adding network delays, jitter, and energy constraints to our simulations, we reveal failure modes that simple models miss, and we trace these failures to a specific normalization error in the code.

5. Results

Figure 1 shows the performance comparison across increasing task load for all scheduling strategies.

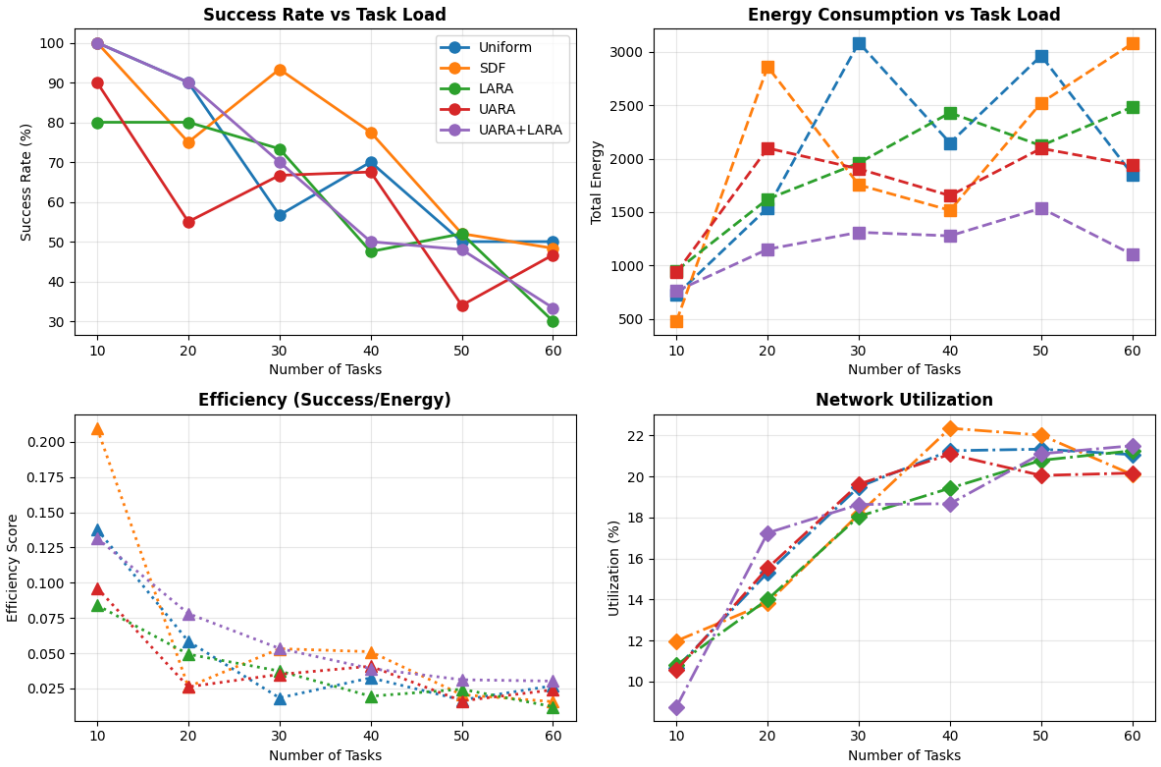

Figure 1. Performance comparison across increasing task load. Top-left: success rate. Top-right: total energy consumption. Bottom-left: efficiency (success per energy). Bottom-right: network utilization.

5.1. Visualization Across Load Conditions

Success Rate. All strategies perform well under light load. As task count increases, learning-aware strategies degrade more rapidly. At 60 tasks, Uniform scheduling completes roughly 50% of tasks, outperforming both LARA (30%) and UARA+LARA (33%).

Energy Consumption. UARA-based strategies significantly reduce energy usage under congestion. UARA+LARA consistently achieves the lowest total energy consumption, particularly at high load.

Efficiency. Measured as successful completions per unit energy, UARA+LARA achieves the highest efficiency despite poor absolute success rates, indicating excessive conservatism rather than inefficacy.

Network Utilization. Network utilization converges across strategies at high load, suggesting that performance differences arise primarily from allocation logic rather than bandwidth availability.

Table 1. Performance summary for 60 tasks across different scheduling strategies.

Strategy	Success (%)	Latency	Energy	Efficiency
Uniform	50.0	27.4	1852	0.0270
SDF	48.3	32.6	3080	0.0157
LARA	30.0	29.0	2485	0.0121
UARA	46.7	28.5	1940	0.0241
UARA+LARA	33.3	32.2	1101	0.0303

5.2. Performance Summary for 60 Tasks

Table 1 shows the performance summary for 60 tasks across different scheduling strategies.

6. Failure Modes and Problems in the Code

6.1. The UARA Normalization Paradox

The most severe issue arises in the UARA network allocator function. Allocation weights are normalized over all active tasks, including those still transmitting data. However, the resulting cap is applied only to tasks that are actively computing.

As network congestion increases, the number of non-computing tasks inflates the denominator, driving per-task compute allocation toward zero:

$$\text{Alloc}_i \propto \frac{1}{N_{\text{total}}} \ll \frac{1}{N_{\text{compute}}}.$$

This causes the cluster to idle while tasks wait, producing the starvation behavior observed in Figure 1.

6.2. Estimation Fragility

LARA depends on accurate learning-curve estimation. Under heavy load, tasks receive sparse and irregular updates, increasing estimation noise and leading to misprioritization.

6.3. Visualization Inconsistencies

Early versions of the plotting code used inconsistent markers and styles across metrics, complicating interpretation. While not affecting execution, this obscured cross-metric trends and was corrected in later revisions.

7. Discussion

The results reveal a fundamental robustness–efficiency trade-off. Simple schedulers waste energy but guarantee progress, while sophisticated schedulers optimize aggressively and become brittle under uncertainty.

Importantly, the failure of UARA+LARA is not theoretical. A single normalization mismatch was sufficient to negate the benefits of learning-aware scheduling. As systems incorporate richer feedback loops, correctness becomes increasingly sensitive to implementation alignment.

8. Conclusion and Future Work

We show that combining learning-aware scheduling with congestion-aware throttling can lead to catastrophic failure if system state and allocation logic are misaligned. While UARA+LARA achieves impressive energy efficiency, it sacrifices robustness under heavy load.

Future work will explore corrected normalization, minimum allocation guarantees, and hybrid schedulers that dynamically revert to robust baselines under high uncertainty.

References

[1] J. Kim, H. Park, and S. Lee. Distributed task offloading and resource allocation for latency-sensitive mobile edge computing. *arXiv preprint arXiv:2407.XXXX*, 2024.

[2] J. Wang, M. Yu, P. Zhao, and Z.-H. Zhou. Learning with adaptive resource allocation. In *Proceedings of the 41st International Conference on Machine Learning*, PMLR 235, 2024.

[3] Z.-H. Zhou. Learnability with time-sharing computational resource concerns. *arXiv preprint arXiv:2305.02217*, 2023.