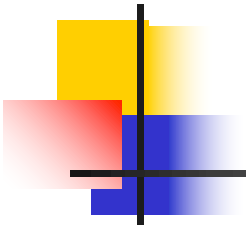
Machine Learning for Networking: Applications and Opportunities
-- Cluster Scheduling

Yan Li

Email: li.yan3@mail.mcgill.ca

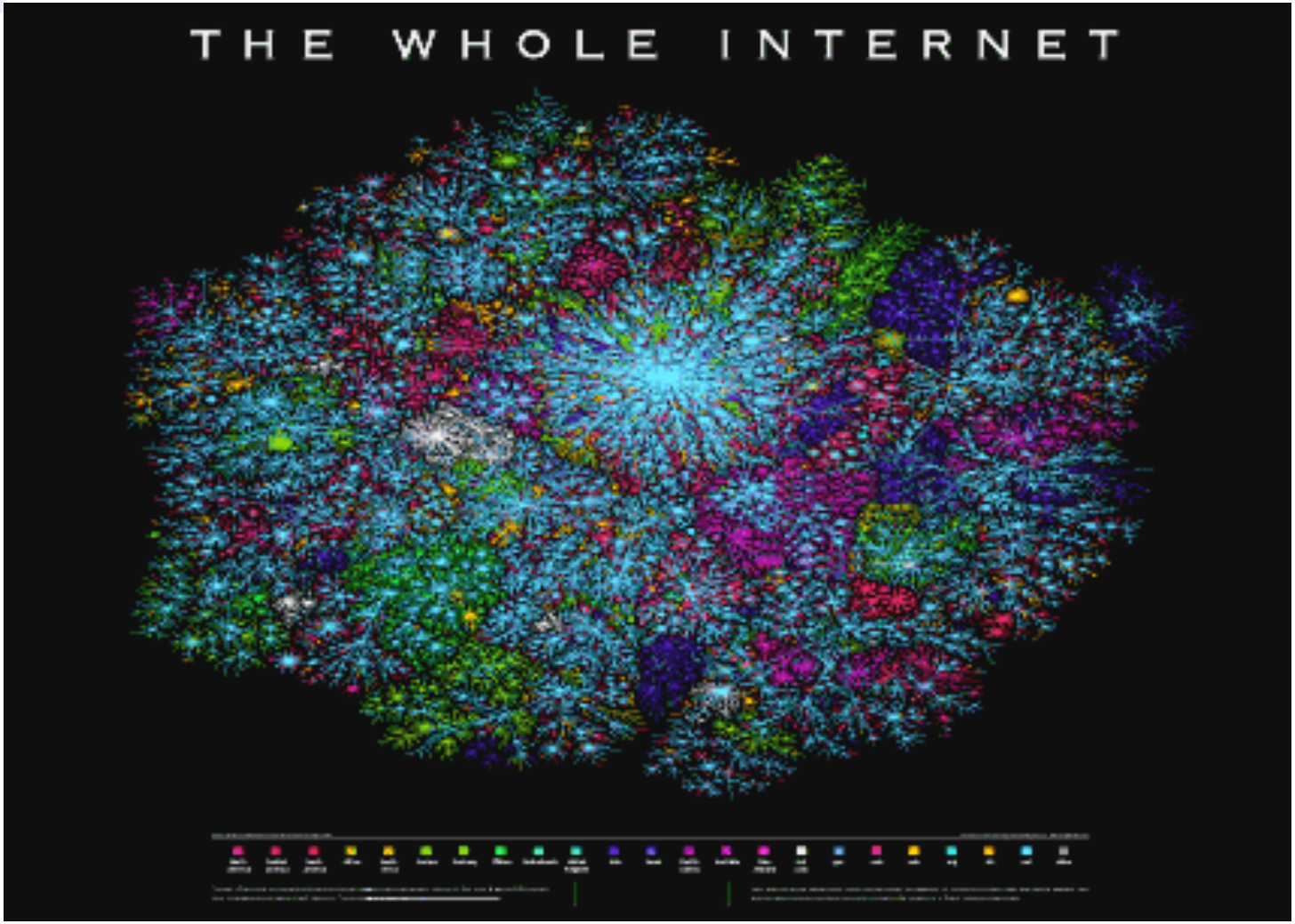
McGill



Outline

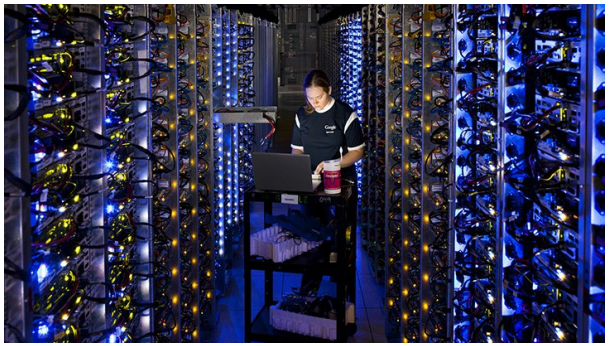
- **Introduction: Why We Need Intelligent Network?**
- What Networking Problems Can ML help?
  - Traffic Classification
  - Congestion Control
  - Resource Management
  - …
- Case Study: Cluster Scheduling
  - Decima, Sigcomm 2019
- Future: challenges and opportunities for MLN

We are more connected than ever



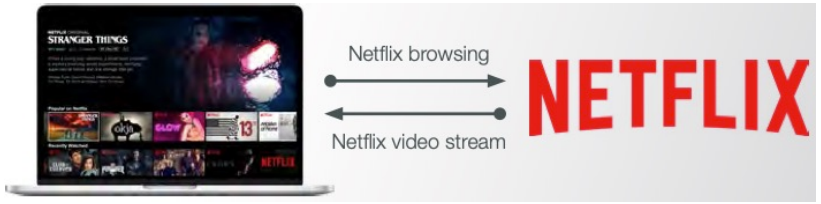
The network is more diverse than ever

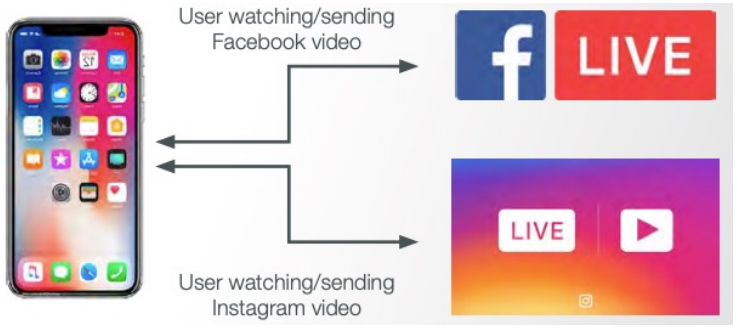
- Infrastructure diversity
  - Data center network (Google)
  - Mobile network (Rogers, Bell…)
  - Satellite network (Starlink)
- Application diversity
  - Video streaming (#1, 58%)
  - Multi-player online games
  - Social networking …
- Device diversity
  - laptops, tablets, smartphones
  - sensors, Web cams, game consoles, washing machines…



Network complexity and diversity cause challenges

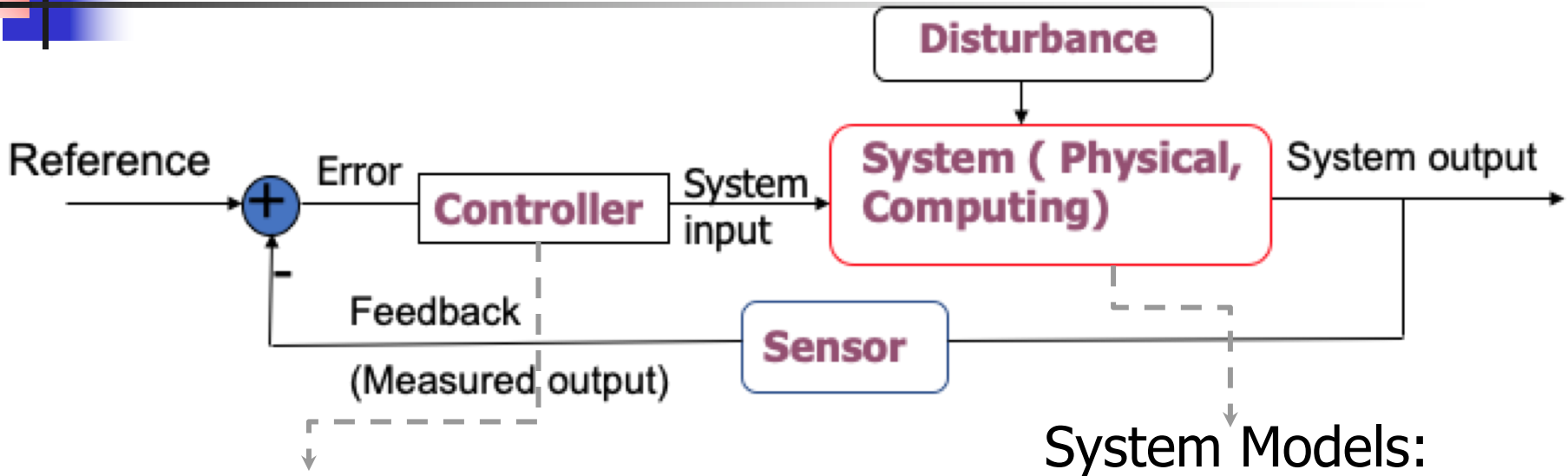
- Each network has its own features and characteristics
  - Overprovision for DCN
  - High variety for mobile network
- Each application has its own performance requirements
  - High throughput for video
  - Low latency for VoIP
- Each device has its own user demands
  - Security for smart car
  - Battery power for mobile phone





Specific algorithms are built for different network scenarios based on the network characteristics and user demands

What People Do Today?



Controller Designs:

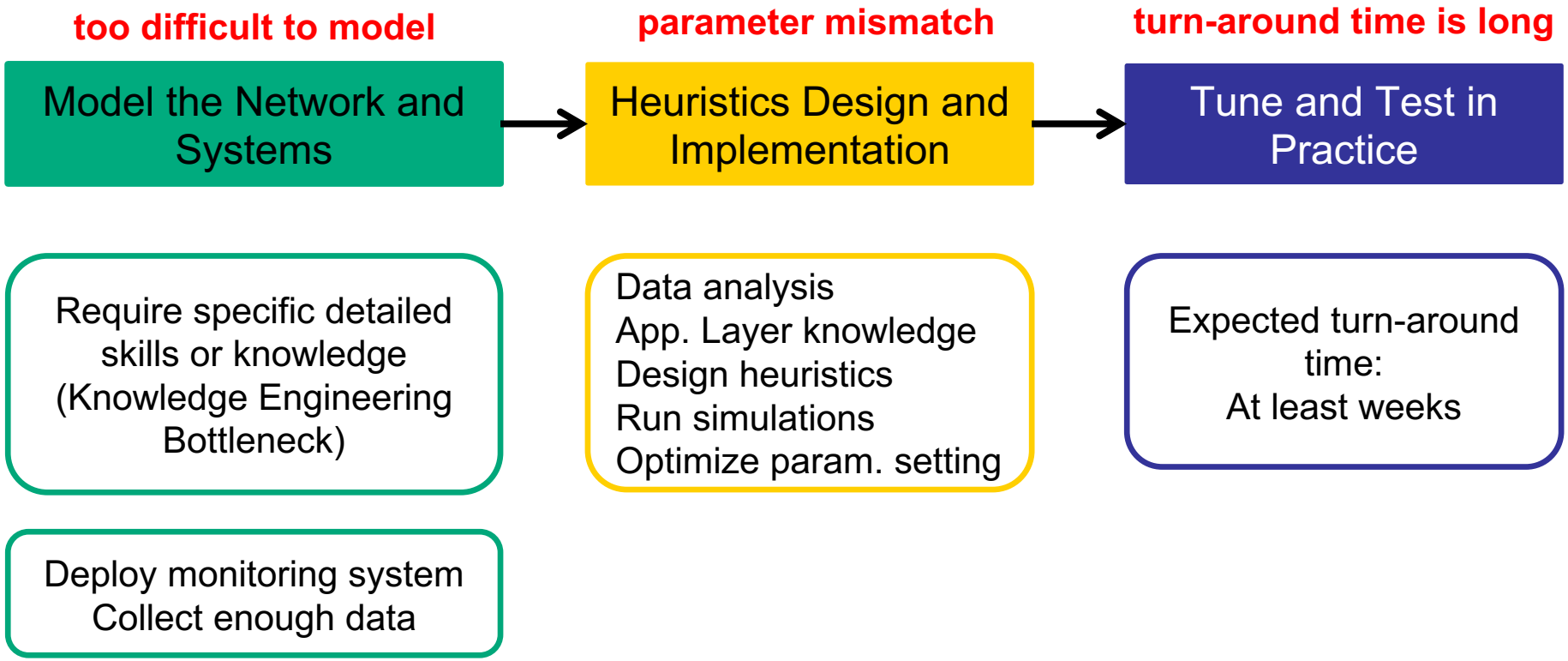
- PID
- Model Predictive Control (MPC)
- Adaptive control
- ……

System Models:

- First principles
  - Newton's laws
  - Queueing theory
- System identifications
- …….

1. Assume a simple model for the system
2. Come up with some clever heuristics
3. Painstakingly test and tune the heuristics in practice

The Hardness of Manually Heuristics

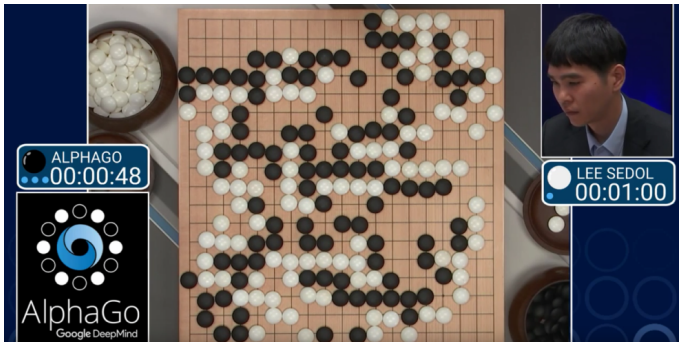


Is there a way around human-generated heuristics?

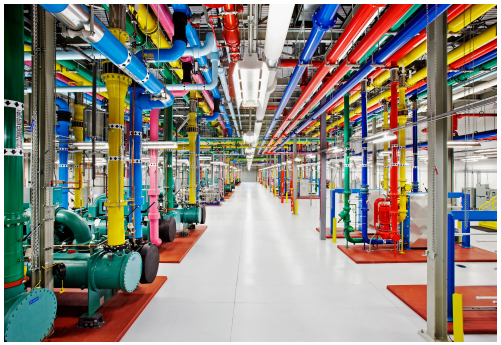
self-managed network and systems

Successes of Learning in Decision Making

Robotics control

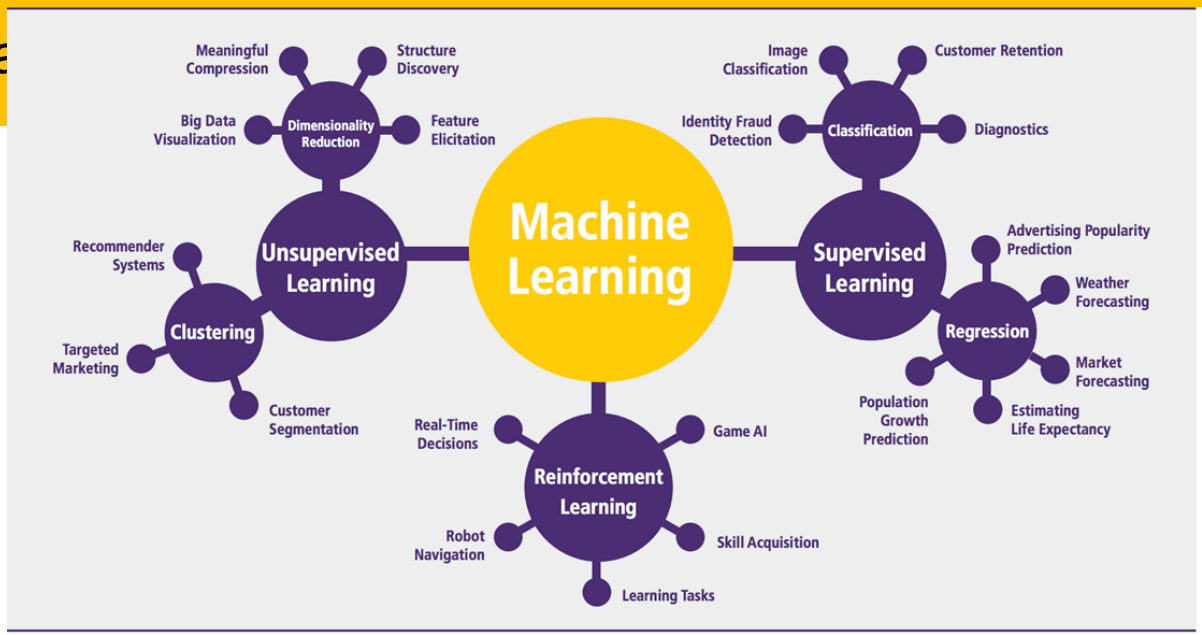


Game of Go

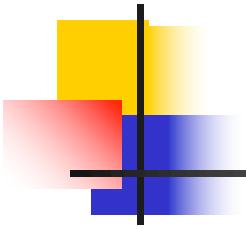


Datacenter cooling

Learning via



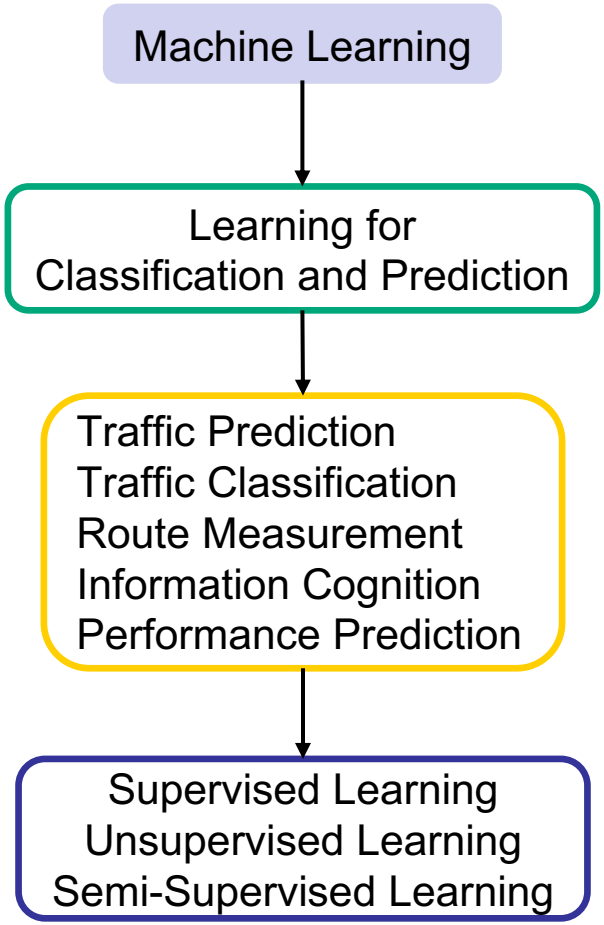
t Learning



Outline

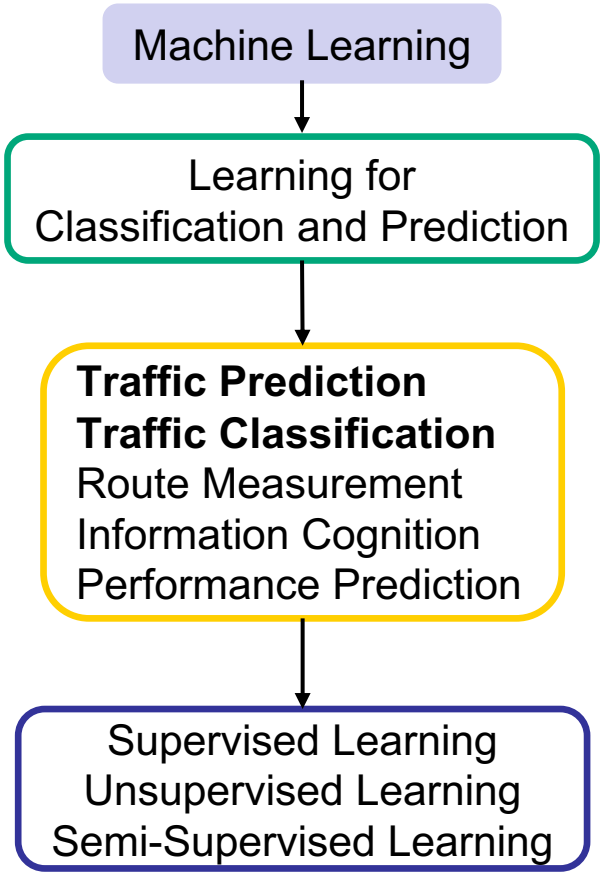
- Introduction: Why We Need Intelligent Network?
- **What Networking Problems Can ML help?**
  - Traffic Classification
  - Congestion Control
  - Resource Management
  - …
- Case Study: Cluster Scheduling
  - Decima, Sigcomm 2018
- Future: challenges and opportunities for MLN

Overview of Recent Applications



Learning for Classification and Prediction

- Traffic Prediction
  - Estimation of traffic volume (e.g., the traffic matrix)
  - Ref[1]: traffic volume prediction using supervised learning with Hidden-Markov Model
- Traffic Classification
  - Match traffic flows with the corresponding network applications and protocols
  - Fundamental component in security systems, thus high cost
  - RTC[2]: clustering and classification with SL and USL



[1] Z. Chen et al., "Predicting Future Traffic Using Hidden Markov Models," Proc. IEEE 24th Int'l. Conf. Network Protocols (ICNP) 2016, pp. 1–6.
[2] J. Zhang et al., "Robust Network Traffic Classification," IEEE/ ACM Trans. Networking (TON), vol. 23, no. 4, 2015, pp. 1257–70.

Overview of Recent Applications

Machine Learning

Learning for Classification and Prediction

Traffic Prediction
Traffic Classification
Route Measurement
Information Cognition
Performance Prediction

Supervised Learning
Unsupervised Learning
Semi-Supervised Learning

Learning for Acting and Planning

Cluster Scheduling
Video Streaming
Internet Telephony
VM Placement
Congestion Control

Online Decision Making

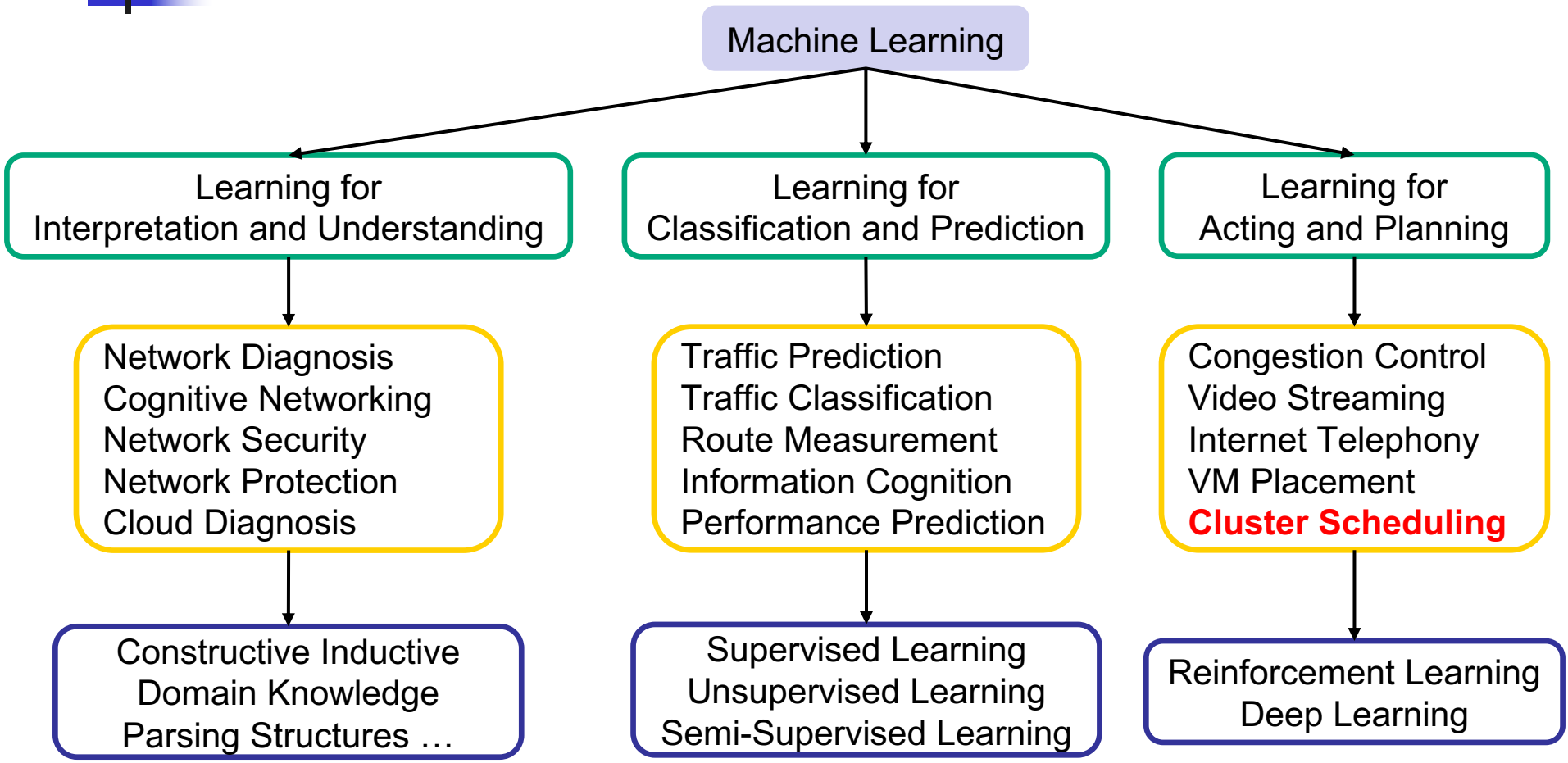
Reinforcement Learning
Deep Learning

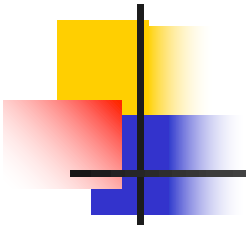
amazon web services™ | EC2

Google Cloud Platform

Microsoft Azure

Overview of Recent Applications



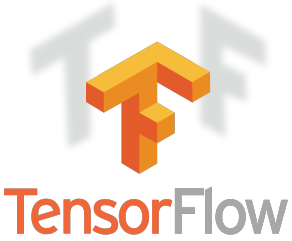


Outline

- Introduction: Why We Need Intelligent Network?
- What Networking Problems Can ML help?
  - Traffic Classification
  - Congestion Control
  - Resource Management
  - …
- **Case Study: Cluster Scheduling**
  - Decima, Sigcomm 2018
- Future: challenges and opportunities for MLN

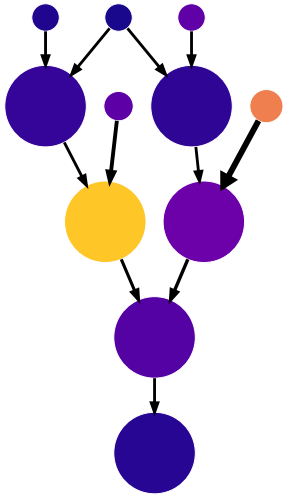
Scheduling is Ubiquitous

- Scheduling is a fundamental task in computer systems
  - Cluster management (e.g., Kubernetes, Mesos)
  - Data analytics frameworks (e.g., Spark, Hadoop)
  - Machine learning (e.g., TensorFlow)
- Cluster scheduling
  - scheduling on large clusters

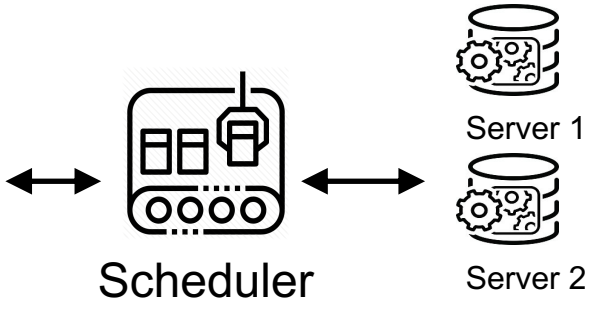


Cluster Scheduling Problem

- Cluster scheduling is an online resource allocation problem
  - Schedule – Mapping of tasks to machines and time
  - The schedule is subject to feasibility constraints and optimisation objectives.



Job 1

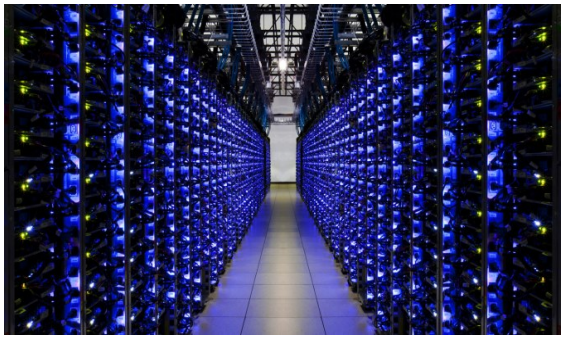


Goal: match tasks to machines to achieve:

- High cluster utilization
- Fast job completion
- Guarantees (deadlines, fair shares, …)

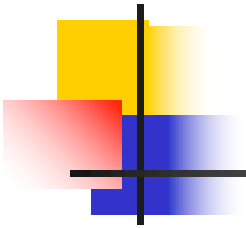
Efficiency Matters a Lot

- Efficient scheduler matters for large datacenters
  - Improving the efficiency of a scheduler is the key to save operation cost
  - Translates directly to cost arguments: better scheduling -> smaller clusters
  - Small improvement can save millions of dollars at scale



"as large as cities"

5%



How We Measure Efficiency

- Job flow time:
  - Time a job is completed minus the time the job was first available for processing;
- Average # jobs in system:
  - Measures amount of work-in-progress;
- Makespan:
  - The time it takes to finish a batch of jobs;
- Job lateness:
  - Whether the job is completed ahead of, on, or behind schedule;
- Job tardiness:
  - How long after the due date a job was completed, measures due date performance

Scheduling Efficiency Calculations

Job A finishes on day 10	Job B finishes on day 13	Job C finishes on day 17	Job D ends on day 20

Calculation mean flow time:

MFT= (sum job flow times)/ # of jobs

= (10+13+17+20)/4 = 60/4 = **15 days**

Calculating average number of jobs in the system:

Average # Jobs =(sum job flow times)/ # days to complete batch

= (60)/20 = **3 job**

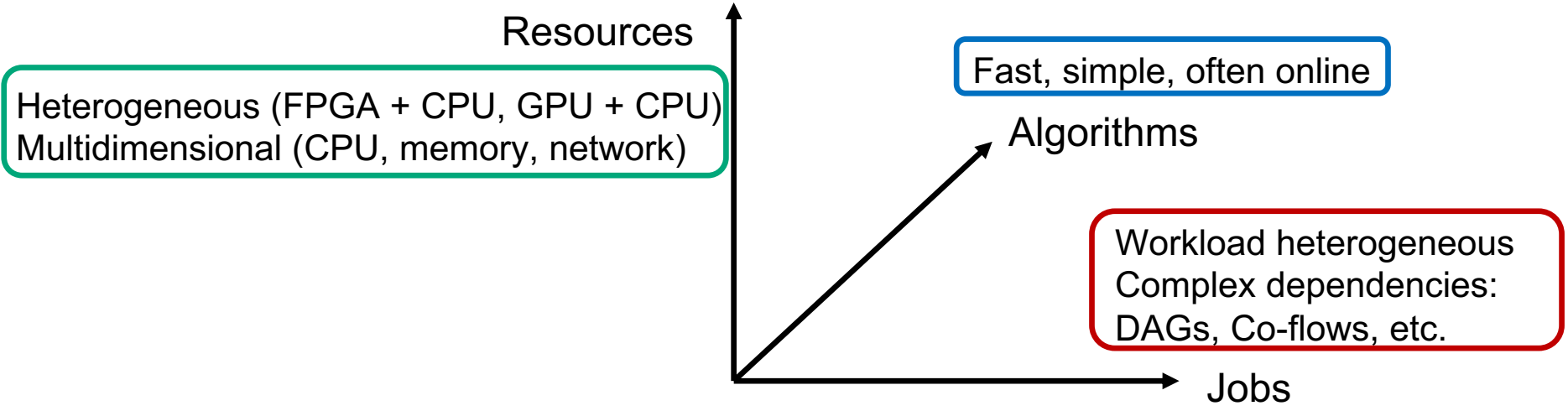
Makespan is the length of time to complete a batch

Makespan = Completion time for Job D minus start time for Job A

= 20 – 0 = 20 days

Designing efficient schedulers is intractable

- Must consider many factors for performance:
  - Placement constraints (e.g., special purpose hardware)
  - Modeling complexity (e.g., multiple resource types)
  - Workload heterogeneity (e.g., elephant flow)
  - Complex job structure (e.g., DAGs, co-flows)

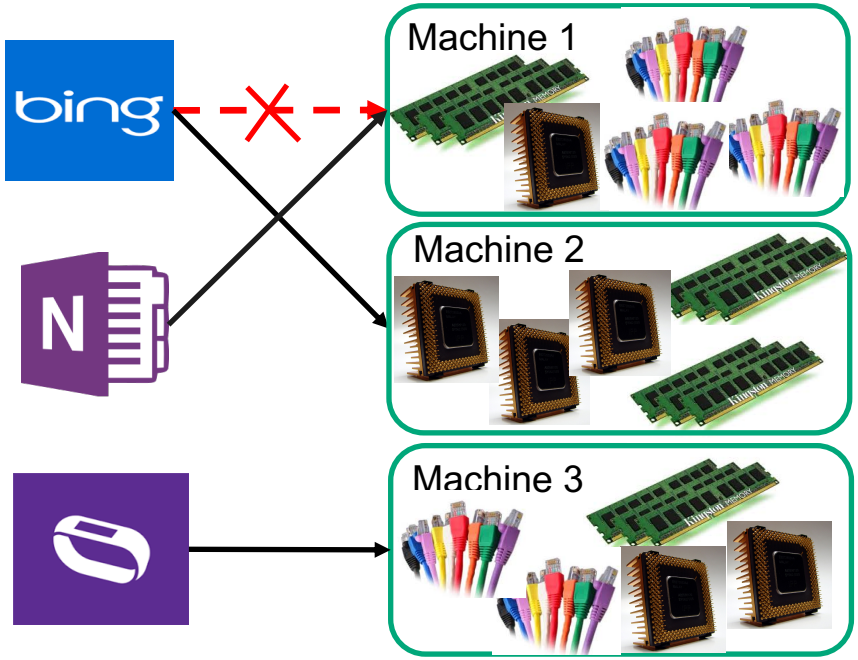


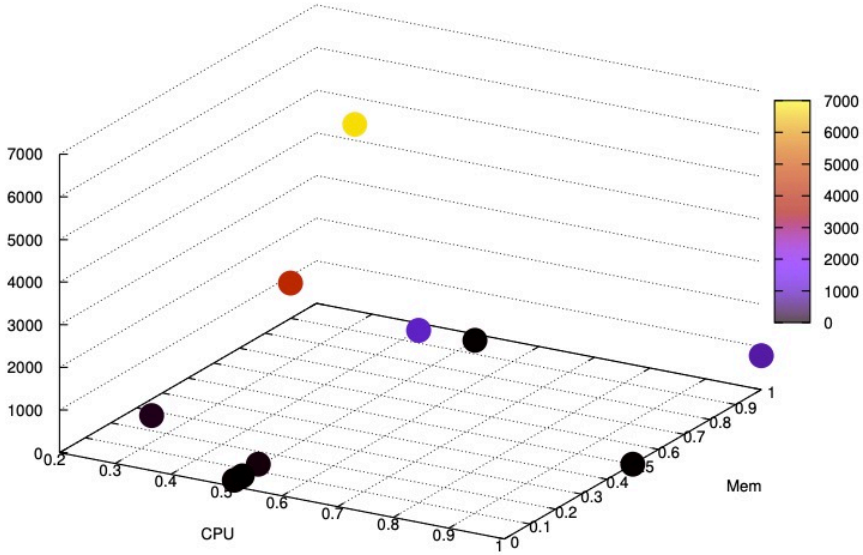
Placement Constraints

- Jobs are scheduled on heterogeneous clusters
- Must consider placement constraints for heterogeneous clusters:
  - Special purpose hardware (e.g., GPU, FPGA)
  - Data locality
  - Geographic location
  - Privacy concerns

Modeling Complexity

- Jobs run faster on some machines and slower on others
- Findings from Bing and Facebook traces analysis shows that jobs have (very) diverse resource needs
  - Tasks need varying amounts of each resource
  - Demands for resources are weakly correlated

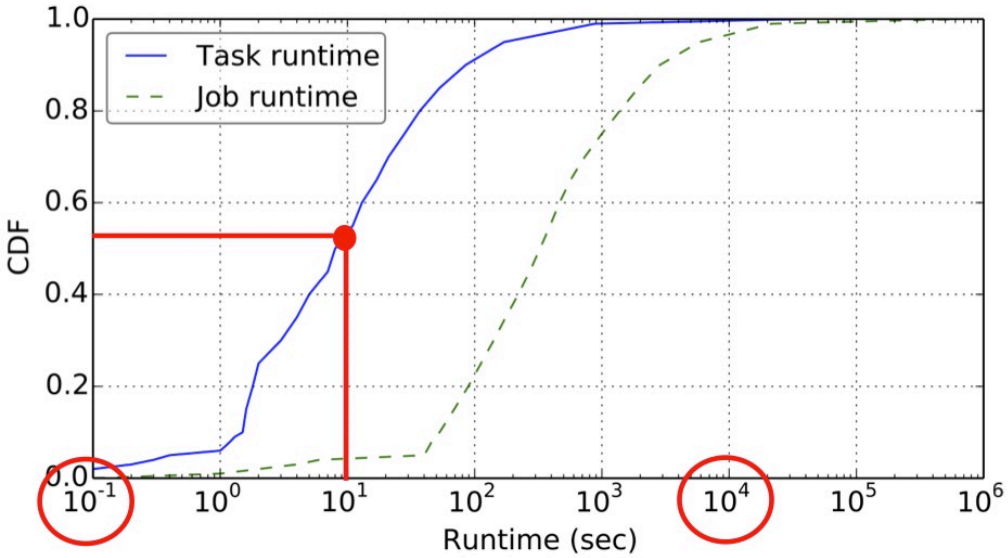




Number of machines (CPU, Mem) configuration
Minet, et al. "Analyzing traces from a Google data center"

Workload Heterogeneity

- Cluster scheduler must support heterogeneous workloads
  - Workloads can be made of a variety of applications
  - Workloads evolve in time
- Knowing your workload is fundamental!



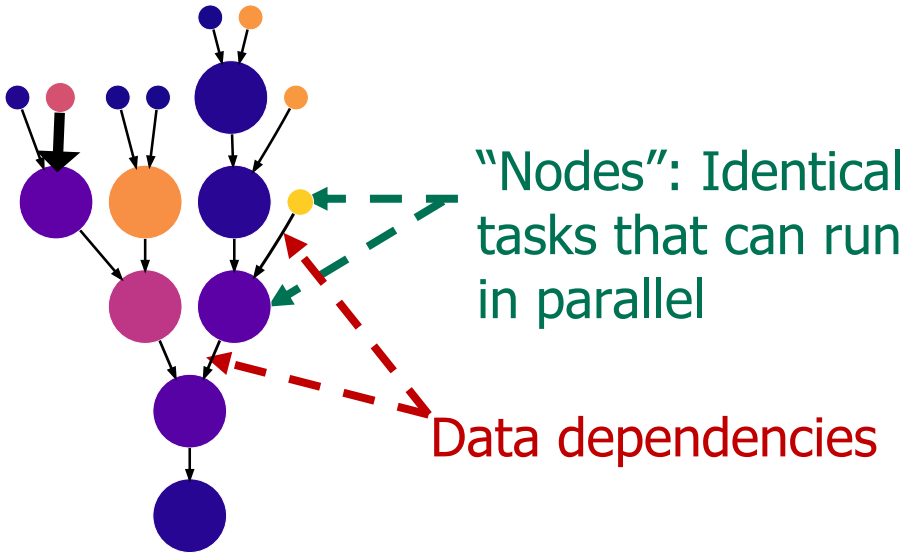
Workload heterogeneity in Cosmos

- Task runtime varies from sub-sec to 10,000+ sec
- 50% of tasks are shorter than 10 sec

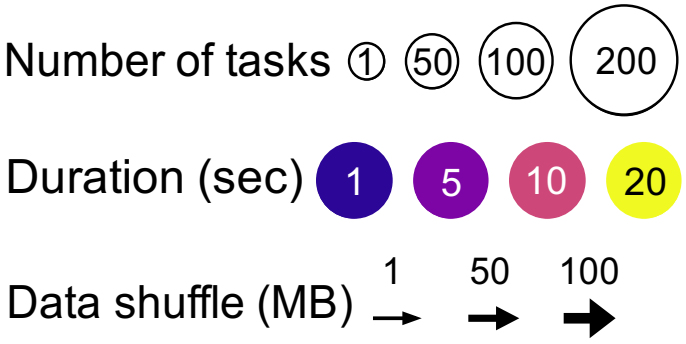
[7] Karanasos, Konstantinos, et al. "Mercury: Hybrid Centralized and Distributed Scheduling in Large Shared Clusters." USENIX ATC'15

Complex Job Structure

- Data processing systems and query compilers (e.g., Hive, Pig, Spark- SQL) create heterogeneous DAG-structured jobs.
  - DAGs have deep and complex structures.
  - Node (or stage) durations range from <1s to >100s
  - Each node has a different number of parallel tasks.



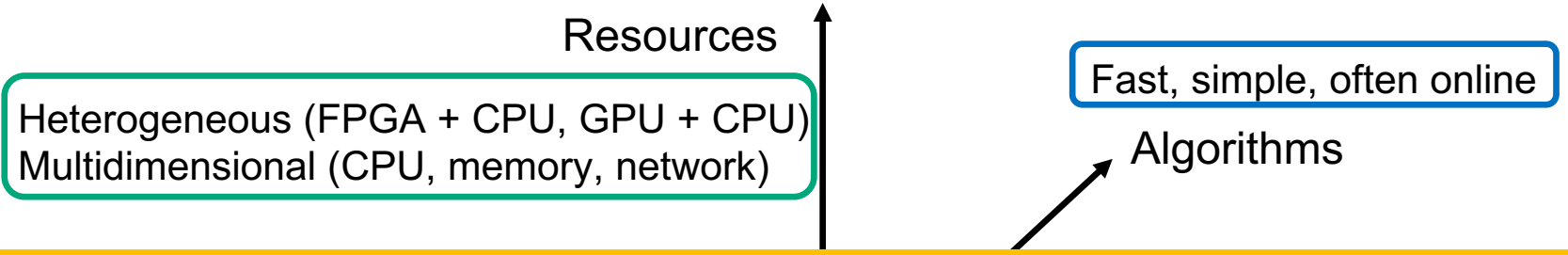
Job DAG
(example from TPC-H query in Spark)



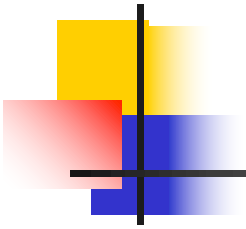
[6] Mao, Hongzi, et al. "Learning scheduling algorithms for data processing clusters." SIGCOMM'19

Designing efficient schedulers is intractable

- Must consider many factors for performance:
  - Placement constraints (e.g., special purpose hardware)
  - Modeling complexity (e.g., multiple resource types)
  - Workload heterogeneity (e.g., elephant flow)
  - Complex job structure (e.g., DAGs, co-flows)



No "one-size-fits-all" solution:
Best algorithm depends on specific workload and system



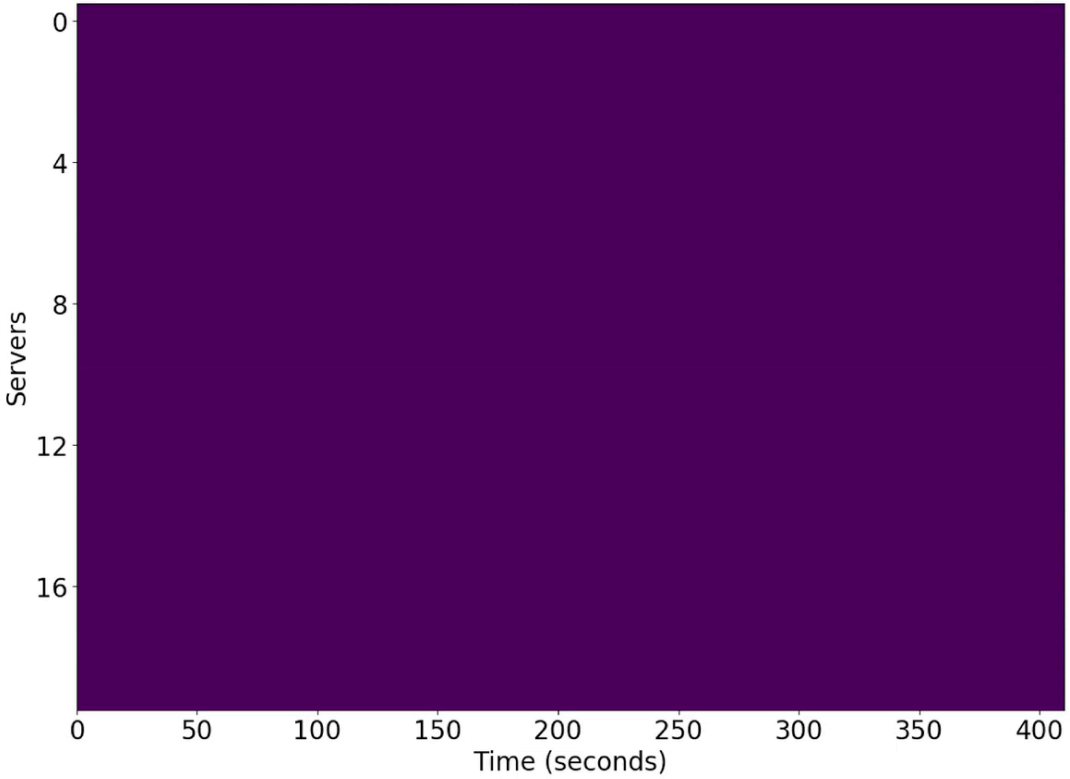
Commonly Used Heuristics

- First come, first served (FCFS)
- Last come, first served (LCFS)
- Earliest due date (EDD)
- Shortest processing time (SPT)
- Longest processing time (LPT)
- Fair sharing

Scheduling policy: FCFS

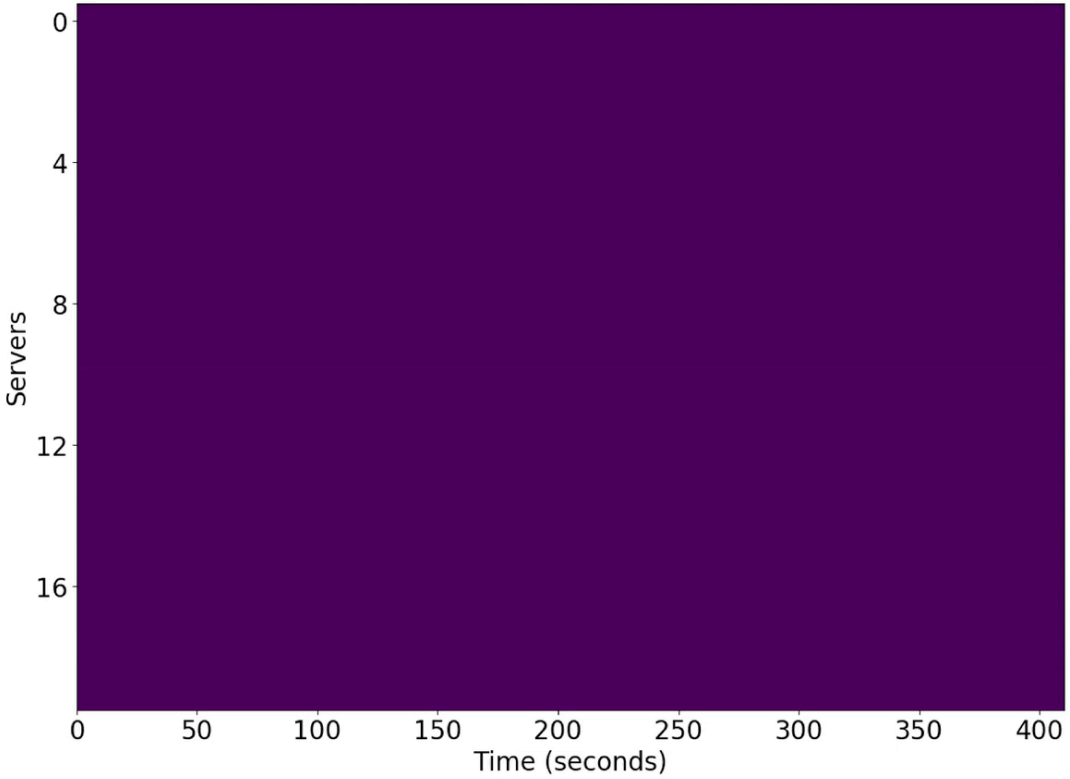
Demo

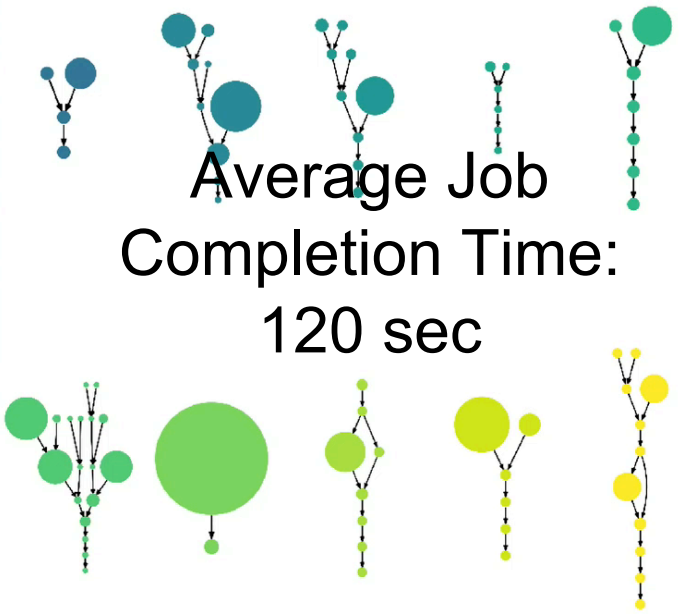
Scheduling Policy: Shortest-Job-First



Demo borrowed from Mao, Hongzi, et al. "Learning scheduling algorithms for data processing clusters." SIGCOMM'19

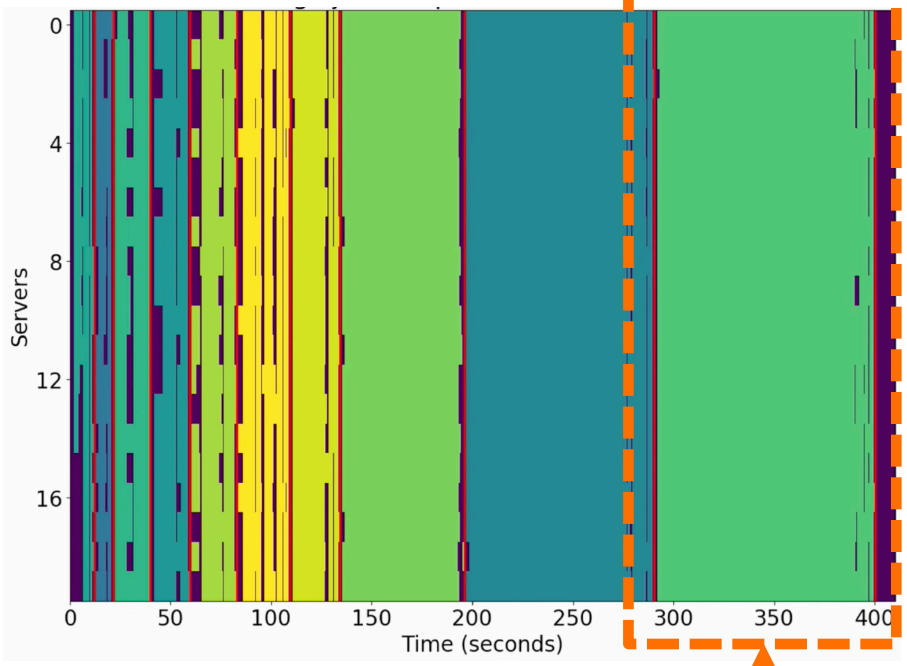
Scheduling Policy: Fair





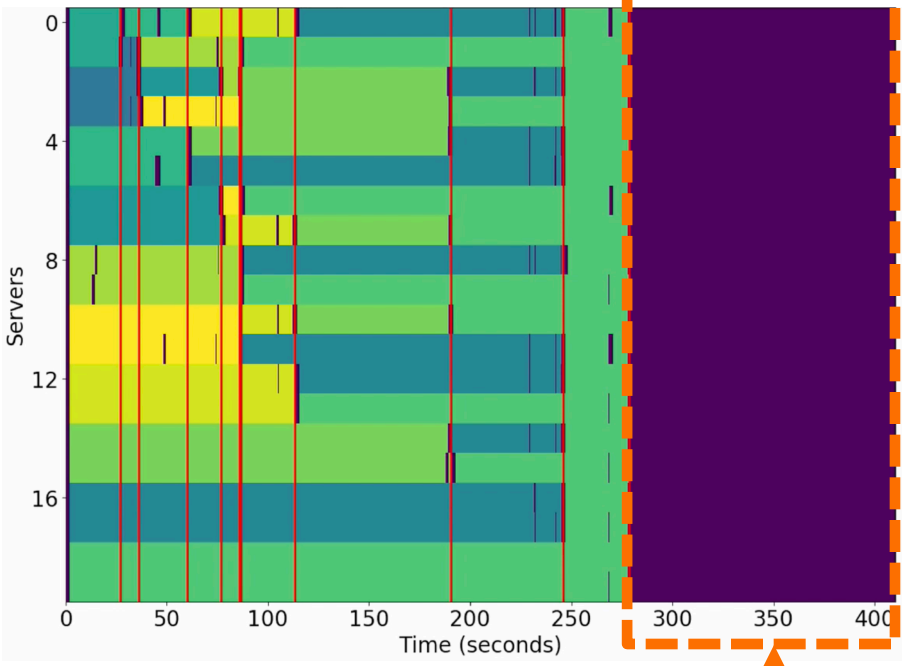
Demo borrowed from Mao, Hongzi, et al. "Learning scheduling algorithms for data processing clusters." SIGCOMM'19

Shortest-Job-First



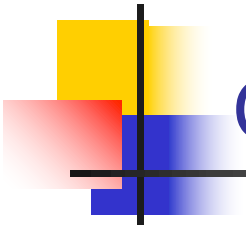
Average Job Completion Time:
135 sec

Fair



Average Job Completion Time:
120 sec

Demo borrowed from Mao, Hongzi, et al. "Learning scheduling algorithms for data processing clusters." SIGCOMM'19



Challenges in scheduling heterogeneous DAGs

- Simple heuristics lead to poor schedules
- Production DAGs are roughly 50% slower than lower bounds
- Simple variants of "Packing dependent tasks" are NP-hard problems
- Prior analytical solutions miss some practical concerns
  - Multiple resources
  - Complex dependencies
  - Machine-level fragmentation
  - Scale; Online; …

Can machine learning help tame the complexity of efficient schedulers for data centers?

A very recent work: Decima

Hongzi Mao, Malte Schwarzkopf, Shaileshh Bojja Venkatakrishnan, Zili Meng, Mohammad Alizadeh.

"Learning scheduling algorithms for data processing clusters." SIGCOMM'19

SIGCOMM is an annual 'flagship' conference organized by ACM's Special Interest Group on Data Communications, which specializes in the field of communication and computer network.

Learning Scheduling Algorithms for Data Processing Clusters

Hongzi Mao, Malte Schwarzkopf, Shaileshh Bojja Venkatakrishnan, Zili Meng*, Mohammad Alizadeh
MIT Computer Science and Artificial Intelligence Laboratory *Tsinghua University
{hongzi,malte,bjjvnkt,alizadeh}@csail.mit.edu,mengzl15@mails.tsinghua.edu.cn

Abstract

Efficiently scheduling data processing jobs on distributed compute clusters requires complex algorithms. Current systems use simple, generalized heuristics and ignore workload characteristics, since developing and tuning a scheduling policy for each workload is infeasible. In this paper, we show that modern machine learning techniques can generate highly-efficient policies automatically.

Decima uses reinforcement learning (RL) and neural networks to learn workload-specific scheduling algorithms without any human instruction beyond a high-level objective, such as minimizing average job completion time. However, off-the-shelf RL techniques cannot handle the complexity and scale of the scheduling problem. To build Decima, we had to develop new representations for jobs' dependency graphs, design scalable RL models, and invent RL training methods for dealing with continuous stochastic job arrivals.

Our prototype integration with Spark on a 25-node cluster shows that Decima improves average job completion time by at least 21% over hand-tuned scheduling heuristics, achieving up to 2× improvement during periods of high cluster load.

CCS Concepts: **Software and its engineering** → Scheduling; **Networks** → Network resources allocation; **Computing methodologies** → Reinforcement learning

Keywords: resource management, job scheduling, reinforcement learning

ACM Reference Format:
Hongzi Mao, Malte Schwarzkopf, Shaileshh Bojja Venkatakrishnan, Zili Meng and Mohammad Alizadeh. 2019. Learning Scheduling Algorithms for Data Processing Clusters. In *SIGCOMM '19, August 19-23, 2019, Beijing, China.* ACM, Beijing, China, 19 pages. https://doi.org/10.1145/3341302.3342080

1 Introduction

Efficient utilization of expensive compute clusters matters for enterprises: even small improvements in utilization can save millions of dollars at scale [11, §1.2]. Cluster schedulers are key to realizing these savings. A good scheduling policy packs work tightly to reduce fragmentation [34, 36, 76], prioritizes jobs according to high-level metrics such as user-perceived latency [77], and avoids inefficient configurations [28]. Current cluster schedulers rely on heuristics that prioritize generality, ease of understanding, and straightforward implementation over achieving the ideal performance on a specific workload. By using general heuristics like fair scheduling [8, 31], shortest-job-first, and simple packing strategies [34], current systems forego potential performance optimizations. For example, widely-used schedulers ignore readily available information about job structure (i.e., internal dependencies) and efficient parallelism for jobs' input sizes. Unfortunately, workload-specific scheduling policies that use this information require expert knowledge and significant effort to devise, implement, and validate. For many organizations, these skills are either unavailable, or uneconomic as the labor cost exceeds potential savings.

In this paper, we show that modern machine-learning techniques can help side-step this trade-off by *automatically learning* highly efficient, workload-specific scheduling policies. We present Decima[1], a general-purpose scheduling service for data processing jobs with dependent stages. Many systems encode job stages and their dependencies as directed acyclic graphs (DAGs) [10, 19, 42, 80]. Efficiently scheduling DAGs leads to hard algorithmic problems whose optimal solutions are intractable [36]. Given only a high-level goal (e.g., minimize average job completion time), Decima uses existing monitoring information and past workload logs to automatically learn sophisticated scheduling policies. For example, instead of a rigid fair sharing policy, Decima learns to give jobs different shares of resources to optimize overall performance, and it learns job-specific parallelism levels that avoid wasting resources on diminishing returns for jobs with little inherent parallelism. The right algorithms and thresholds for these policies are workload-dependent, and achieving them today requires painstaking manual scheduler customization.

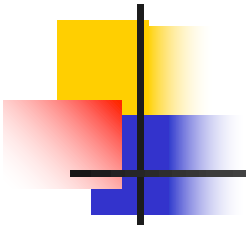
Decima learns scheduling policies through experience using modern reinforcement learning (RL) techniques. RL is well-suited to learning scheduling policies because it allows learning from actual workload and operating conditions without relying on inaccurate assumptions. Decima encodes its scheduling policy in a neural network trained via a large number of simulated experiments, during which it schedules a workload, observes the outcome, and gradually improves its policy. However, Decima's contribution goes beyond merely applying off-the-shelf RL algorithms to scheduling: to successfully learn high-quality scheduling policies, we had to develop novel data and scheduling action representations, and new RL training techniques.

First, cluster schedulers must scale to hundreds of jobs and thousands of machines, and must decide among potentially hundreds of configurations per job (e.g., different levels of parallelism). This leads to much larger problem sizes compared to conventional RL applications (e.g., game-playing [61, 70], robotics control [51, 67]), both in the amount of information available to the scheduler (the *state space*), and the number of possible choices it must consider (the *action space*).[2] We designed a scalable neural network architecture that combines a *graph neural network* [12, 23, 24, 46] to process job and cluster information without manual feature engineering, and a *policy*

SIGCOMM '19, August 19-23, 2019, Beijing, China

ACM ISBN 978-1-4503-5956-6/19/08…$15.00
https://doi.org/10.1145/3341302.3342080

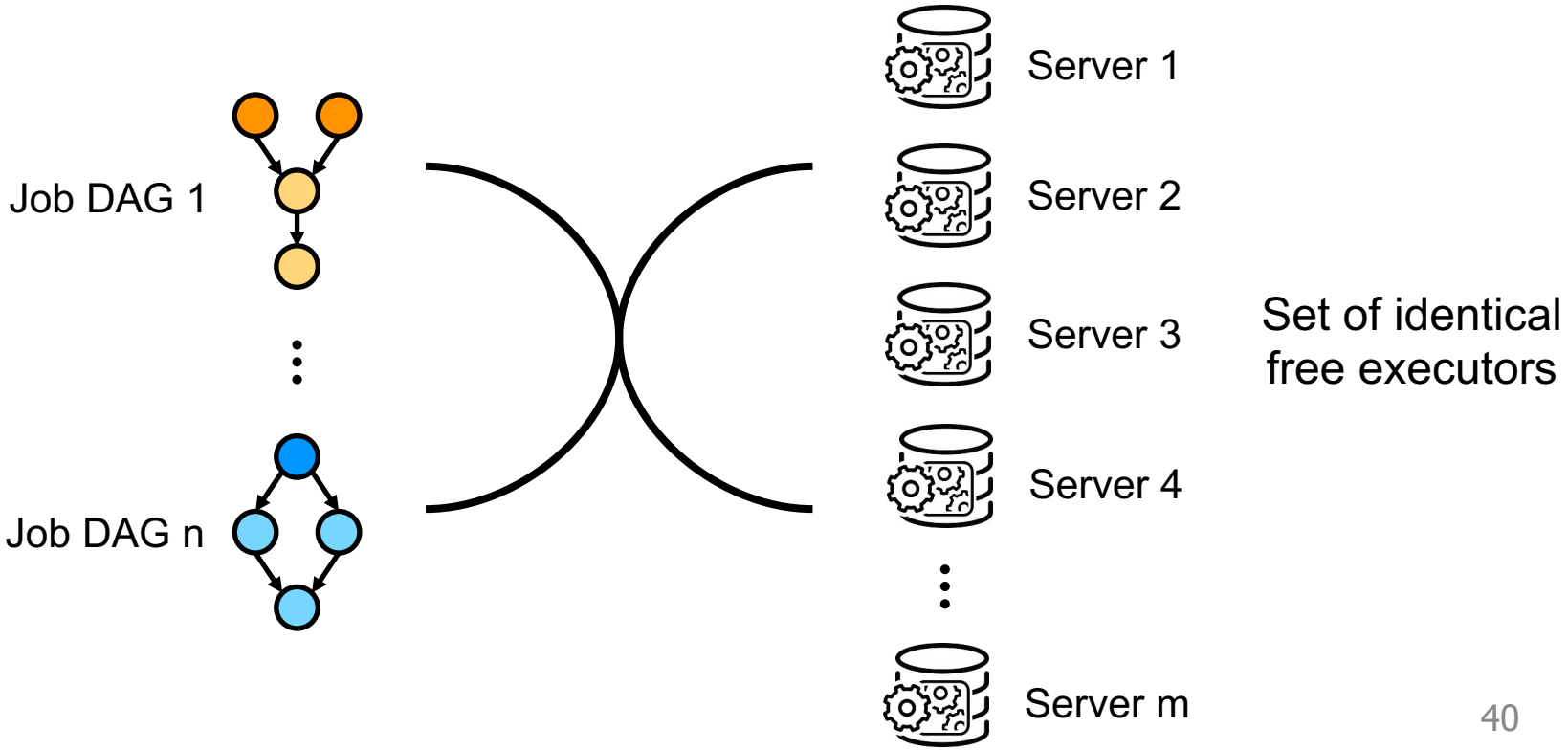
[1] In Roman mythology, Decima measures threads of life and decides their destinies.
[2] For example, the state of the game of Go [71] can be represented by 19 × 19 = 361 numbers, which also bound the number of legal moves per turn.



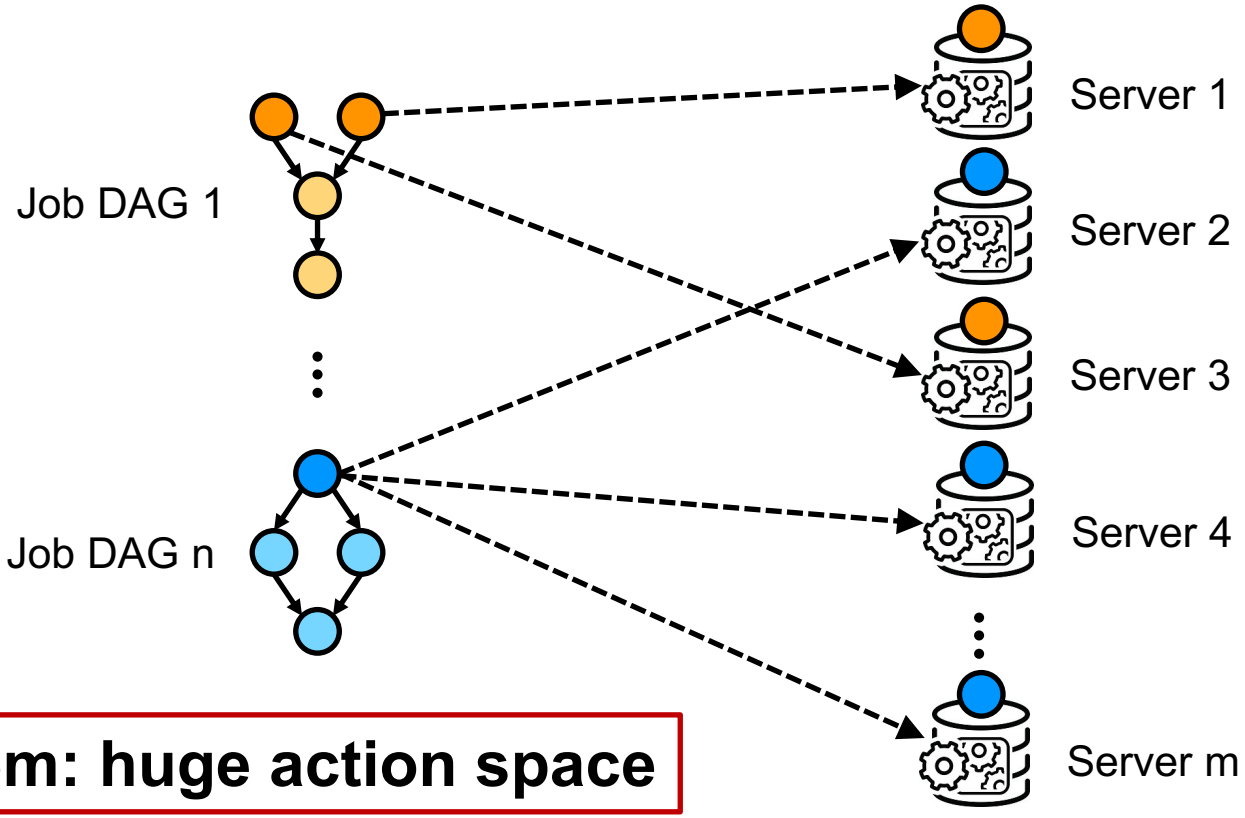
Decima: Overview and Design

Encoding scheduling decisions

- Scheduling is a mapping between runnable node and available servers.
  - Potentially thousands of runnable nodes and available servers cause an exponentially large set of mappings



Option 1: Assign All Servers in 1 Action

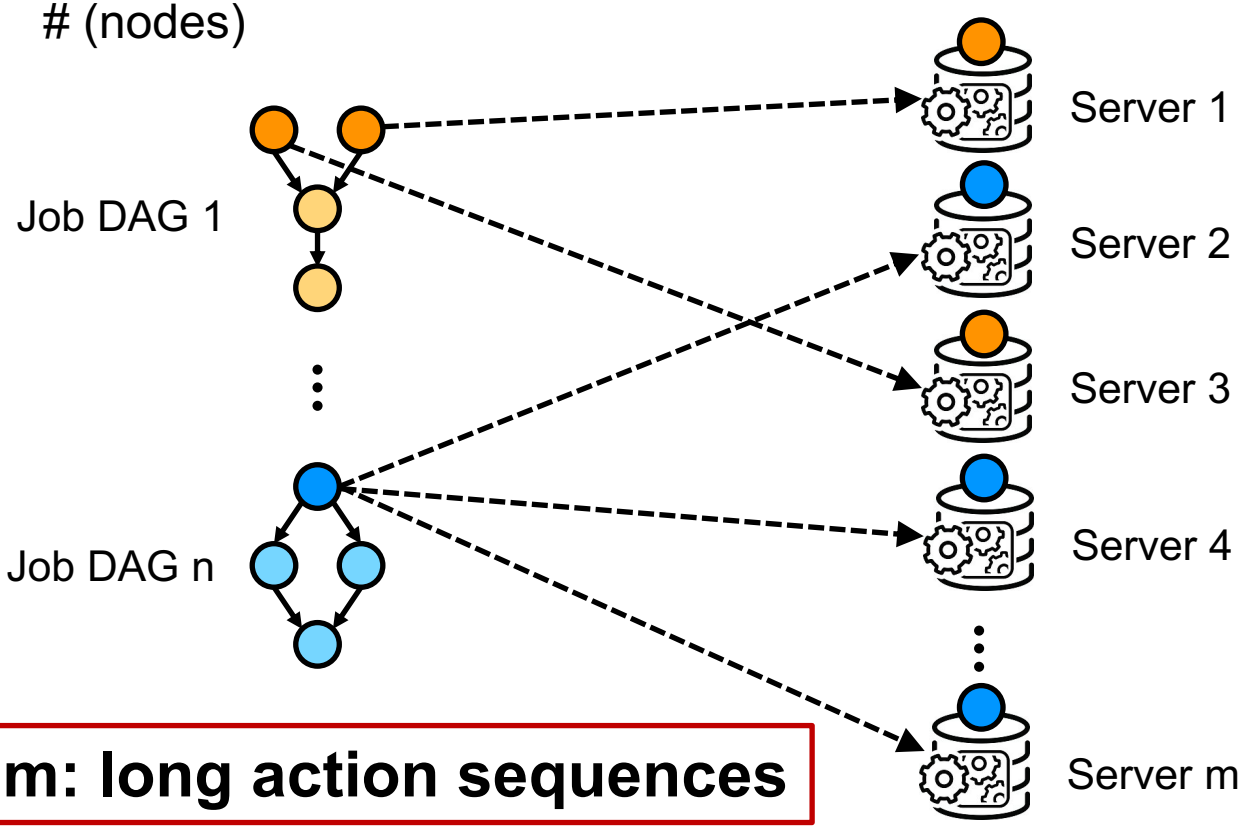


Problem: huge action space

action space has to contain every single possible mapping

Option 2: Assign one Servers Per Action

Size of action space:
(nodes)

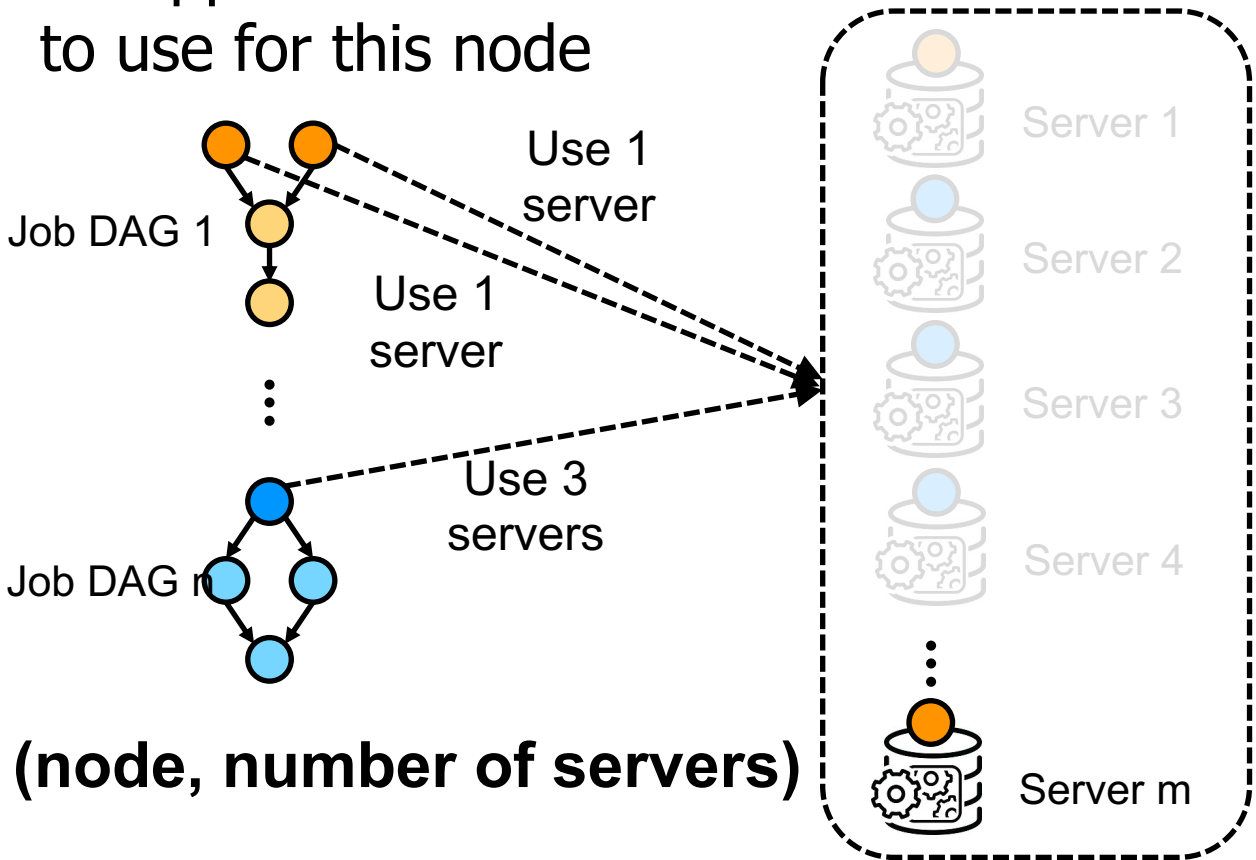


Problem: long action sequences

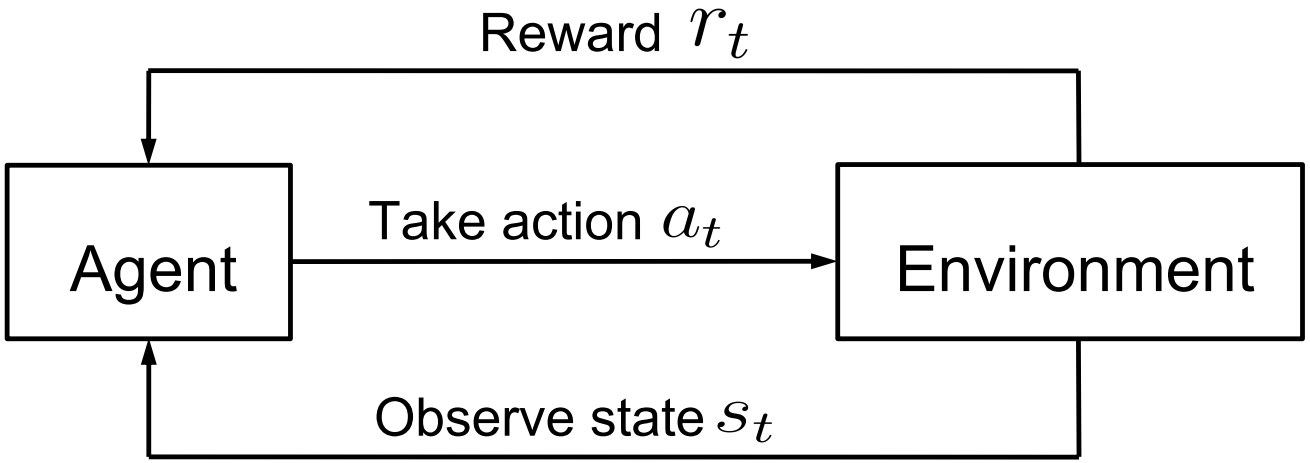
as many actions as the number of free servers

Decima: Assign Groups of Servers per Action

- Decima balances the size of the action space and the length of action sequence, which outputs:
  - A node selected to be scheduled next
  - An upper bound limit on the number of servers assigned to use for this node



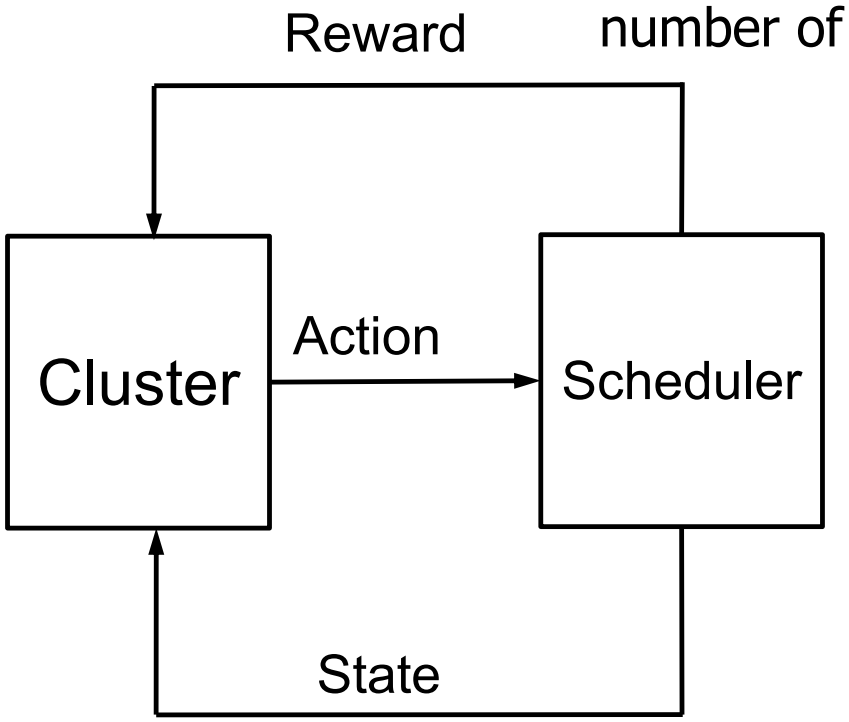
Action = (node, number of servers)



Goal: maximize the cumulative reward $\sum_t r_t$

Decima Design Details

Action:
(next schedule node, number of servers)



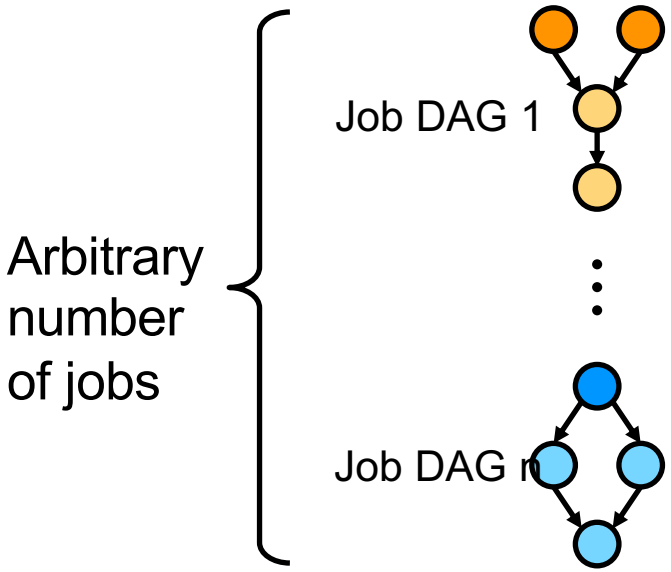
Two types of state
Job DAG:
- # of stages
- # of tasks
- avg. task duration
- # of servers currently assigned to the job
- dependency structure

Server status:
- # of available servers

Objective:
minimize average *job completion time*

Reward:
A penalty for every unfinished job

Process Job Information



Node features:

- # of tasks
- avg. task duration
- # of servers currently assigned to the node
- are free servers local to this job?

Data dependency between nodes

Arbitrary number of nodes

Graph Neural Network (1)

- Neighborhood Aggregation
  - Nodes aggregate information from their neighbors using neural network
- Assume we have a DAG G_i:
  - v_i is the node
  - $\mathbf{x}_v^i$ is the node feature vector
- Node embedding

$$(G_i, \mathbf{x}_v^i) \longmapsto \mathbf{e}_v^i$$

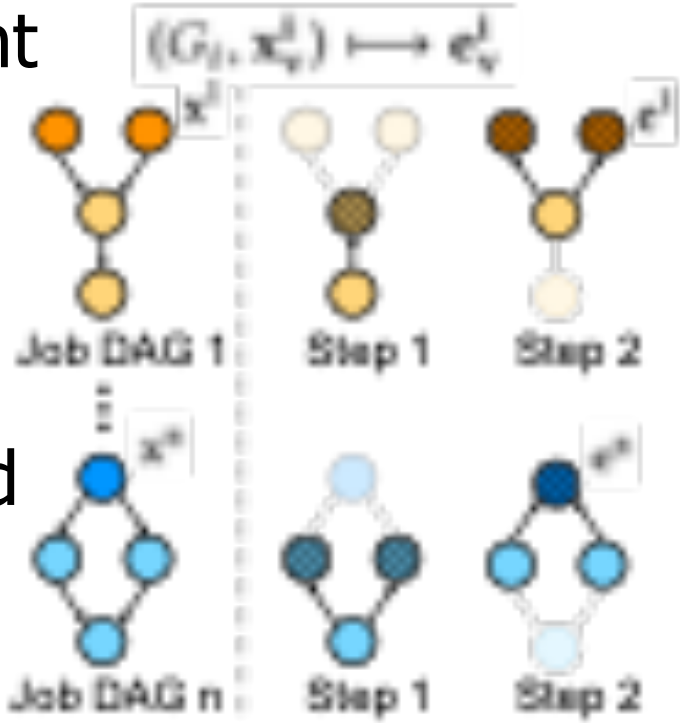
  - is a vector that captures information from all nodes reachable from v

Graph Neural Network (2)

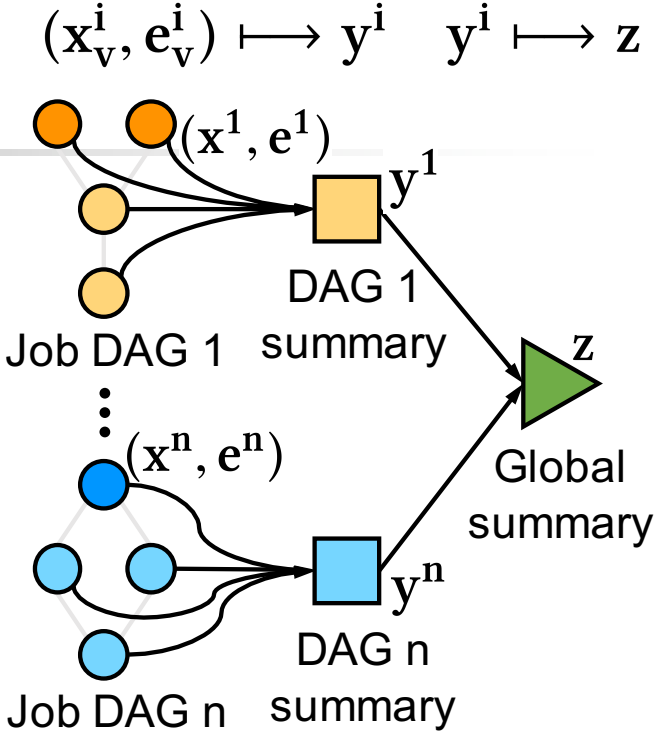
- Decima Decima propagates information from children to parent nodes in a sequence of message passing steps
  - starting from the leaves of the DAG
- In each message passing step, node embedding can be computed as:

$$\mathbf{e}_v^i = g\left[\sum_{u \in \xi(v)} f(\mathbf{e}_u^i)\right] + \mathbf{x}_v^i$$

  - f and g are non-linear transformation
  - $\xi(v)$ is the set of v's children



Graph Neural Network (3)



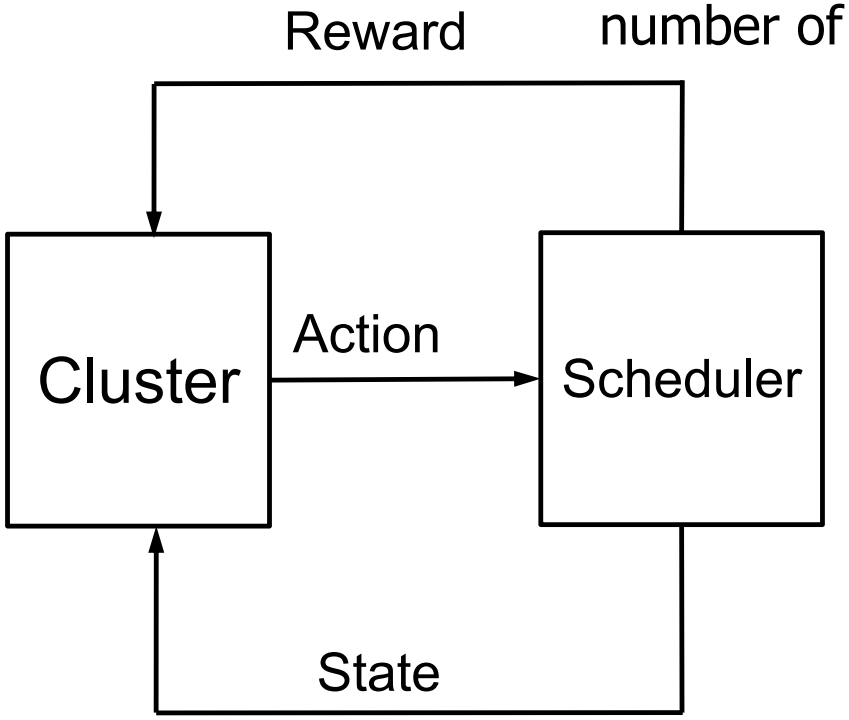
- Per-job Embedding
  - a summary of all node embeddings for each DAG
  - Decima adds a summary node to each DAG, which has all the nodes in the DAG as children
- Global Embedding
  - a global summary across all DAG
  - DAG-level summary nodes are in turn children of a single global summary node
- Per-job and Global Embedding computed as:

$$\mathbf{e}_v^i = g\left[\sum_{u \in \xi(v)} f(\mathbf{e}_u^i)\right] + \mathbf{x}_v^i$$

  - f and g are different for different level embeddings

Decima Design Details

Action:
(next schedule node, number of servers)



Two types of state
Job DAG:
- # of stages
- # of tasks
- avg. task duration
- # of servers currently assigned to the job
- dependency structure

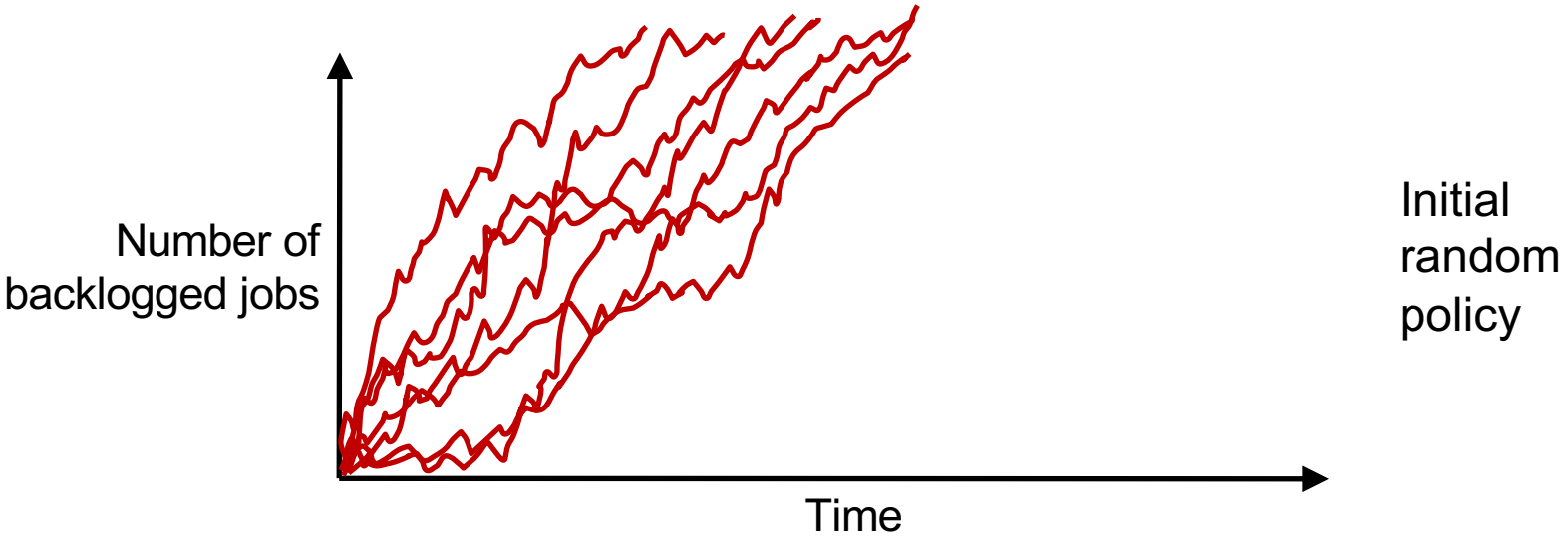
Server status:
- # of available servers

Objective:
minimize average *job completion time*

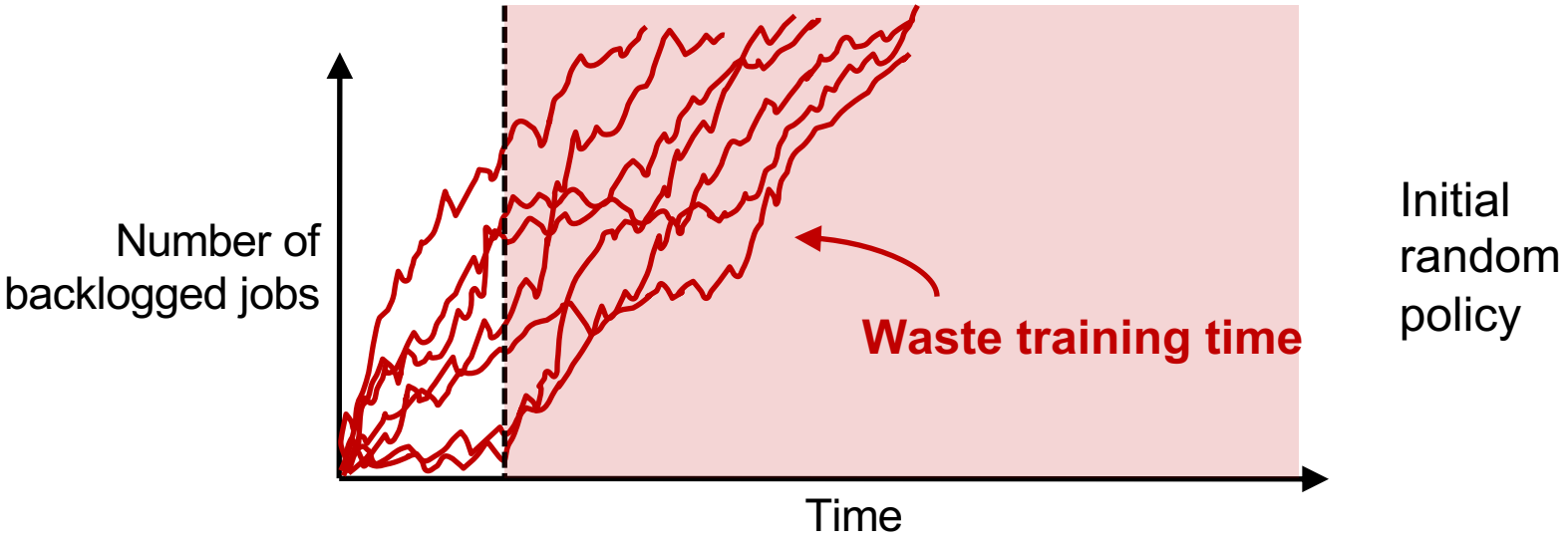
Reward:
A penalty for average # of jobs

Challenge: Handle Online Job Arrival

- The RL agent has to experience continuous job arrival during training.
- → inefficient if simply feeding long sequences of jobs



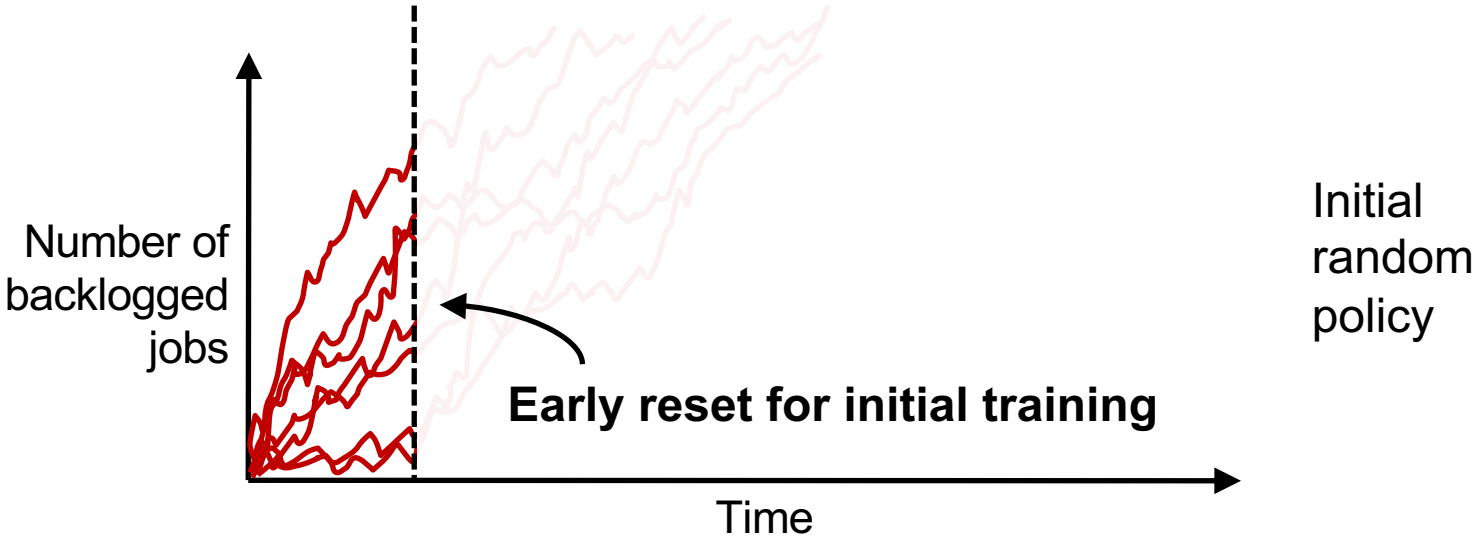
Challenge: Handle Online Job Arrival



Challenge: Handle Online Job Arrival

The RL agent has to experience **continuous job arrival** during **trainin**

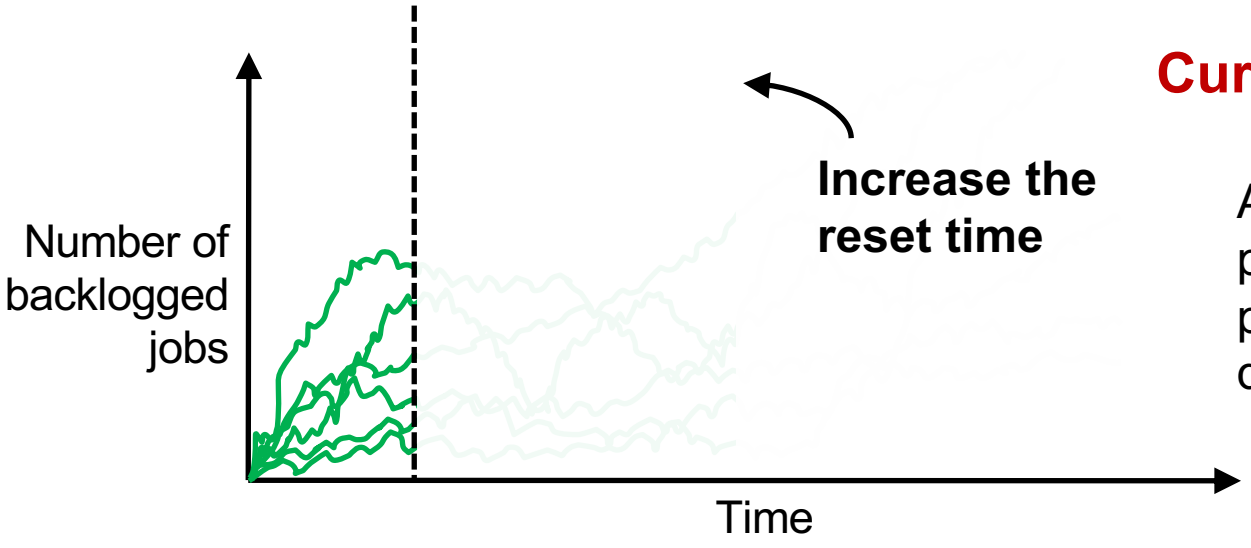
→ inefficient if simply feeding long sequences of jobs



Challenge: Handle Online Job Arrival

The RL agent has to experience **continuous job arrival** during **trainin**

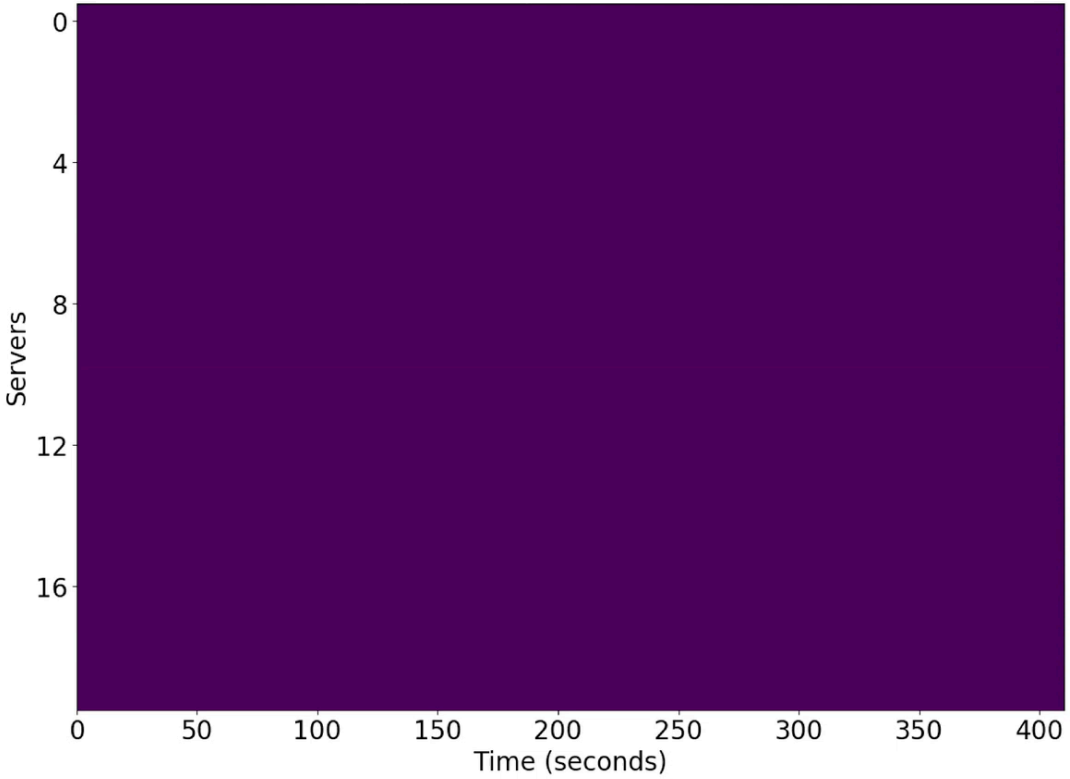
→ inefficient if simply feeding long sequences of jobs

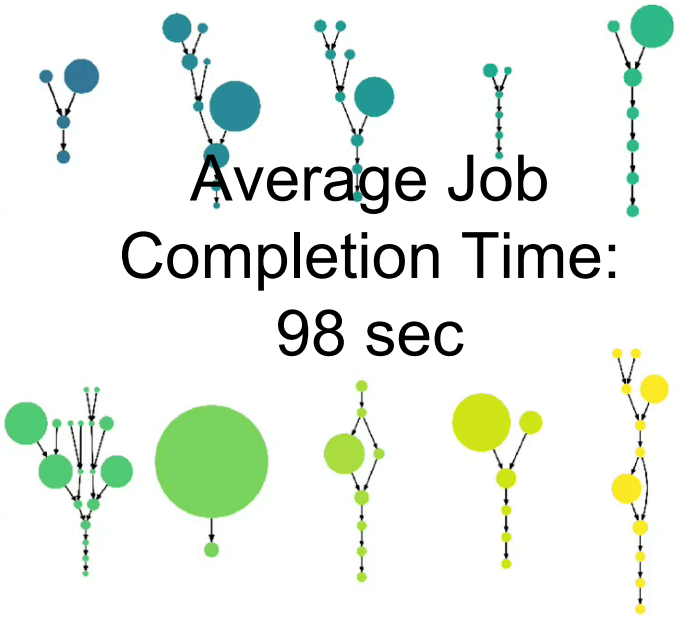


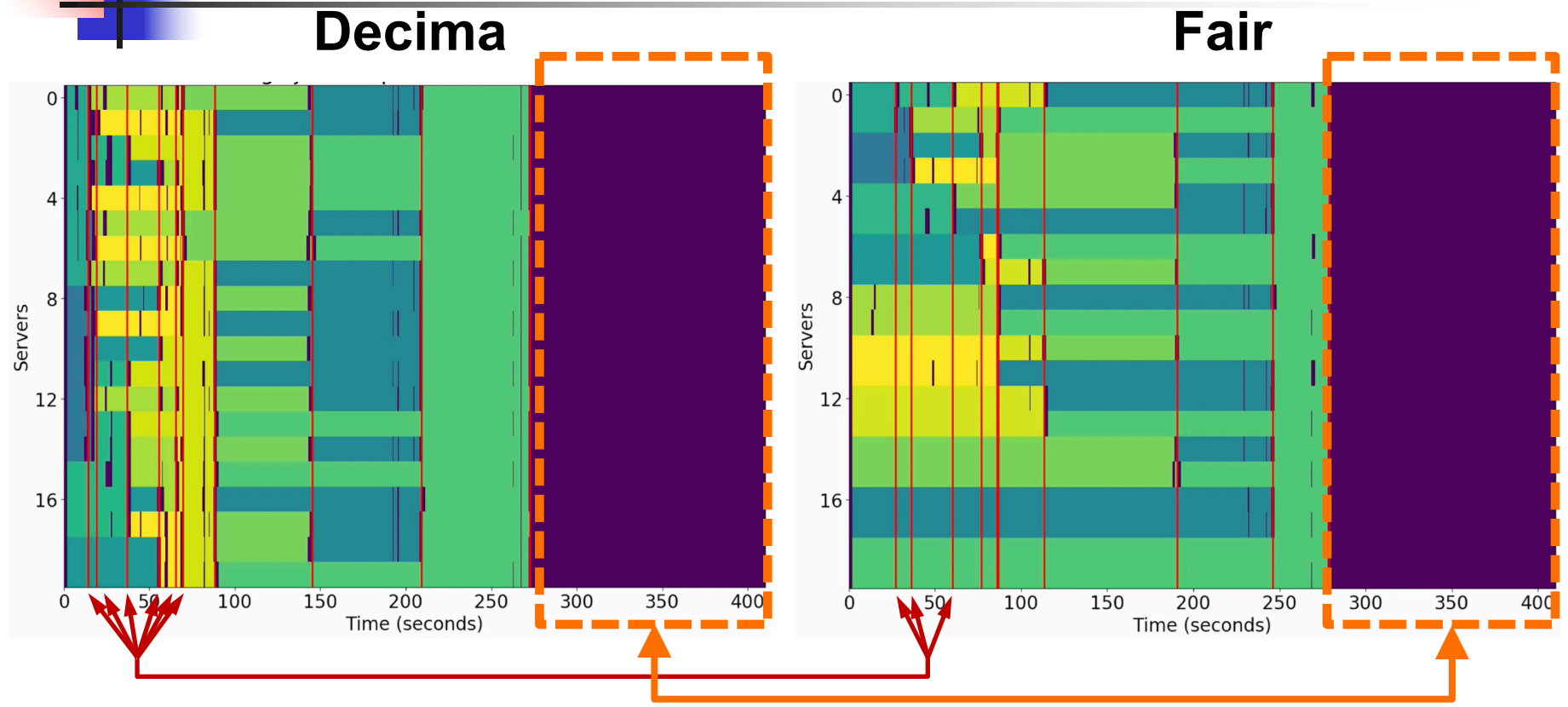
Curriculum learning

As training proceeds, stronger policy keeps the queue stable

Scheduling Policy: Decima





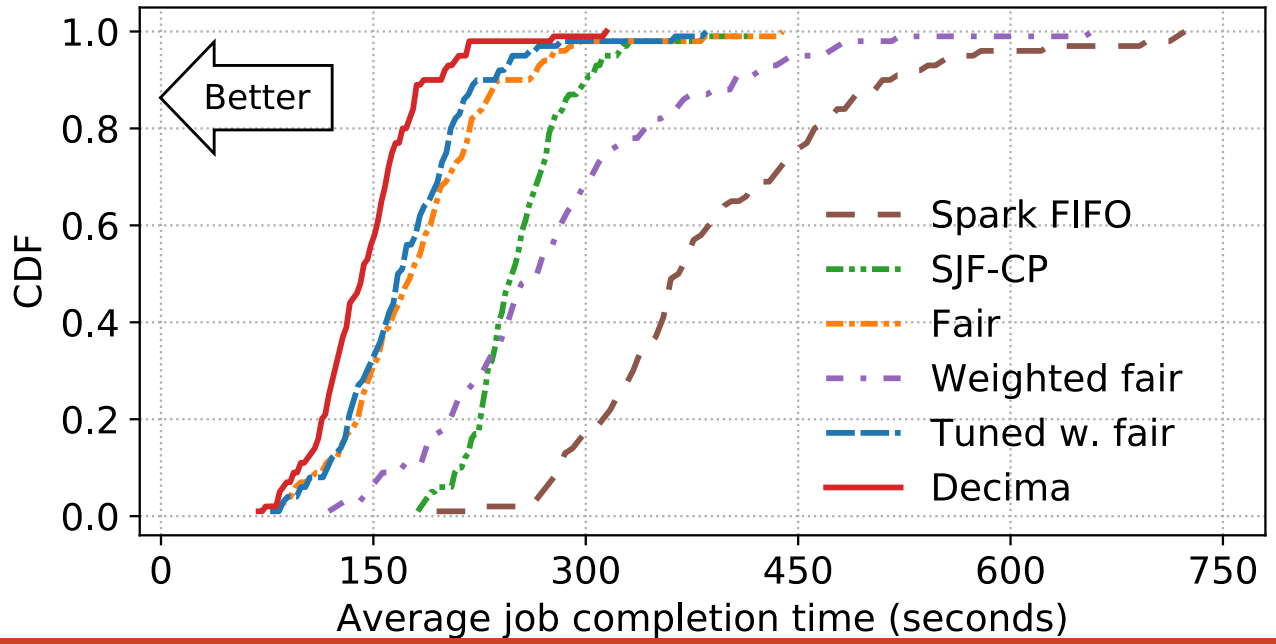


Average Job Completion Time: 98 sec

Average Job Completion Time: 120 sec

Decima vs. Baselines: Batched Arrivals

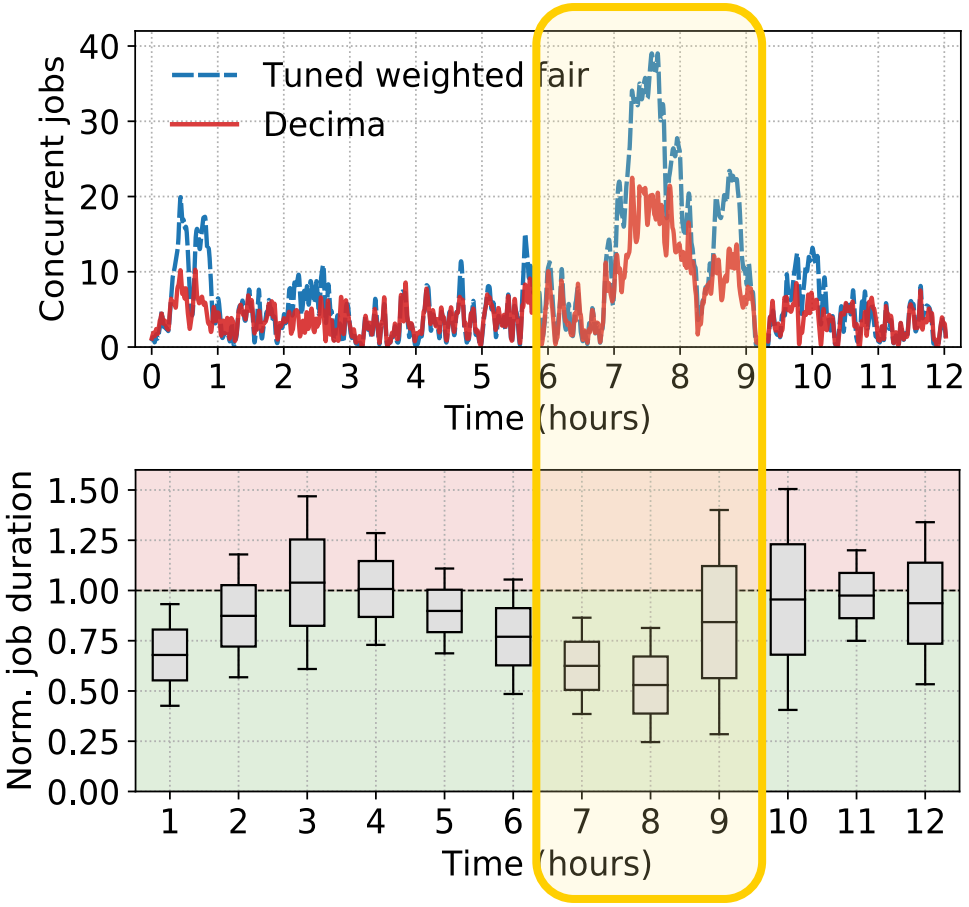
- 20 TPC-H queries sampled at random; input sizes: 2, 5, 10, 20, 50, 100 GB
- Decima trained on simulator; tested on real Spark cluster

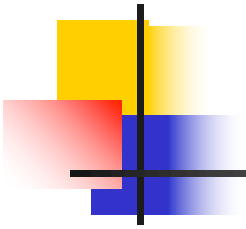


Decima improves average job completion time by 21%-3.1x over baseline schemes

Decima with Continuous Job Arrivals

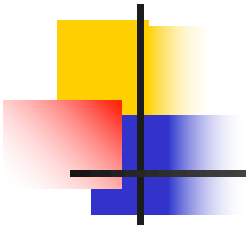
- 1000 jobs arrives as a Poisson process with avg. inter-arrival time = 25 sec
- Decima achieves 28% lower average JCT than best heuristic, and 2X better JCT in overload





Outline

- Introduction: Why We Need Intelligent Network?
- What Networking Problems Can ML help?
  - Traffic Classification
  - Congestion Control
  - Resource Management
  - …
- Case Study: Cluster Scheduling
  - Decima, Sigcomm 2018
- **Future: challenges and opportunities for MLN**

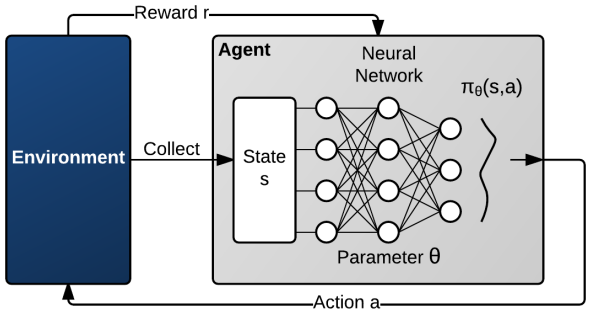


Challenges Faced by MLN

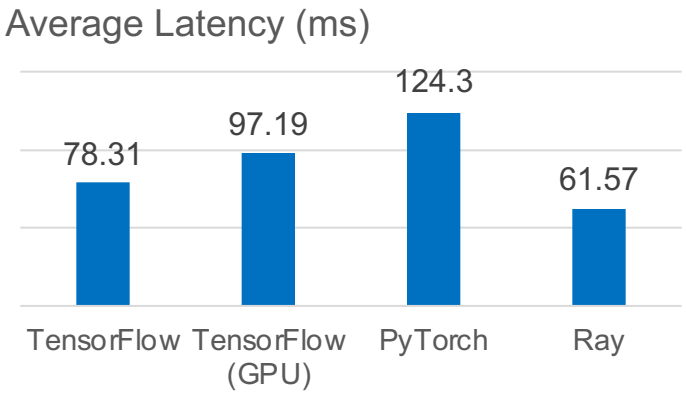
- **BIG Challenge: feasibility**
  - Most networking applications are delay-sensitive
- Lack of open labeled data
  - Acquire one's own data is expensive vs. limited open dataset
  - Learning with a simulator sometimes can be more effective in RL scenarios
- High dynamic network environments
  - Networks are dynamic in an unexpected and previously unobserved way.
  - Need to constantly retrain the ML model.
- High cost brought by learning errors
  - i.e., network security

A Toy Example: Deep RL for Flow Scheduling

- Flow Scheduling: Ordering flows using priorities
- Experiments from AuTO* find out:
  - Simplest neural network to model $\pi_\theta(a_t|s_t)$: only one hidden layer.
  - Implemented on popular deep learning frameworks
  - Hardware: 4-core Intel E5-1410 2.8GHz CPU, NVIDIA K40 GPU, and Broadcom 1Gbps NICs
  - 1000 flows per second.



* Li Chen, et al. "Scaling Deep Reinforcement Learning to Enable Datacenter-Scale Automatic Traffic Optimization" *ACM SIGCOMM computer communication review*. Vol. 40. No. 4. ACM, 2010.

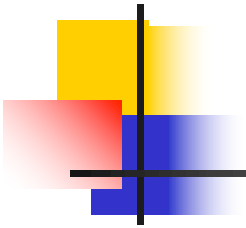


- The processing delays are more than 60ms.
  - Any flow within 7.5MB would've finished on a 1Gbps link.
  - A 7.5MB flow is larger than 95.13% of all flows in production data centers*.

* Alizadeh, Mohammad, et al. "Data center tcp (dctcp)." *ACM SIGCOMM computer communication review*. Vol. 40. No. 4. ACM, 2010.

Too Slow!

Most of the DRL actions are useless: Short flows are already gone when the actions arrive.



Challenges Faced by MLN

- BIG Challenge: feasibility
  - Most networking applications are delay-sensitive
- Lack of open labeled data
  - Acquire one's own data is expensive vs. limited open dataset
  - Learning with a simulator sometimes can be more effective in RL scenarios
- High dynamic network environments
  - Networks are dynamic in an unexpected and previously unobserved way.
  - Need to constantly retrain the ML model.
- High cost brought by learning errors
  - i.e., network security

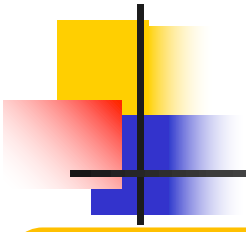
Lessons Learned: Discipline vs. Tool

ML for Networking IS NOT about trying different algorithms to obtain better results. To build a solid house on your own, you need to know about architecture, as well as about the characteristics of the construction toolbox…

Two commonly arising problems when coupling ML and Networking:

You have to understand the PROBLEM:

- Even a ML expert fails to achieve a good networking solution if he neither knows the good descriptors nor understands the problem (e.g., try to classify flows using only port numbers).
- Keep the scope narrow, to better understand the overall process (i.e., from selecting features to evaluation and conclusions).
- The solution must be meaningful in practical terms (e.g., flow deadline)



Lessons Learned: Discipline vs Tool

ML for Networking IS NOT about trying different algorithms to obtain better results. To build a solid house on your own, you need to know about architecture, as well as about the characteristics of the construction toolbox…

Two commonly arising problems when coupling ML and Networking:

You have to understand the TOOL:

- The broader overview you have about the particularities of each ML approach, the better chances to apply the correct one (e.g., avoid killing mosquitos with a hammer)
- The research community does not benefit any further from yet another untried ML approach (e.g., IDS based on KDD'99 dataset)
- A good grasp of calculus, linear algebra, and probability is essential for a clear understanding of ML for Networking

Thanks for Your Attention !!
Remarks & Questions ?