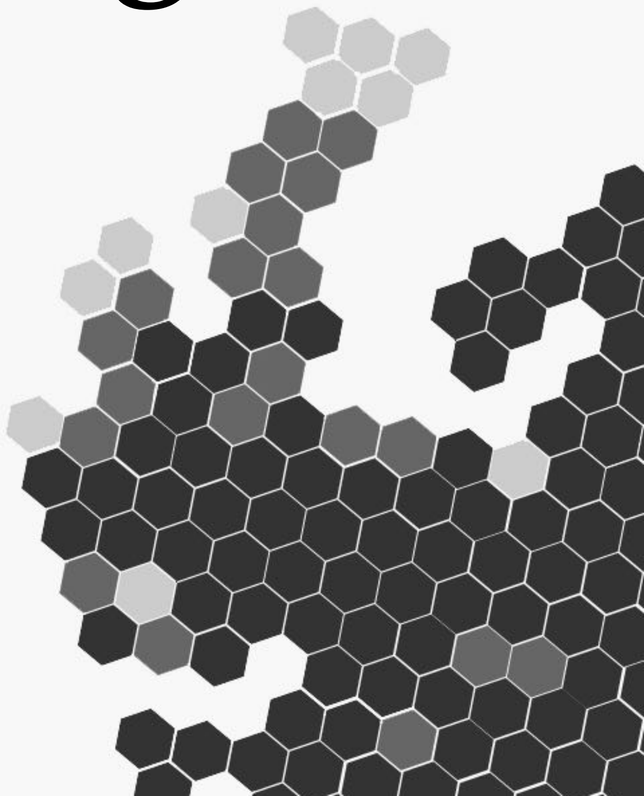
Cloud Management with Reinforcement Learning

Qizhen WENG
Supervisor: Prof. Wei WANG

CSE HKUST
PhD Qualifying Exam
8th May 2019



Introduction - Cloud Management

- **Challenges for Cloud Service Providers:**
  - Operating *large-scale*, *heterogeneous*, *commodity* cluster with low cost
  - Dynamically allocating *physical* and *virtual* resources among consumers
  - Guaranteeing Quality of Services (QoS) (e.g., EMR job latency, S3 throughput)
  - …
- **Require expertise in:**
  - Networking
  - System
  - Database
  - ……

Image source: Google Datacenter, https://www.google.com/about/datacenters/

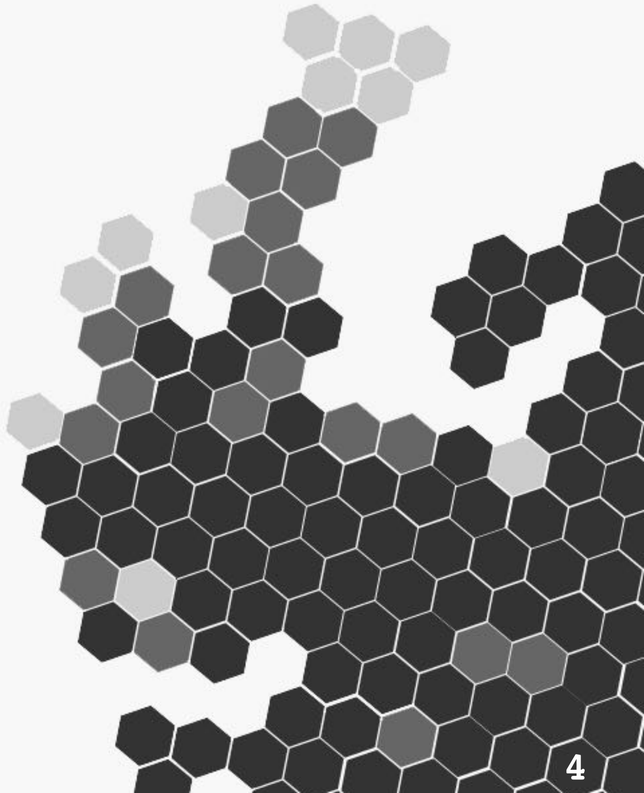
Introduction - Reinforcement Learning for Systems

- **Apply RL techniques to improve computing systems:**
  - Automate data collection, analysis, policy design, testing and deployment
  - Using trial-and-error in repetitive environment towards better performance
  - Adaptive to continuous changing of workloads
- **Challenges:**
  - Tailoring problem formulations, state representations, reward structures, etc.
  - Curse of dimensionality: at the scale of clusters.
  - Instability and data-hunger of RL with deep learning techniques.

Image source: Nature Volume 529, Issue 7587, 28 Jan. 2016, https://www.nature.com/nature/volumes/529/issues/7587

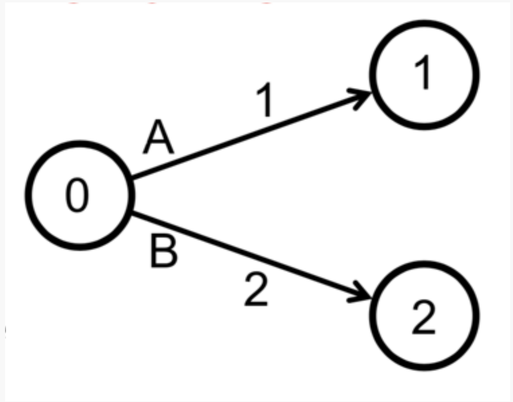
Outline

- Introduction to Reinforcement Learning (RL)
  - Markov Decision Process (MDP)
  - Reinforcement Learning Basics
  - Value-based Method
  - Policy-based Method
  - Deep Reinforcement Learning
- RL Applications in Cloud Management



Markov Decision Process (MDP)

State: `0, 1, 2;`
Action: `A, B;`
Transition: Pr(1|0,A)=1, Pr(2|0,B)=1
Reward: `(0,A,1):1, (0,B,2):2`



Markovian Property*:* Transitions & Rewards only based on *current* state and action.

Formally, an MDP is

- A set of ***states***, $\mathcal{S} = \{S_1, S_2, \cdots, S_T\}$
- A set of ***actions***, $\mathcal{A} = \{A_1, A_2, \cdots, A_T\}$
- A ***transition function***, $\mathcal{P}^a_{ss'} \doteq \Pr[S_{t+1} = s' | S_t = s, A_t = a]$ $\quad \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to [0, 1]$
- A ***reward function***, $\mathcal{R}^a_{ss'} \doteq \mathbb{E}[R_{t+1} | S_t = s, A_t = a, S_{t+1} = s']$ $\quad \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}$

image source: Smart, B. D. (2005). Reinforcement learning: A user's guide. In Tutorial at International Conference on Autonomic Computing (ICAC 2005).

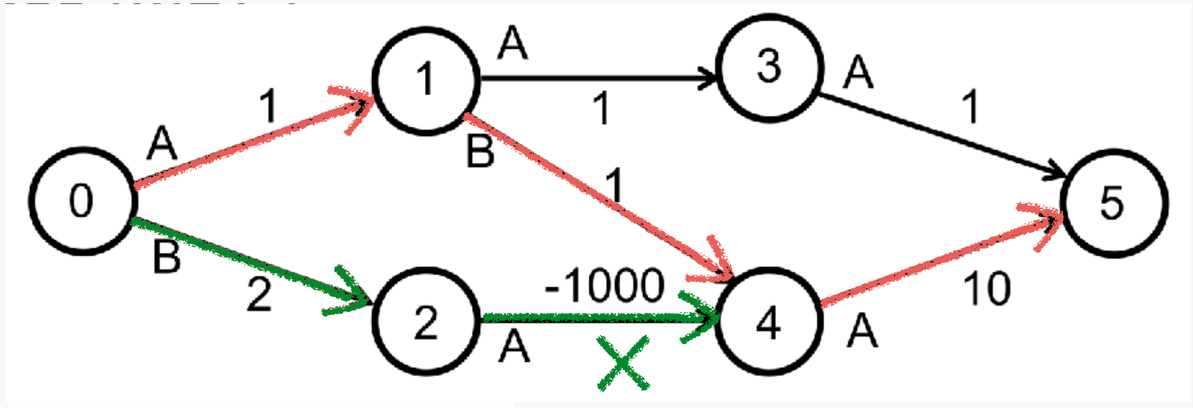
Reinforcement Learning Basics

Policy: $\pi(a|s) \doteq \Pr[A_t = a | S_t = s]$

Pr(B | 0)=1, Pr(A | 0)=0

Optimal Policy $\pi_*(a|s)$: the *goal* of *RL*

Pr(A | 0)=1, Pr(B | 1)=1, Pr(A | 4)=1



maximize the **return** (sum of reward): with discount factor $\gamma \in [0, 1]$

$$G_t \doteq R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1} = R_{t+1} + \gamma G_{t+1}$$

The **state-value function:** expected return from state s following policy π

$$v_\pi(s) \doteq \mathbb{E}_\pi[G_t | S_t = s]$$

The **action-value function:** expected return from s, taking action a, then following π

$$q_\pi(s, a) \doteq \mathbb{E}_\pi[G_t | S_t = s, A_t = a]$$

image source: Smart, B. D. (2005). Reinforcement learning: A user's guide. In Tutorial at International Conference on Autonomic Computing (ICAC 2005).

Value-based Method: DP

An optimal policy can be derived by maximizing over optimal value functions (greedy)

$$\pi_*(a|s) = \begin{cases} 1 & \text{if } a = \arg\max_{a \in \mathcal{A}} q_*(s,a) \\ 0 & \text{otherwise} \end{cases}$$

where $q_*(s,a) = \max_\pi q_\pi(s,a)$

Dynamic Programming (DP): **model-based** policy evaluating. (s / s': current/next state)

$$q_*(s,a) = \sum_{s' \in \mathcal{S}} [\mathcal{R}^a_{ss'} + \gamma \mathcal{P}^a_{ss'} \max_{a'} q_*(s', a')]$$

$$v_*(s) = \max_a \left\{ \sum_{s' \in \mathcal{S}} [\mathcal{R}^a_{ss'} + \gamma \mathcal{P}^a_{ss'} v_*(s')] \right\}$$

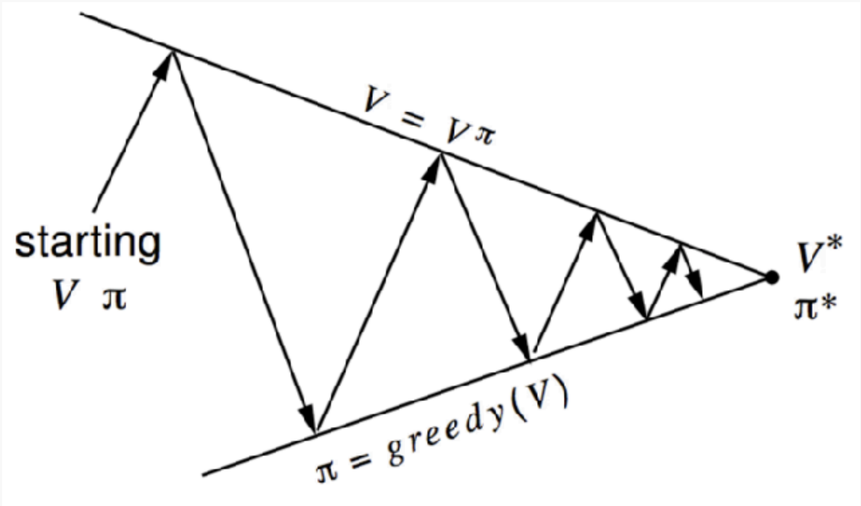


image source: Smart, B. D. (2005). Reinforcement learning: A user's guide. In Tutorial at International Conference on Autonomic Computing (ICAC 2005).

Value-based Method: MC

- Monte Carlo (MC) method: **model-free** policy evaluation
  - Learn directly from *complete* episodes of *experience*
  - *Model-free*: *unknown* of transition/reward function
  - Use *empirical mean* to *approximate* the expected return
  - $V(S)$, estimating $v(S)$, is unbiased but with high variance

$$V(S_t) \leftarrow V(S_t) + \alpha[G_t - V(S_t)]$$

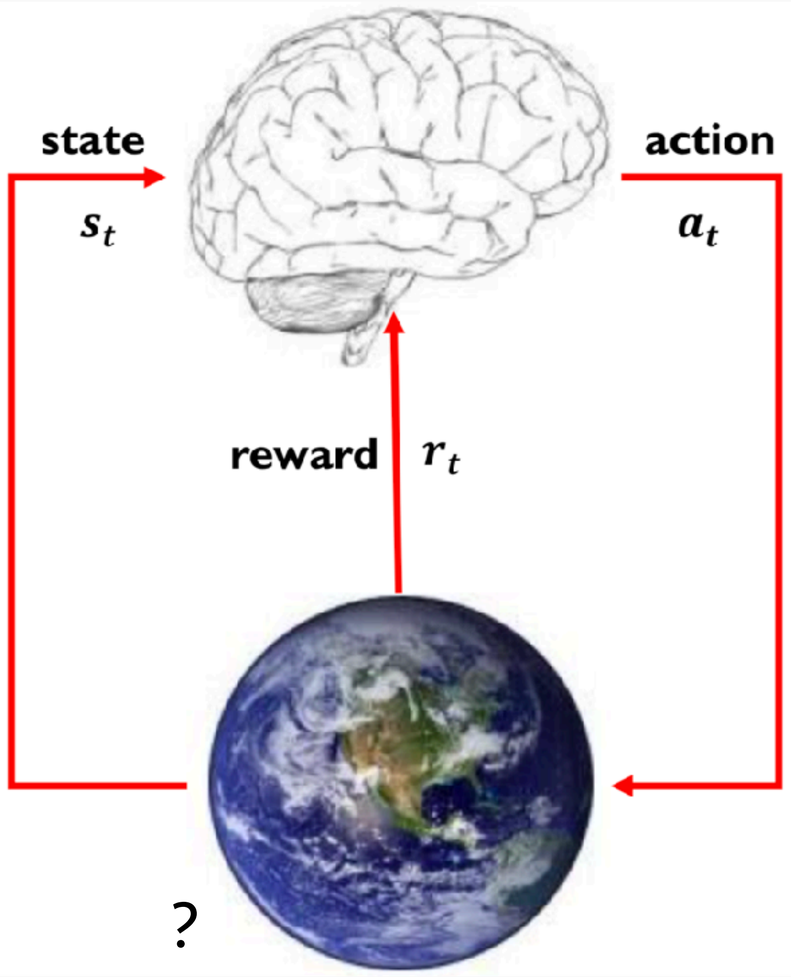


Image source: Silver, D. (2015). UCL Course on RL. David Silver's Webpage, http://www0.cs.ucl.ac.uk/staff/d.silver

Value-based Method: TD

- Temporal Difference (TD) learning: **model-free** policy evaluation
  - Can learn from *incomplete* episodes
  - *Bootstrapping*: update value estimates on the basis of other value estimates
  - (Compared to MC) Lower variance, more bias; more efficient, worse convergence.

TD(0) $$V(S_t) \leftarrow V(S_t) + \alpha[\overbrace{R_{t+1} + \gamma V(S_{t+1})}^{\text{MC: } G_t} - V(S_t)]$$

SARSA $$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha[R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t)]$$

Q-learning $$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha[R_{t+1} + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t)]$$

Policy-based Method: Policy Gradient

- Greedy policy improvement by performing *argmax Q(s,a)* is computationally demanding when the action space is large or continuous.
- Instead, policy-based methods represent the policy directly with parameter θ

$$\pi_\theta(s, a) = \Pr[a|s; \theta]$$

- And perform *policy gradient* to improve policy with performance measurement, which is also expressed as a loss function of θ: *J(*θ*)*. *(e.g., the negative of return)*

$$\theta_{t+1} = \theta_t + \alpha_t \nabla_\theta J(\theta_t) \qquad \nabla_\theta J(\theta) = \frac{\partial J}{\partial \pi_\theta} \frac{\partial \pi_\theta}{\partial \theta}$$

Policy-based Method: Representative Alg.

- ***REINFORCE***: Monte Carlo Policy Gradient

$$\theta_{t+1} = \theta_t + \alpha_t \gamma^t G_t \nabla_\theta \ln \pi(A_t|S_t;\theta)$$

REINFORCE with baseline: $\theta_{t+1} = \theta_t + \alpha_t \gamma^t (G_t - \underbrace{b(S_t)}_{\text{baseline}}) \nabla_\theta \ln \pi(A_t|S_t;\theta)$

reduces variance and helps a lot in practice

- ***Actor-Critic***: combines policy-based (actor) and value-based (critic) methods.
  - Actor: generates actions by policy networks and interacts with the environment.
  - Critic: supplies the actor with low-variance policy gradient estimates.
  - Critic: updates the value function parameters.
  - Actor: updates the policy parameters with policy gradient suggested by the actor.

Deep Reinforcement Learning (DRL)

- **Deep Neural Networks as Function Approximators**
  - Represent v*alue functions* or *policy functions*.
  - Optimize loss function by stochastic gradient descent (SGD).

- ***Deep Q-Networks*** [Mnih et al. *Nature'15*]: extend Q-learning with DNN
  - overcoming instability issues with *experience replay* and *periodically update*.

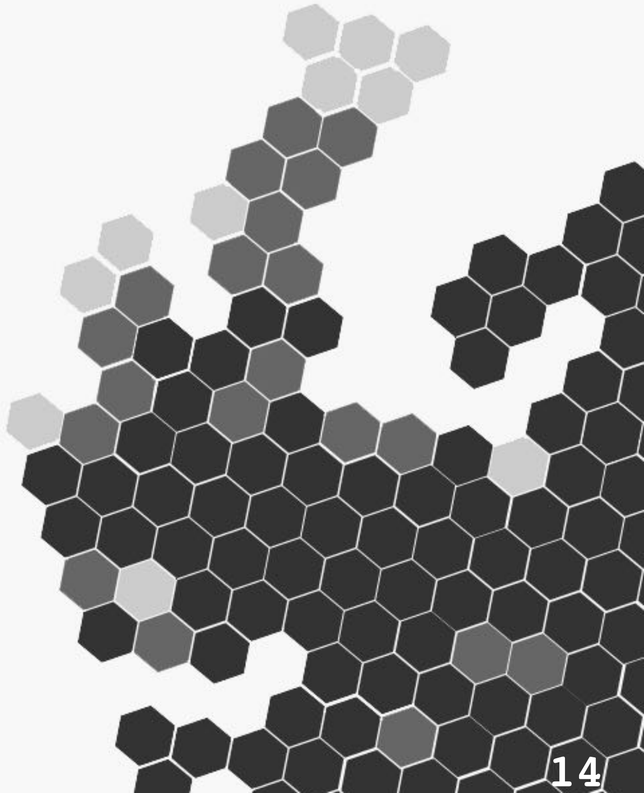
- ***Asynchronous Advantage Actor-Critic (A3C)*** [Mnih et al. *ICML'16*]: extend AC
  - Accelerate and stabilize the training with parallelism in multiple agents

Deep Reinforcement Learning (DRL)

- **Trust Region Policy Optimization (TRPO)** [Schulman et al. *JMLR'15*]
  - An analytic approach towards training stability issues in policy gradient method
  - Avoid parameter updates changing the policy too much within one single step
- ***Proximal Policy Optimization (PPO)*** [Schulman et al. *arXiv'17*]
  - Further simplify TRPO while maintaining similar performance
  - Place soft constraints of policy difference instead of hard constraints.
- ***Monte Carlo Tree Search (MCTS)*** [Silver et al. *Nature'16*, *Nature'17*]
  - Efficiently search large space for optimal value by a *double approximation*.
  - Can be integrated with large neural networks as value function or policy.

Outline

- Introduction to Reinforcement Learning (RL)
- RL Applications in Cloud Management
  - Network Optimization
  - Cluster Scheduling
  - Virtual Machine Configuration
  - Power Management

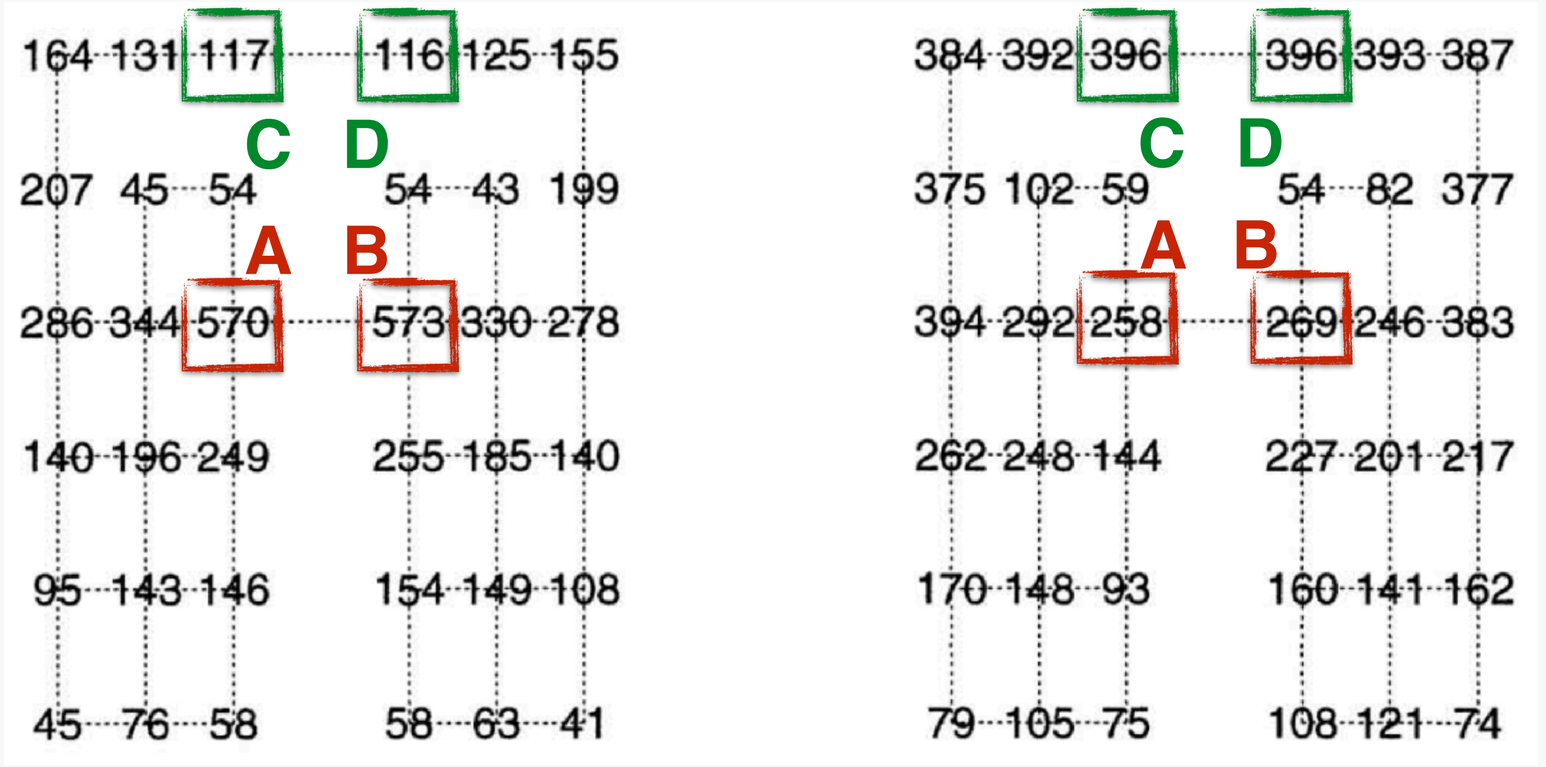


Network Optimization: Preview

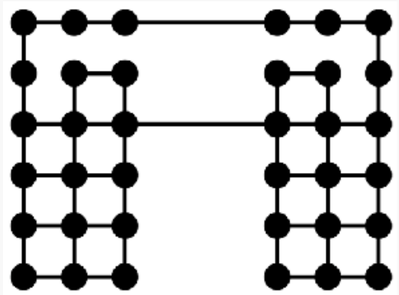
	Tasks	**Publication**	**Authors**
Q-routing, PQ-routing	Routing	NIPS'93, NIPS'96	Boyan & Littman, Choi & Yeung
Pensieve	Adaptive Bitrate	SIGCOMM'17	Mao et al.
Pytheas	Server Selection	NSDI'17	Jiang et al.
AuTO	Traffic Opt. (e.g., Flow scheduling)	SIGCOMM'18	Chen et al.

Network Optimization: Q-routing

- Use Q-learning to route packets in a network [Boyan & Littman, NIPS'93]
  - Policy determines *which adjacent node* to send a packet to, using **local info.**
  - Learn to avoid network congestions on *shortest path* (A-B) under *high load*



Topology



Shortest Path

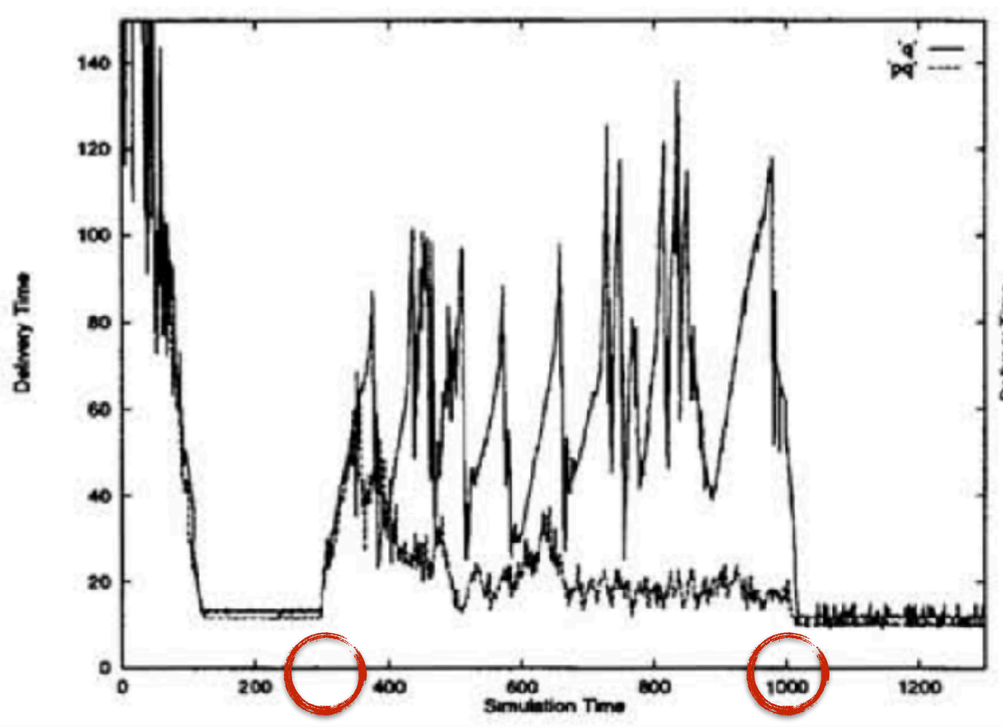
Q-routing

Network traffic load on each node, under different policies

Network Optimization: PQ-routing

- ***Predictive Q-routing***: [Choi & Yeung, NIPS'96]
  - Verify Q-routing and improve its learning speed and adaptability
  - ***Exploration* vs. *Exploitation***:
    probing shortest path's congestion recovery with controlled frequency.

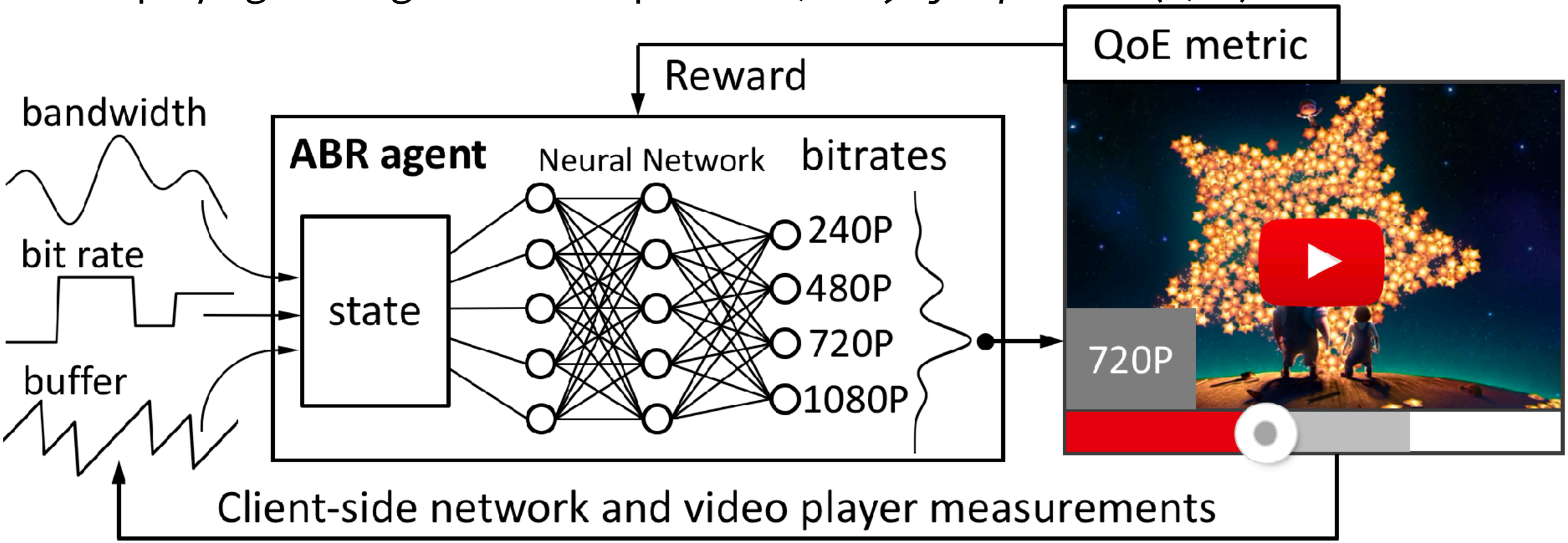
- Problem settings (identical to Q-routing)
  - *State*: current packet's destination node d
  - *Action*: route packet to which neighbor node y
  - *Value function* $Q_x(d, y)$: *delivery time*
    estimated from node x to node d via node y.



Network Optimization: Adaptive Bitrate (ABR)

[Mao et al. SIGCOMM'17]

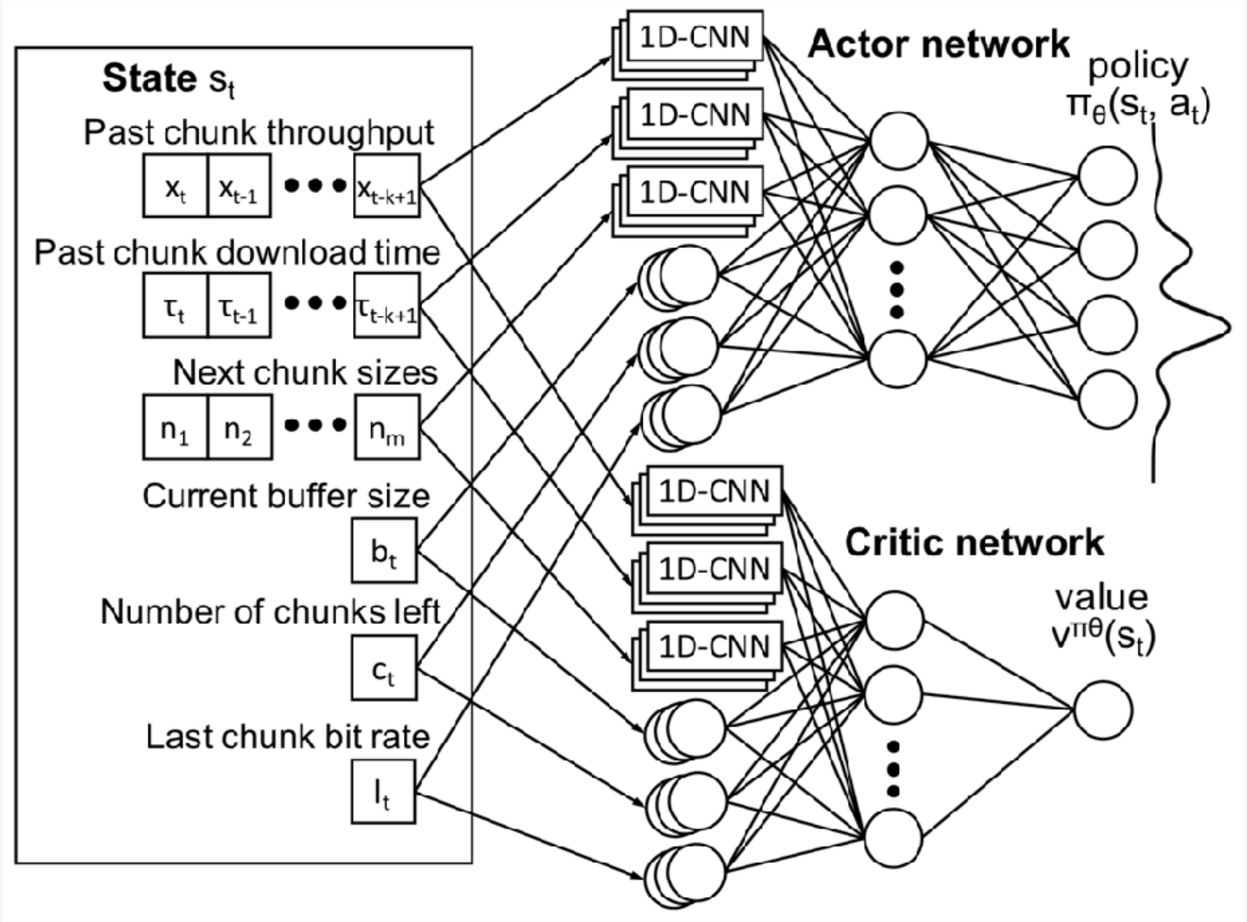
- Client-side video players with *Dynamic Adaptive Streaming over HTTP* (*DASH*): employing *ABR* algorithms to optimize *Quality of Experience* (*QoE*).

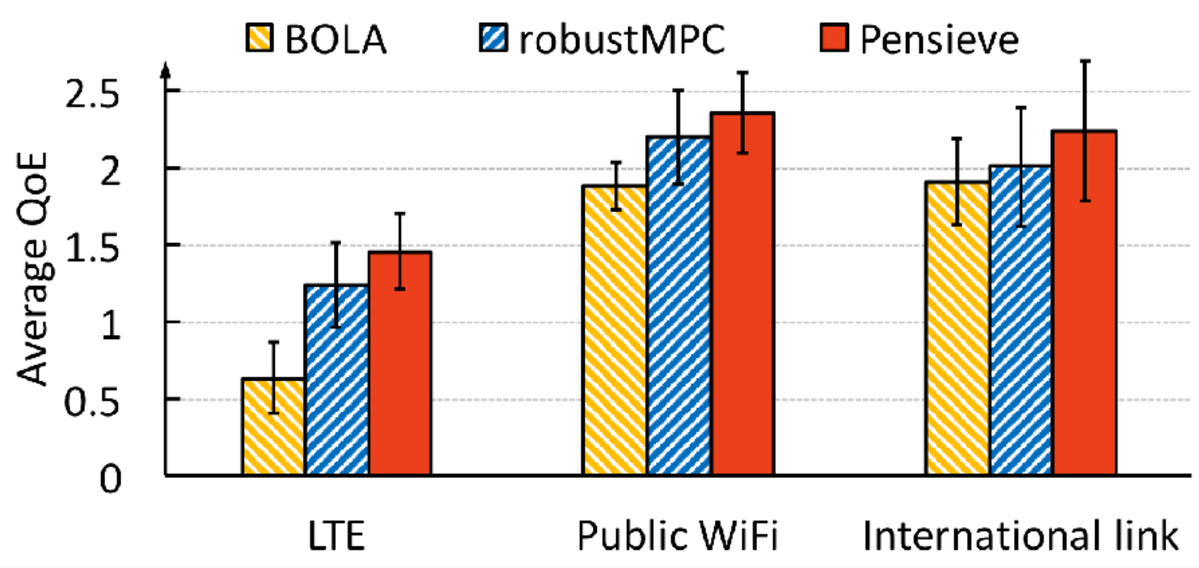


Network Optimization: Adaptive Bitrate (ABR)

[Mao et al. SIGCOMM'17]

- ***Pensieve***, the proposed ABR system, makes decisions solely through observations.



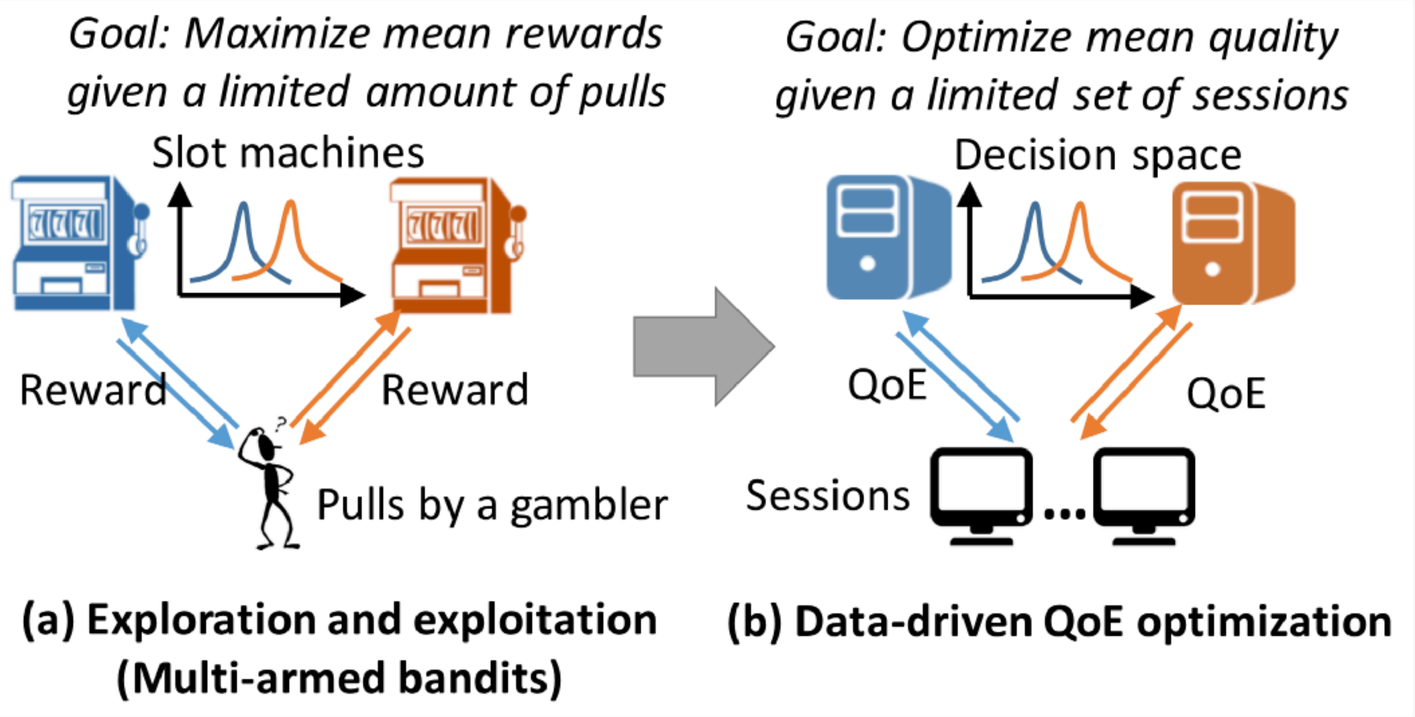


- *BOLA* [CoRR'16]: solves optimization problem considering buffer occupancy observations
- *robustMPC* [SIGCOMM'15]: uses buffer occupancy observations and throughput predictions

Network Optimization: Server Selection

[Jiang et al. NSDI'17]

- ***Pytheas***: Data-driven QoE optimization via group-based exploration-exploitation (*E2*)
- Prediction-based mechanisms: 1) incomplete info. biases; 2) insensitive to changes
- *E2* in real-time: measurement collection and decision making as a joint process
- Use cases: server selection for
  - Live streaming
  - Video on demand
  - Voice over IP (e.g., Skype)
  - File sharing (e.g., Dropbox)
  - Search, social, web services



Network Optimization: Server Selection

[Jiang et al. NSDI'17]

- *Group-based* for **scalability**: group sessions with *similar network context* (IP prefix, location), which usually share same network- and application-level features.
- Algorithm: Discounted *Upper Confidence Bound (UCB)** algorithm [Auer et al. ML '02]

- Case study: video streaming sessions choosing 1 of 2 CDNs
  - baseline: prediction-based method, i.e., *CFA* [ACML'15]
  - trace: 8.5 million video sessions in US over 24 hours
  - QoE: sampled from trace

*: select arm $I = \arg\max_i \{\frac{w_i}{n_i} + c\sqrt{\frac{\ln n}{n_i}}\}$

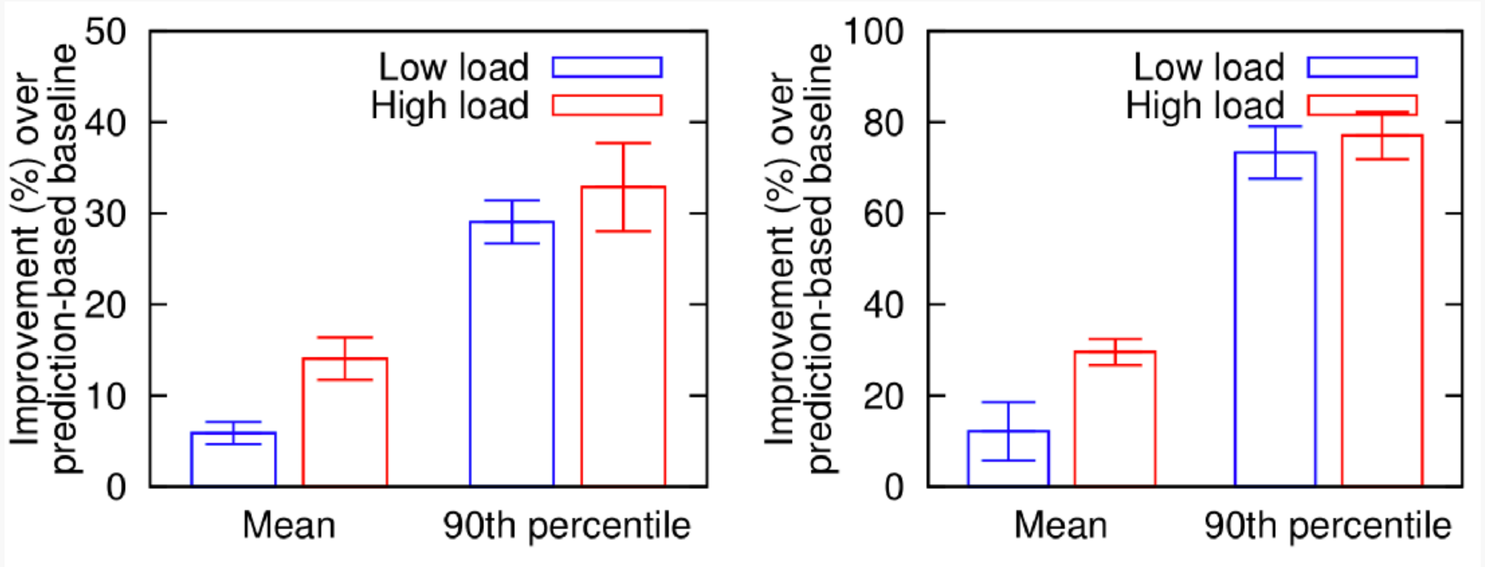


Figure 9: *Improvement in presence of load effect.*

Network Optimization: Datacenter Traffic

[Chen et al. SIGCOMM'18]

- ***AuTO***: end-to-end Automatic Traffic Optimization (TO) system ***at datacenter scale***
- Observation: designing TO heuristics that matches traffic takes weeks, but automating it with DRL systems cannot handle flow-level TO at datacenter scale.
  - *60ms*, during which *1Gbps* link can finish flows within 7.5MB, > *95%* flows in real DC traffic.

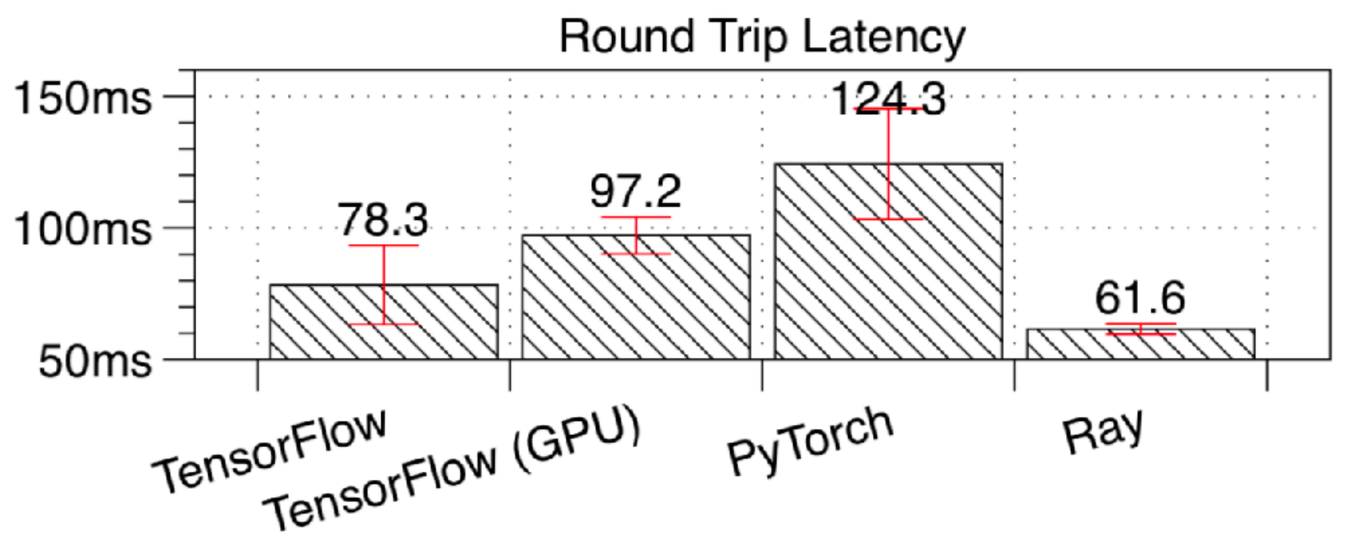


Figure 2: Current DRL systems are insufficient.

Network Optimization: Datacenter Traffic

[Chen et al. SIGCOMM'18]

- Solution: two-level design: *Peripheral Systems (PS)* and *Central Systems (CS)*
  - *PS*: run on all end-hosts, collect flow info., make decisions for *short flows ASAP*.
  - *CS*: optimize *Multi-Level Feedback Queuing (MLFQ)* thresholds for *short flows*, and make individual TO decisions for *long flows*, with two DRL agents respectively.

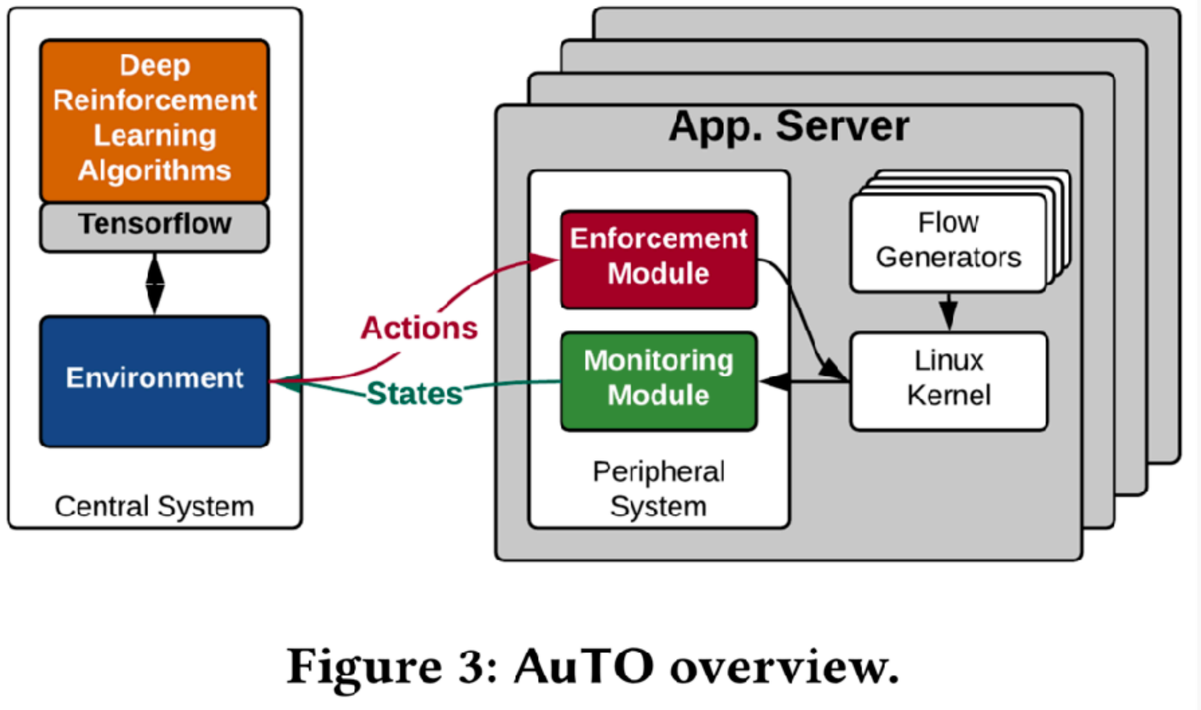
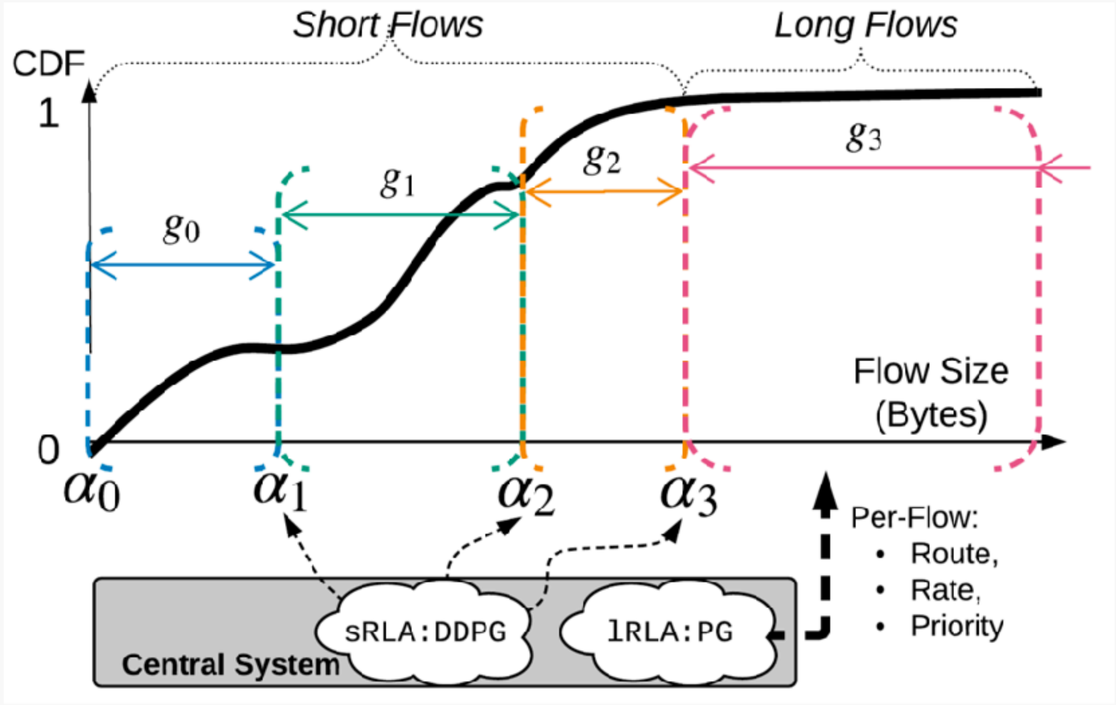


Figure 3: AuTO overview.



Network Optimization: Summary

	Tasks	**Algorithm**	**Scale**
Q-routing, PQ-routing	Routing	Q-learning	Single node
Pensieve	Adaptive Bitrate	A3C	Single node
Pytheas	Server selection	UCB	Multiple nodes
AuTO	Traffic Opt. (e.g., Flow scheduling)	PG & DDPG	Datacenter

Cluster Scheduling: Preview

	Job Features	**Publication**	**Authors**
DeepRM	Resource demand	HotNets'16	Mao et al.
Spear	Resource & Task DAG	ICDCS'19	Hu et al.
Decima	Resource & Task DAG & Parallelism lv.	arXiv'18 (updated in 2019)	Mao et al.

Cluster Scheduling: Basic Job Scheduling

- ***DeepRM***: resource management with deep learning [Mao et al. HotNets'16]
  - Challenges of cluster scheduling with heuristics:
    - Modeling underlying systems inaccurately
    - Online decisions with noisy inputs
    - Optimizing complex performance metrics
  - RL approaches are suitable because:
    - Modeling systems with deep neural networks
    - Repetitive decisions as training data
    - Associating reward with metrics of interests

Cluster Scheduling: Basic Job Scheduling

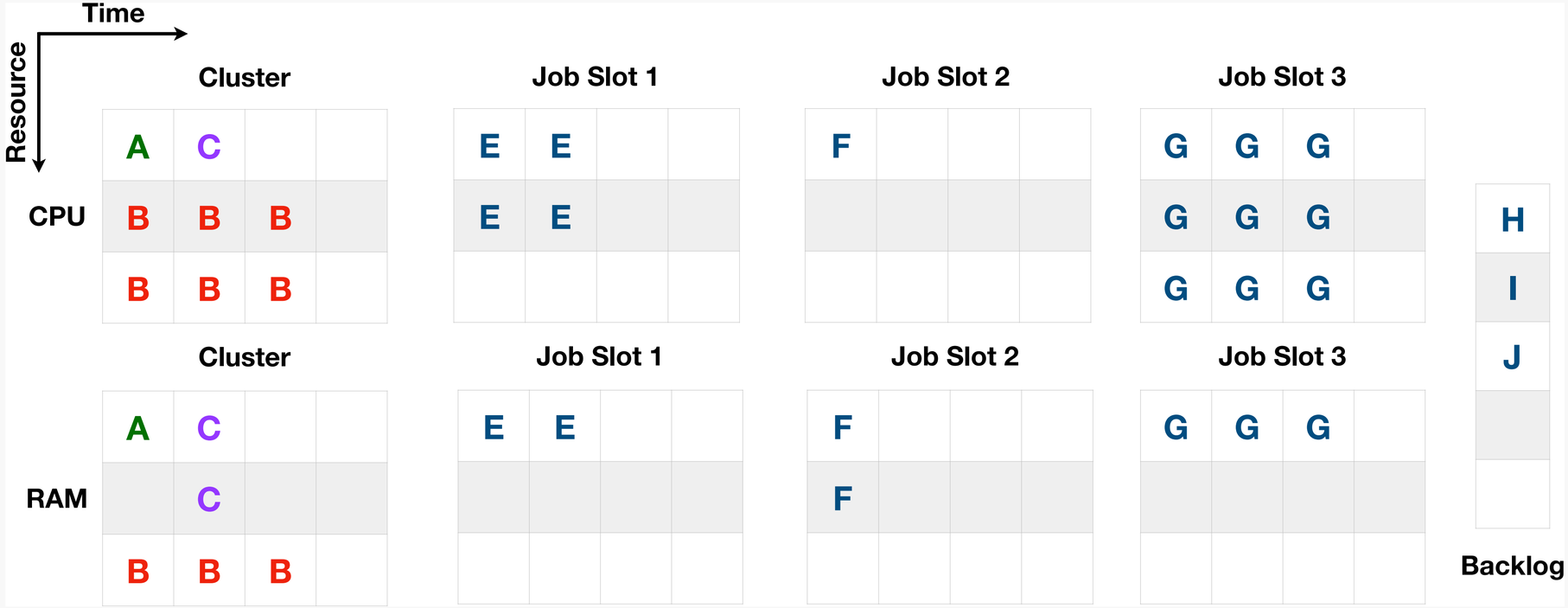
[Mao et al. HotNets'16]

- *States*: represented as *images*.
  - Each job: *required-resource-units* X *duration-in-timesteps. (e.g., B: 2 CPU x 3 ts)*
- *Actions*: any subset of M jobs. How to be $O(M)$?
  $a \in \{1, \cdots, M\}$: schedule the i-th job; a=$\varnothing$: deploy the decisions
- Algorithm: *REINFORCE*, using NN of 1 FC-layer with 20 neurons.

4 Fake timesteps

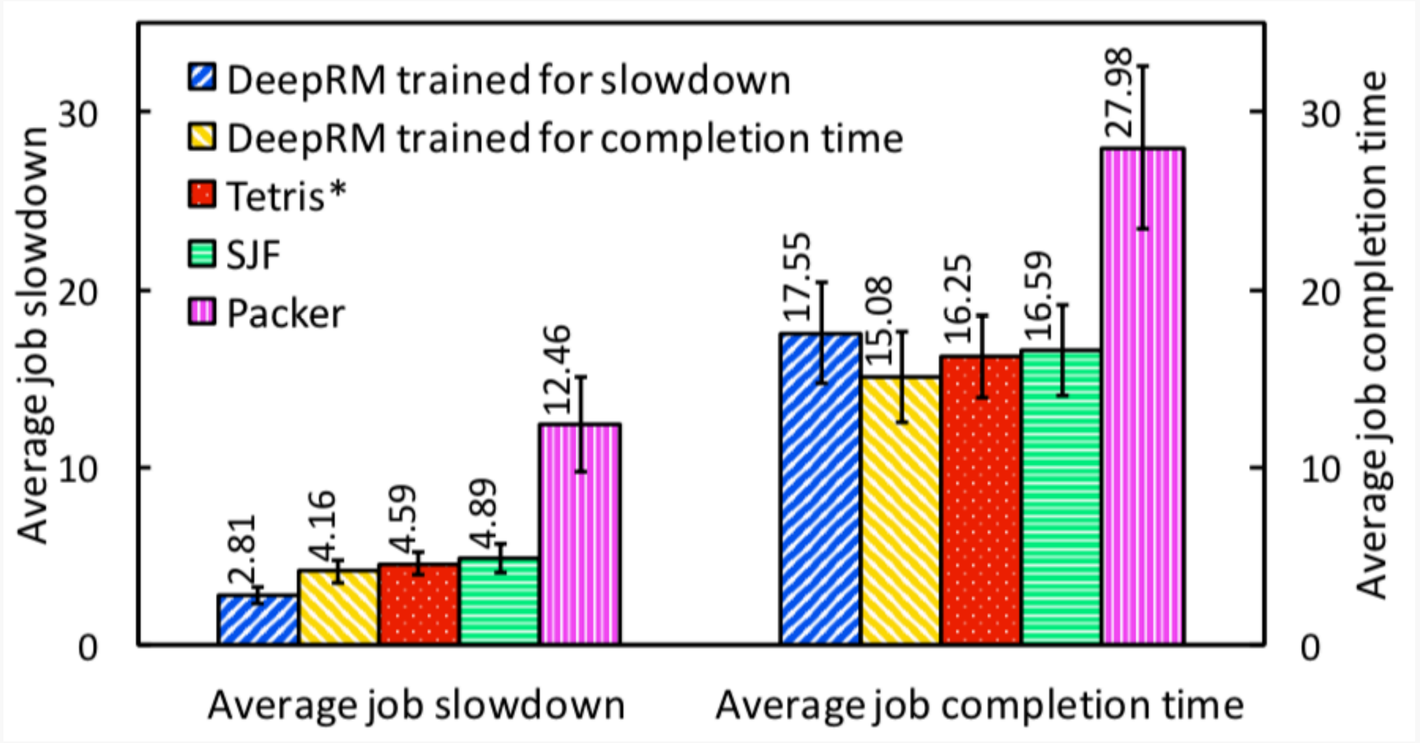
J1	J3	J2	∅

— 1 Real timestep



Cluster Scheduling: Basic Job Scheduling

- *Objective 1:* minimizing sum of all jobs' slowdowns [Mao et al. HotNets'16]
  (i.e., real competition time Cj / ideal job duration Tj . $j \in \mathcal{J}$: unfinished jobs)
  - *Reward* := $\sum_{j \in \mathcal{J}} -1/T_j$
- *Objective 2*: minimizing average completion time
  - *Reward* := $-|\mathcal{J}|$

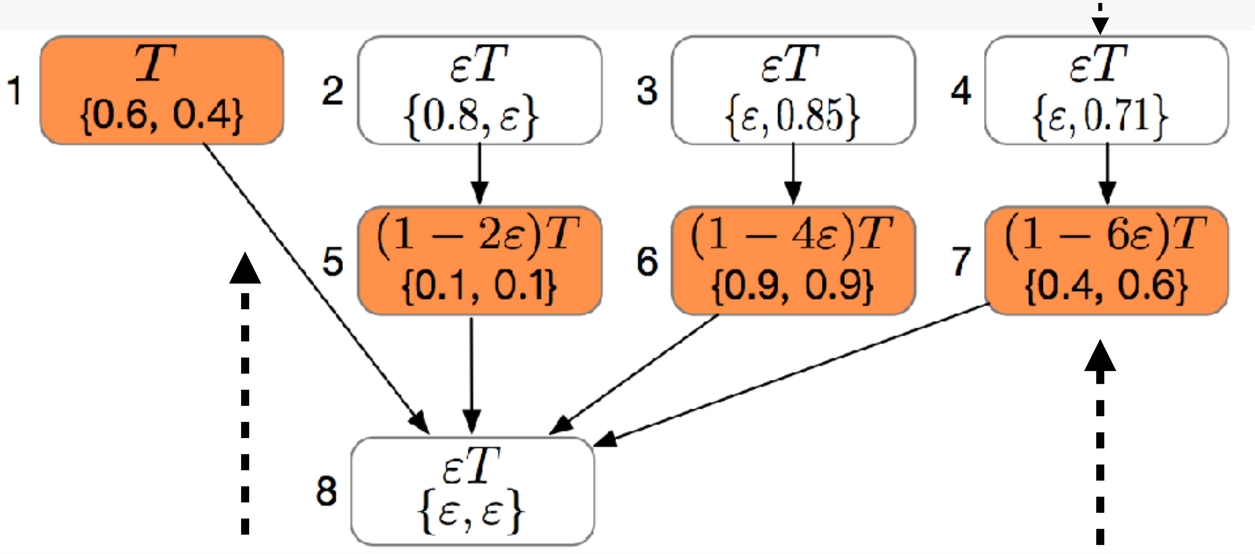


Evaluated by simulation with synthetic workload

Cluster Scheduling: DAG Job Scheduling

- ***Spear***: dependency-aware task scheduling with DRL. [Hu et al. ICDCS'19]
- Key: considering both *resource demands (packing)* and *task dependencies (*in *DAG).*
  - *Resource demands*: *required-resource* X *task-duration*
  - *Task Dependencies*: features including nodes' *b-level**, *num-of-children*, *b-load^*.

DAGs: directed acyclic graphs



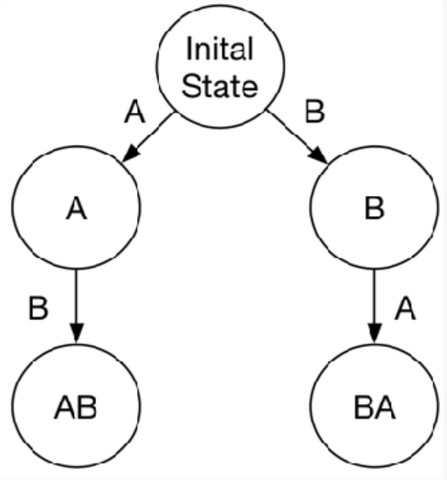
Algorithms	***Scheduling Order***	***Time***
Our Approach	2-3-6-5-4-1-7-8	2T
Graphene	4-2-3-7-6-1-5-8	3T
Tetris	1-3-2-6-5-4-7-8	3T
Critical path	1-2-5-3-6-4-7-8	3T

*: length of the longest path to exit node

^: sum of *resource demands* along the path

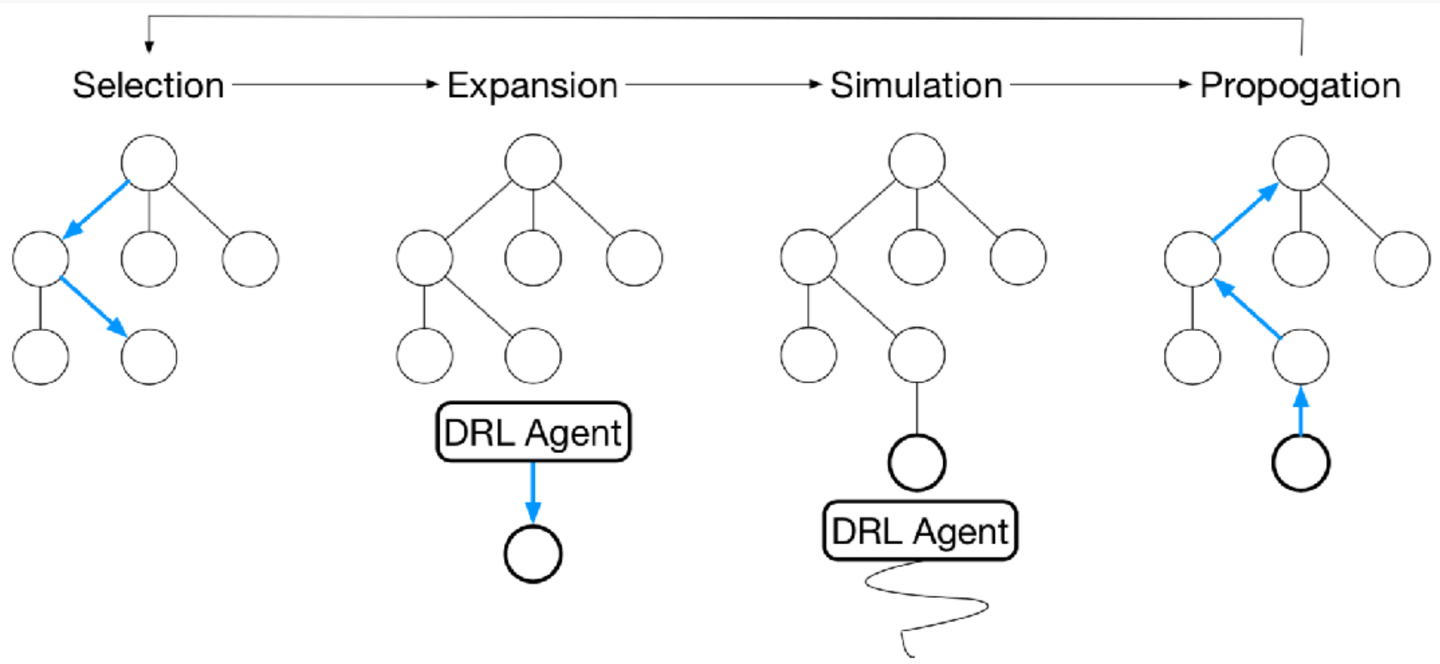
Cluster Scheduling: DAG Job Scheduling

- *Actions*: determine order of M tasks,
  $a \in \{\varnothing, 1, \cdots, M\}$: schedule the i-th task;
  $a=\varnothing$: deploy the decisions.



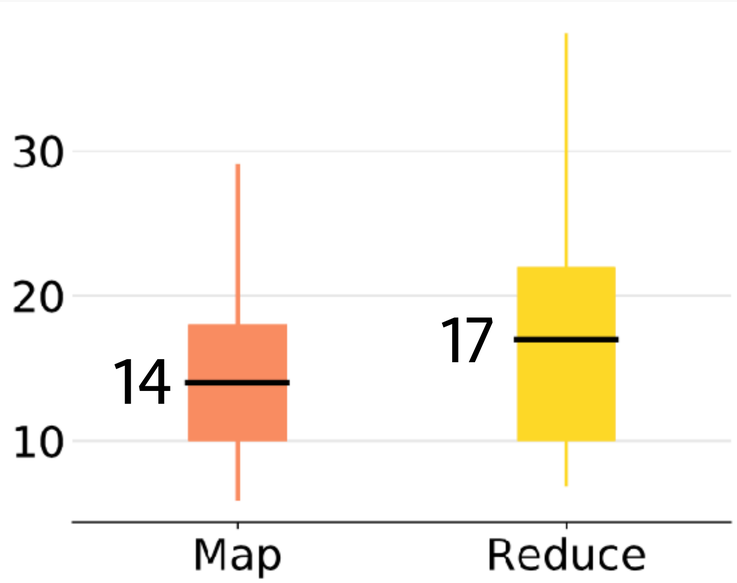
[Hu et al. ICDCS'19]

- Algorithm: *Monte Carlo Tree Search (MCTS)* with DRL (Policy Gradient)
  - *Selection*: follow *UCB* algorithm;
  - *Expansion*: select node to explore (with trained DRL policy);
  - *Simulation*: *rollout*, i.e., act to end, (with trained DRL policy);
  - *Propagation*: $v \leftarrow \max\{v_{old}, v_{new}\}$
    v: negative of expected *makespan*

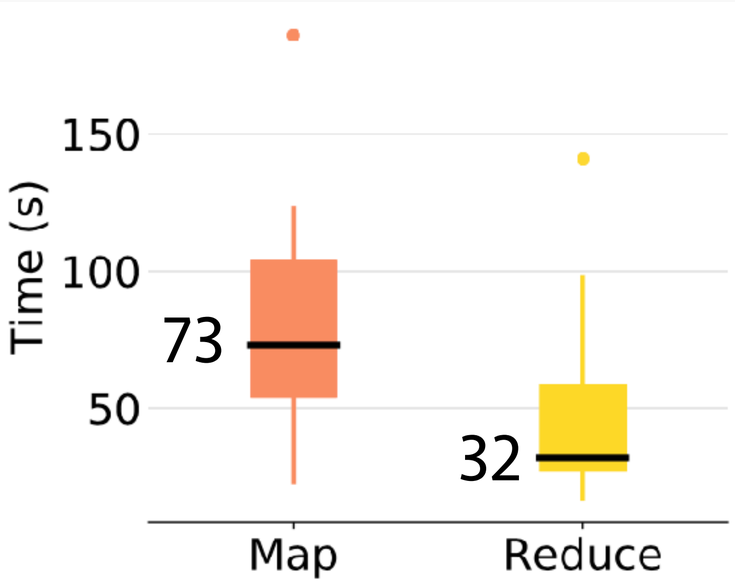


Cluster Scheduling: DAG Job Scheduling

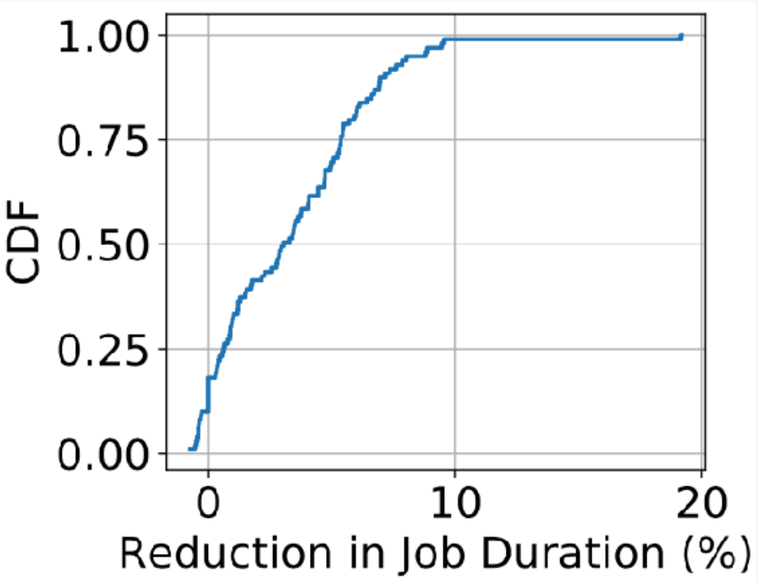
- Evaluation on production Apache Hive workloads [Hu et al. ICDCS'19]
  - 99 MapReduce jobs with ≥ 5 map or reduce tasks
  - ***Spear*** outperforms *Graphene* [NSDI'16] in 90% jobs with at most 20%.



(a) The number of tasks in the jobs.



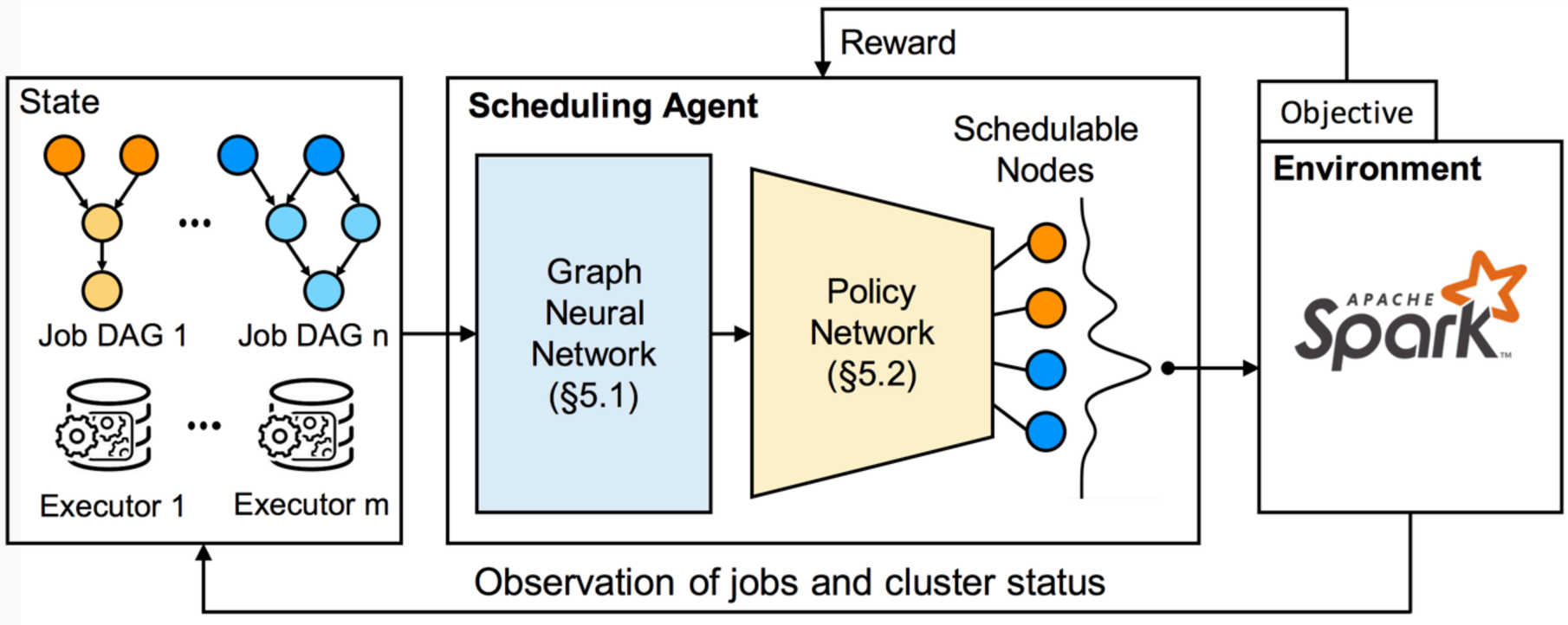
(b) The task runtime.



(c) Results comparing with Graphene.

Cluster Scheduling: Scalable DAG Scheduling

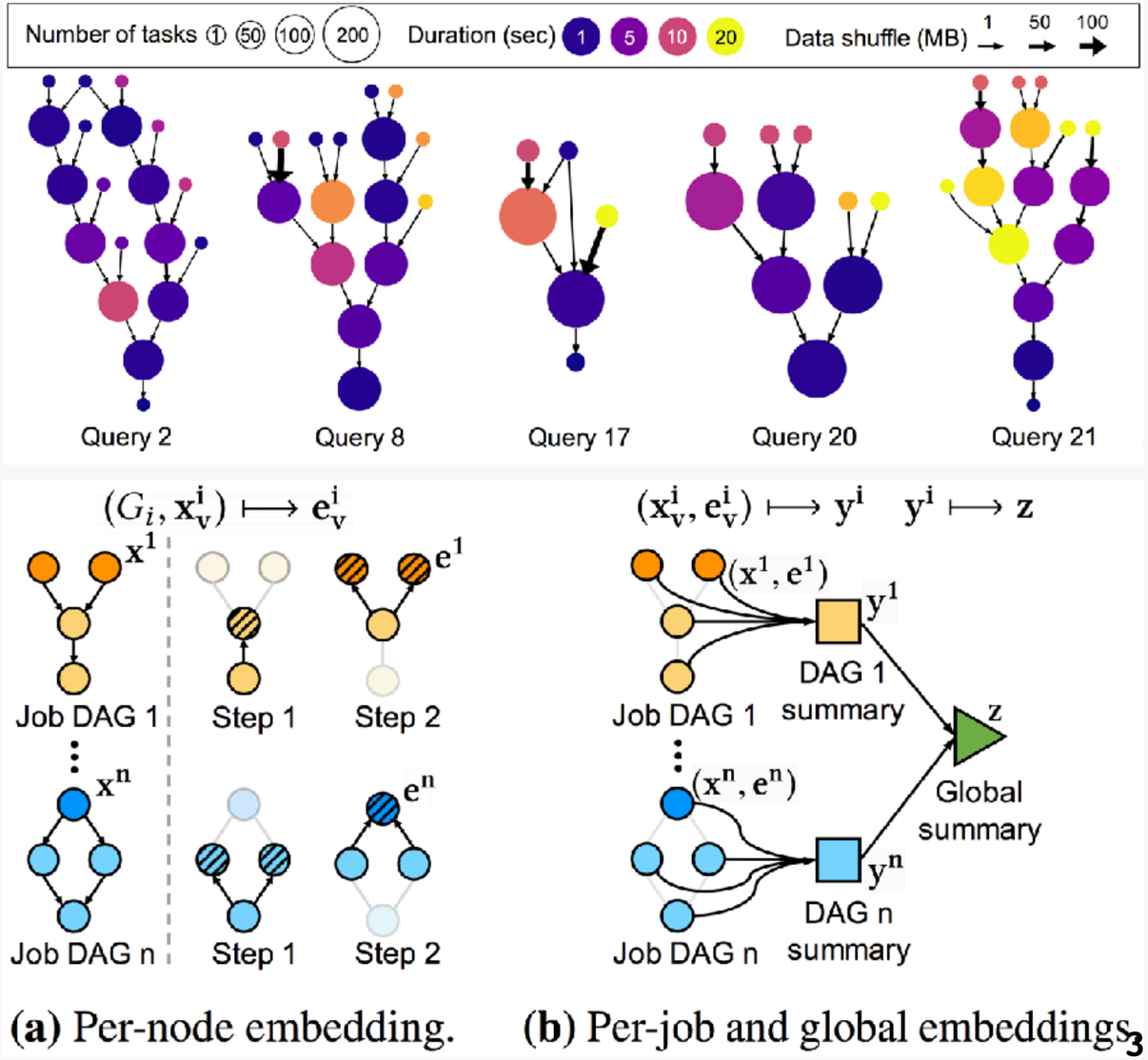
- ***Decima***: Learn to schedule for data processing clusters [Mao et al. arXiv'18]
  - Dependency-awareness with *Graph Neural Networks (NNs)*
  - Configure job's *parallelism* level while scheduling task nodes with *Policy NNs*
  - Scalability (with jobs & machines) & Robustness (to various job arrival patterns)



Cluster Scheduling: Scalable DAG Scheduling

- Dependency-awareness by 3 levels of graph embedding [Mao et al. arXiv'18]
  - Data-parallel jobs have complex data-flow graphs with different attributes
  - But *policy neural networks* usually require *fixed-sized* vectors as input
- Solution: uses a *graph neural network* to encode *any number* of *arbitrary* DAGs
  1. Each task node & its children
  2. All task nodes in the DAG
  3. All per-job embeddings
- Embedding: $g(\cdot)$, $f(\cdot)$: NNs

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



Cluster Scheduling: Scalable DAG Scheduling

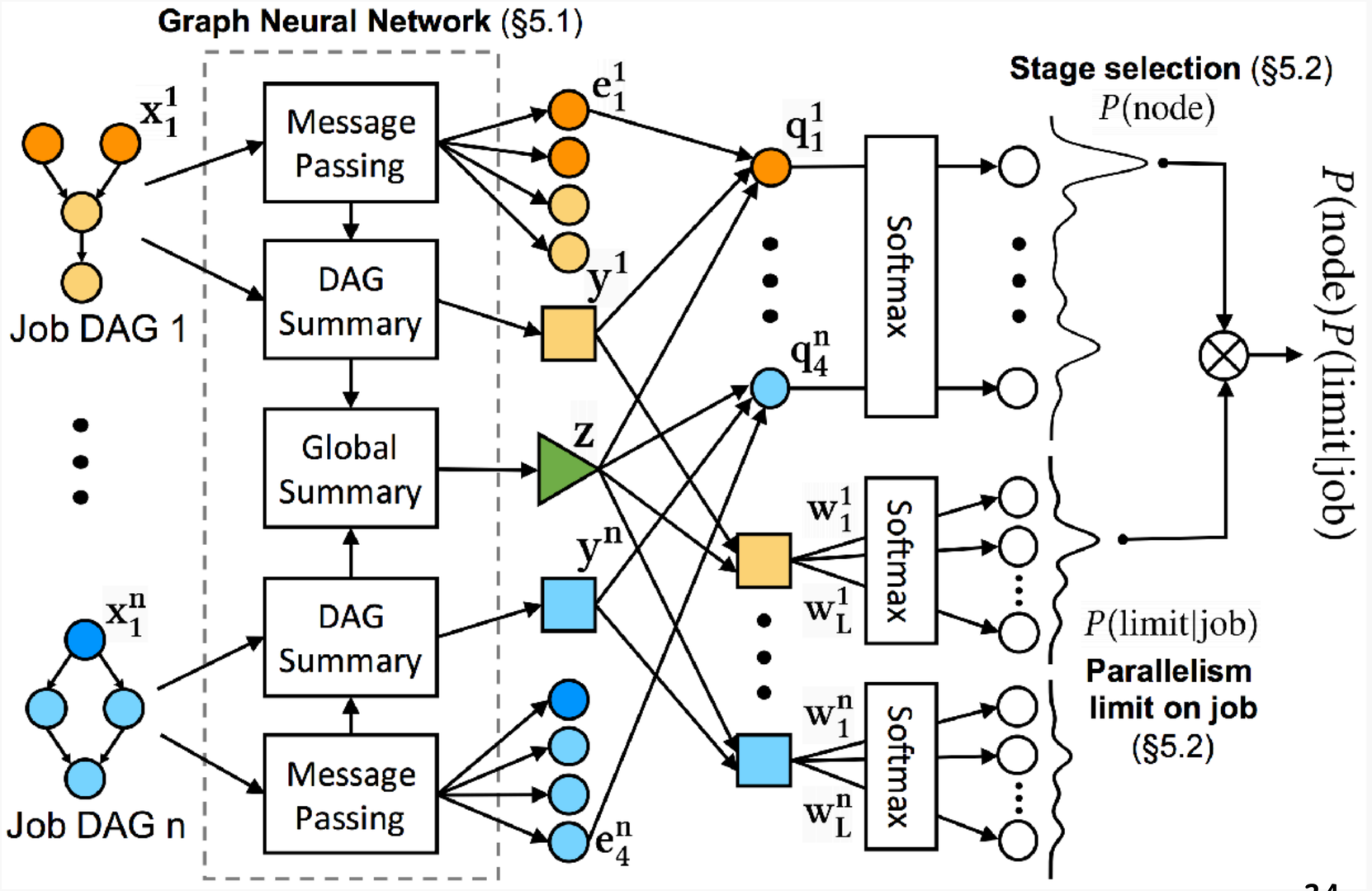
- Configure job's parallelism level & schedule task nodes [Mao et al. arXiv'18]
  - Node selection policy: *softmax* on scores q based on 3 embeddings

$$q_v^i \triangleq q\left(\mathbf{e}_v^i, \mathbf{y}^i, \mathbf{z}\right)$$

  - Parallelism limit selection: *softmax* on scores of different limit l of different jobs with the same function w:

$$w_l^i \triangleq w\left(\mathbf{y}^i, \mathbf{z}, l\right)$$

  - $q(\cdot), w(\cdot)$: NNs

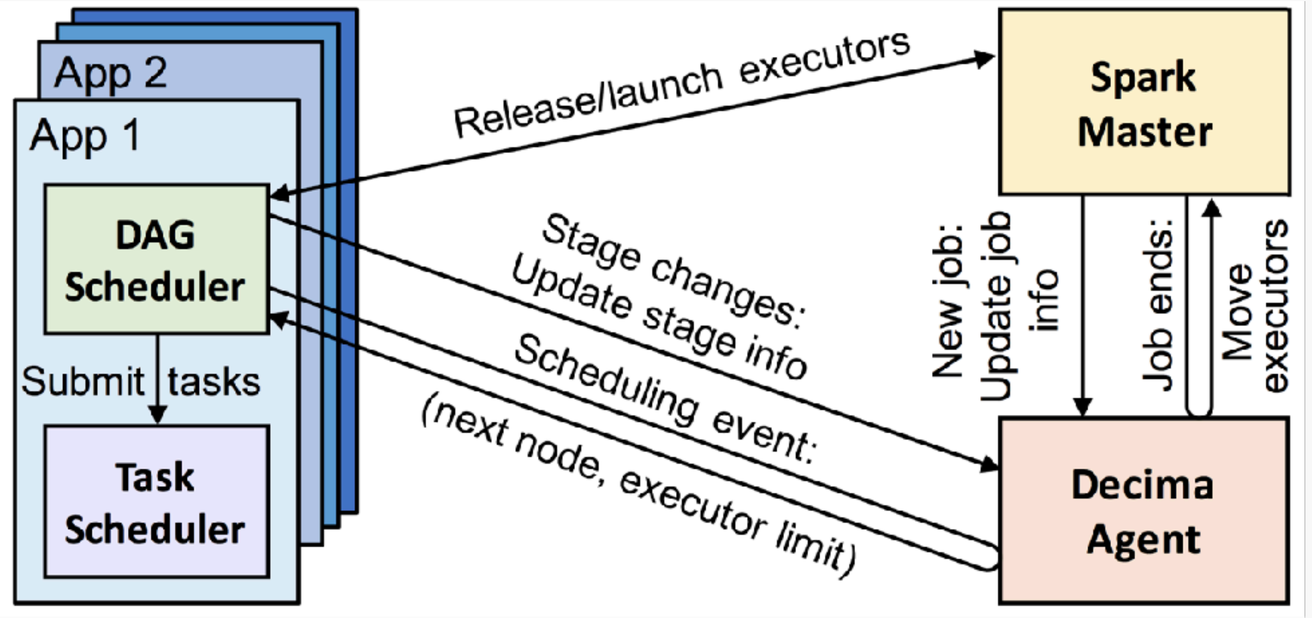


Cluster Scheduling: Scalable DAG Scheduling

- Scalability with jobs and machines

  [Mao et al. arXiv'18]

  - 3 $g(\cdot)$, 3 $f(\cdot)$, 1 $q(\cdot)$, 1 $w(\cdot)$: 8 NNs are *reused* for all *jobs*.
  - ***Decima*** as a service atop other cluster platforms (e.g., Spark, Mesos).
  - Scheduling latency: on avg. < 15ms, ~50x smaller than events intervals.

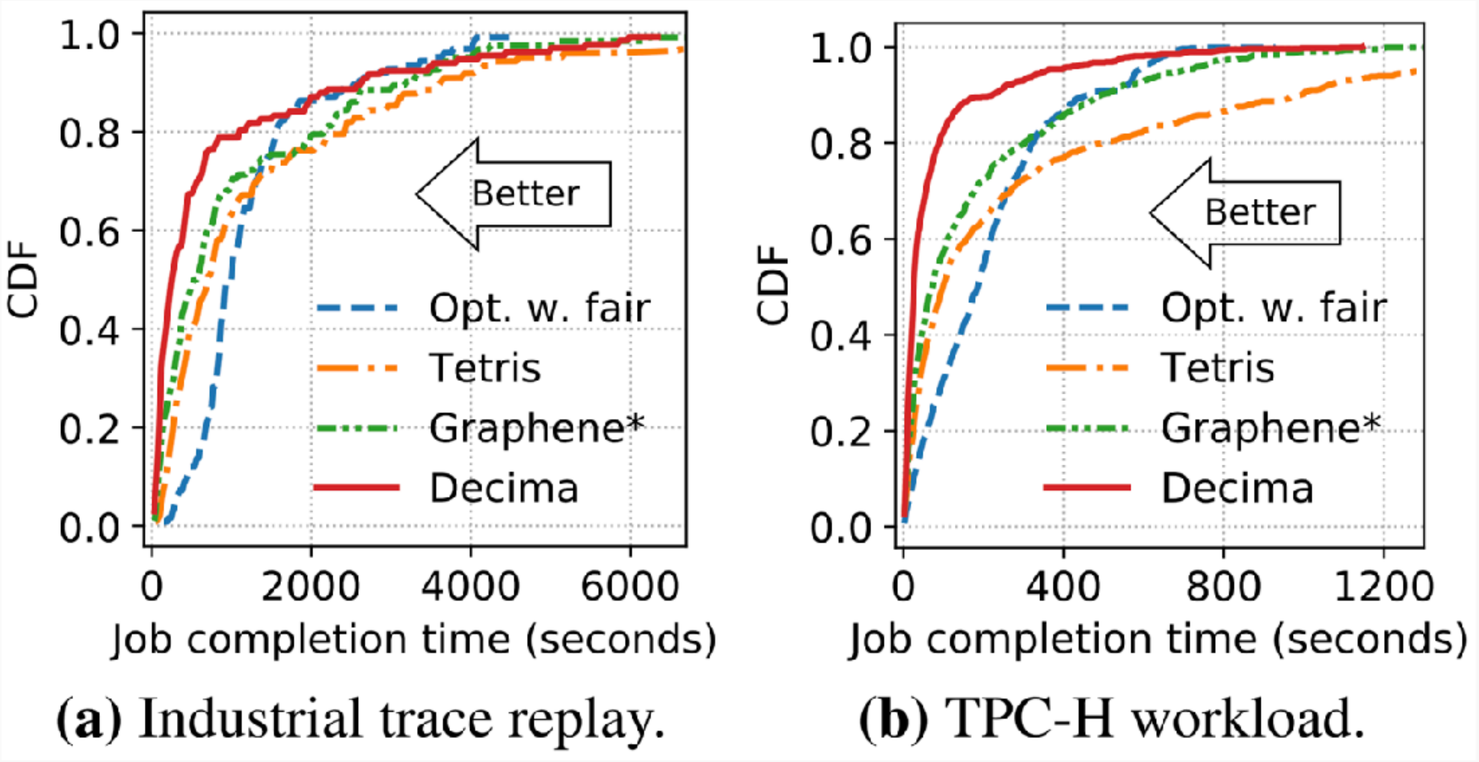


- Robustness to various arrival patterns
  - Different job arrival patterns have large impact on performance which affect *rewards*.
  - Solution: for each arrival sequence, compute separate *baselines* b_k, subtracted in policy gradient algorithm REINFORCE:

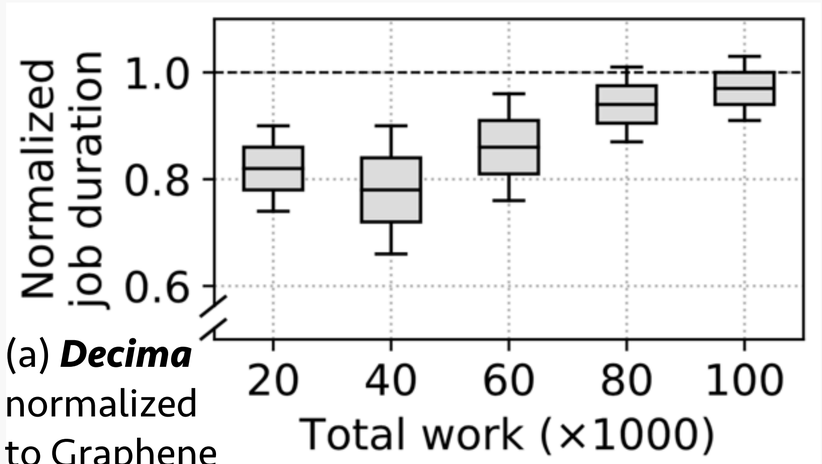
$$\theta \leftarrow \theta + \alpha \sum_{k=1}^{T} \nabla_\theta \log \pi_\theta (s_k, a_k) \left(\sum_{k'=k}^{T} r_{k'} - b_k \right)$$

Cluster Scheduling: Scalable DAG Scheduling

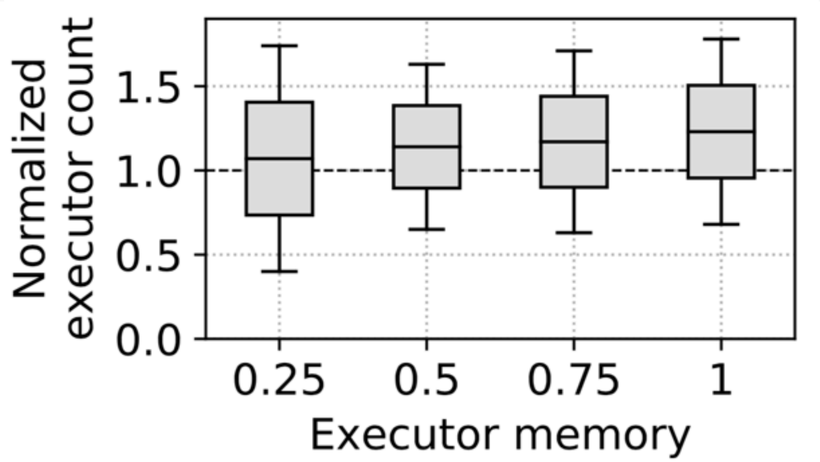
- Evaluation with performance analysis



(a) Industrial trace replay. (b) TPC-H workload.



(a) ***Decima*** normalized to Graphene



(b) Number of executors that Decima uses for "small" jobs, normalized to Graphene*.

- ***Decima*** trades off memory fragmentation against clearing the job queue more quickly:
  - using 39% more executors of largest class on the jobs with smallest 20% total work.

Cluster Scheduling: Summary

	Job Features	**Algorithm**	**NN Structure**
DeepRM	Resource demand	REINFORCE	1 FC-layer w/ 20 hidden units
Spear	Resource & Task DAG	MCTS & PG	3 FC-layer w/ 256, 32, 32 units
Decima	Resource & Task DAG & Parallelism lv.	REINFORCE	8 x [2 FC-layer w/ 32, 16 units]

Virtual Machine Configuration: Summary

	Highlights	**Algorithm**	**Publication**
iBalloon	learning agent for each VM	Q-learning	SIGMETRICS'11 (poster) MASCOTS'11
URL	VM & Application co-configuration	Q-learning	JPDC'12
CoTuner	To larger cluster; reduce space by Simplex method	TD-learning	TPDS'13

Power Management: Summary

	Highlights	**Algorithm**	**Publication**
Adaptive DPM	Model-free	Q-learning	ICCAD'09
Near-optimal DPM	Continuous-time, Workload Predict	TD(λ), Naïve Bayes	DAC'11
Hierarchical PM and Resource Allocation	Cluster & per-node RL agent, Deep Learning	Q-learning, FNN, LSTM, Auto-Encoder	ICDCS'17

Summary of Common Challenges

- **Adaptive to Dynamic Workload**
  - Taking workload prediction as state input. [***Near-optimal DPM, Hierarchical PM***]
  - Building separate baselines and subtracting them when updating. [***Decima***]
- **Cold Start of RL Agent**
  - Initializing network with supervised learning to mimic heuristics policy. [***Spear***]
  - Sampling typical or default configurations as starting point. [***URL***]
- **Data Collection**
  - Faithful simulation capturing critical real-world effects. [***Pensieve***, ***Decima***]
  - Collecting data on the job. [***Pytheas***, ***AuTO***]

Summary of Common Challenges

- **Scalability**
  - “Divide and conquer” (sharing policy or NN). [***Decima***, ***iBalloon***, ***Hierarchical PM***]
  - Compressing global information: e.g., embedding, AE. [***Decima***, ***Hierarchical PM***]
- **Space Reduction**
  - Decoupling actions to different RL agents. [***Decima***, ***Hierarchical PM***]
  - Converting combinatorial decisions to sequential ones. [***DeepRM***, ***Spear***]
  - Merging different models with action masking. [***Pensieve***]
  - Discarding lower-priority information. [***DeepRM***]
  - Preprocessing with other methods or heuristics. [***CoTuner***, ***Spear***]

Thank you

- ***Cloud Management with Reinforcement Learning***
  - Introduction to Reinforcement Learning (RL)
  - RL Applications in Cloud Management
    - Network Optimization
    - Cluster Scheduling
    - Virtual Machine Configuration
    - Power Management

Qizhen WENG
Supervisor: Prof. Wei WANG

CSE HKUST
PhD Qualifying Exam
8th May 2019