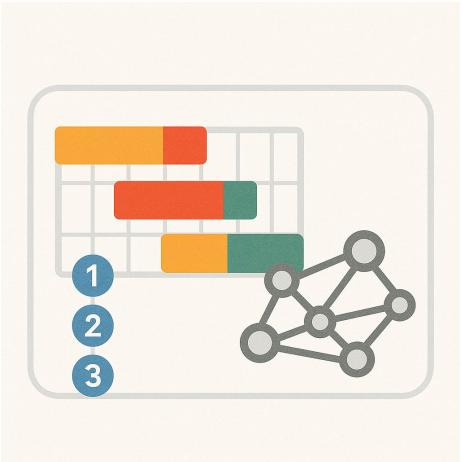
Learning to Schedule: A Supervised Learning Framework for Network-Aware Scheduling of Data-Intensive Workloads

INDIS Workshop, Supercomputing ‘25, Nov 16th, 2025

Sankalpa Timilsina, Susmit Shannigrahi
Department of Computer Science
Tennessee Technological University

Introduction

- Scheduling assigns job to resources over time.
  - Poor scheduling slows jobs, congests networks, and over- or under-utilizes computing nodes' capacity.
- Telemetry-informed scheduling can enable informed decisions.
  - Existing solutions like RL or heuristics are generally sample inefficient, require reward engineering, or do not generalize across heterogeneous environments.
- We propose a network-aware supervised-learning scheduler that predicts per-node job completion time from real-time telemetry.



Motivation

- Need of network-awareness in Data-intensive applications
  - Involves massive data shuffles and I/O operations that make the network a critical performance factor [1].
- Insurgence of Cloud-Native Applications
  - Over 89% of organizations adopted cloud-native platforms, with Kubernetes used by over 93% [2].
- Heuristics lack the ability to learn. RL-based systems typically require huge trails to converge on a good scheduling [3, 4].
  - Supervised models are good on generalizing, can be trained offline, and retraining does not require system downtime.

System Design

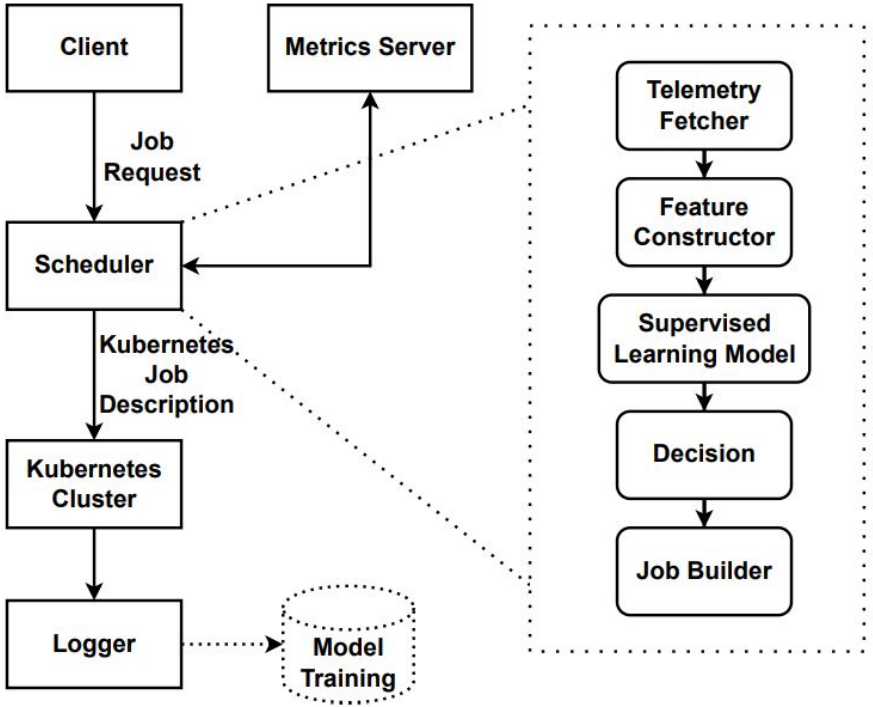
- Supervised learning-based scheduling framework for network-intensive Spark jobs.

Kubernetes Scheduling	Telemetry-aware scheduling
Blind to runtime factors such as network variability, CPU pressure, memory contention	Introduces live system signals into placement decisions. Includes: network telemetry (e.g., RTT, bandwidth usage), host-level metrics (e.g., CPU, memory pressure), or application traces.

System Design (Contd.)

Scheduler consists of five main components:

- **Client:** Initiates job submission. Includes: App-specific parameters (job size), input data size, resource requests, etc.
- **Metrics Server:** Aggregates and exposes system-level telemetry. Includes: RTTs, Tx/Rx rates, CPU, memory.



System Design (Contd.)

- **Scheduler**: Decision-making component. Five sub-modules.

Telemetry Fetcher	Gets real-time metrics from metrics server
Feature Constructor	Converts raw telemetry and job configurations into input vectors
Supervised Model	Linear Regression, Random Forest, XGBoost; predicts job completion per node
Decision Module	Ranks nodes based on predicted time
Job Builder	Builds Kubernetes-ready job manifest with node affinity

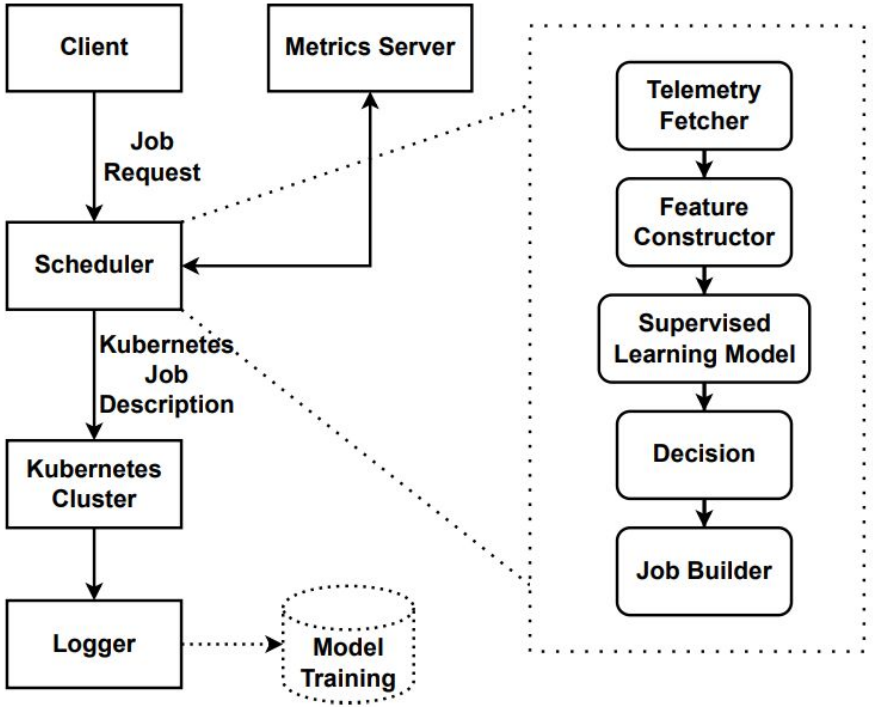
Table: Features Utilized

Feature	Description	Type
RTT	Mean, max, and standard deviation of RTT to all peers	Network
Tx/Rx Rate	Transmit and receive throughput (bytes/sec)	Network
CPU	CPU load average (runnable processes)	Node
Memory	Available memory (bytes)	Node
Application Type	Categorical job type (e.g., sort, join)	Job
Input Size	Size of input data (e.g., number of records)	Job
Other Configuration	Total executions, requested memory, etc.	Job

System Design (Contd.)

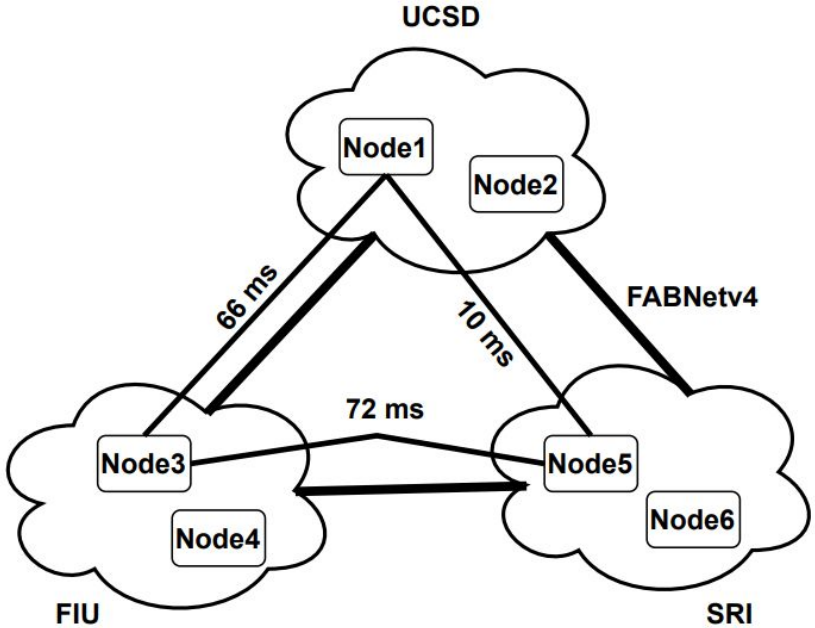
Scheduler consists of five main components:

- **Cluster**: Runs on geographically distributed Kubernetes cluster.
- **Logger**: Captures telemetry and performance data for future model training.



Evaluation

- Kubernetes cluster of 6 nodes across 3 sites
- 60 distinct job configurations across 3 Spark applications. A total of 3600 training samples with varying configurations.
- 3 regression models are trained.



Workloads

Spark applications with varying shuffle patterns (data-exchange) are chosen.

Application	Rationale
Sort	High network and CPU usage from large shuffles; moderate memory load
PageRank	High network and CPU usage from iterative data exchange; moderate memory load
Join	Skewed network, CPU, and memory usage due to imbalanced joins

Results

- We note the scheduler's choice of fastest or among the two fastest nodes.
- Supervised models achieved over 34-54% and 34-62% bigger Top-1 and Top-2 accuracies.

Method	Top-1	Top-2
Kubernetes Default	0.160	0.260
Linear Regression	0.500	0.600
XGBoost	0.560	0.720
Random Forest	**0.700**	**0.880**

Conclusion and Future Work

We show the supervised models outperformed the Kubernetes default scheduling behavior.

Future work:

- Larger-scale evaluation with real workloads such as ML pipelines and multi-stage streaming.
- Quantifying deployability and retraining costs.
- Finer network-telemetry integration, including per-interface throughput and buffer occupancy.

Email: stimilsin43@tntech.edu

References

1. Xin He and Prashant Shenoy. 2016. Firebird: Network-aware task scheduling for spark using sdns. In 2016 25th International Conference on Computer Communication and Networks (ICCCN). IEEE, 1–10.

2. CNCF. 2025. Cloud Native 2024: Approaching a Decade of Code, Cloud, and Change. https://www.cncf.io/reports/cncf-annual-survey-2024/

3. Khaldoun Senjab, Sohail Abbas, Naveed Ahmed, and Atta Ur Rehman Khan. 2023. A survey of Kubernetes scheduling algorithms. Journal of Cloud Computing 12, 1 (2023), 87.

4. Alejandro Del Real Torres, Doru Stefan Andreiana, Álvaro Ojeda Roldán, Alfonso Hernández Bustos, and Luis Enrique Acevedo Galicia. 2023. Deep Reinforcement Learning Approaches for Smart Manufacturing. Encyclopedia (2023). https://encyclopedia.pub/entry/40007 Entry adapted from Appl. Sci. 2022, 12, 12377.