Deep Learning-based Job Placement in Distributed Machine Learning Clusters

Published in INFOCOM 2019

Yixin Bao, Yanghua Peng, Chuan Wu

香 港 大 學
THE UNIVERSITY OF HONG KONG

Review – *OASiS*[1] (INFOCOM 2018)

- The authors design an online algorithm for scheduling the arriving jobs and deciding the adjusted numbers of concurrent workers and parameter servers for each job over its course, to maximize overall utility of all jobs, contingent on their completion times.

- The online algorithm design utilizes an online primal-dual framework coupled with dual subroutines to efficiently tackle the combinatorial online optimization problem.

- The authors prove polynomial running time of the online algorithm and its long-term performance guarantee in terms of a good competitive ratio in total job utility.

[1]Bao, Yixin, Yanghua Peng, Chuan Wu, and Zongpeng Li. "Online job scheduling in distributed machine learning clusters." In IEEE INFOCOM 2018-IEEE Conference on Computer Communications, pp. 495-503. IEEE, 2018.

Background & Motivation - Challenge

In distributed ML training, how to efficiently place different ML jobs onto servers is a fundamental challenge. The problem is co-located ML jobs on the same server may have different levels of negative interference with each other because the jobs share underlying resources. However, existing schedulers used in practical ML clusters are largely interference-oblivious.

CPU caches	CTC – CNN for Sentence Classification
Disk I/O	AlexNet – reading images for preprocessing
Network I/O	VGG-16 – frequent parameter exchanges among workers

Background & Motivation – Case Study

Case study 1: bin packing vs standalone execution

Case study 2: pair-wise interference level

Case study 3: placement under representative policies

Background & Motivation – Case Study

Case study 1: bin packing vs standalone execution

- 6 DL jobs training from official MXNet tutorials

  Each job uses 1 parameter server and 1 worker for simplicity

Model	Application domain	Dataset
ResNet-50	image classification	ImageNet
VGG-16	image classification	ImageNet
ResNeXt-110	image classification	CIFAR10
Seq2Seq	machine translation	WMT17
CTC	sentence classification	mr
WLM	language modeling	PTB

- In experiment 1, each job is run on a dedicated server (standalone)

  In experiment 2, 6 jobs are packed onto 3 servers using multi-resource bin packing

Background & Motivation – Case Study

Case study 1: bin packing vs standalone execution

- Training speed & slowdown percentage

$$Slowdown = \frac{S\ of\ standalone\ - S\ of\ bin\ packing}{S\ of\ standalone}$$

S: Training speed (i.e., number of trained epochs per unit time)

Background & Motivation – Case Study

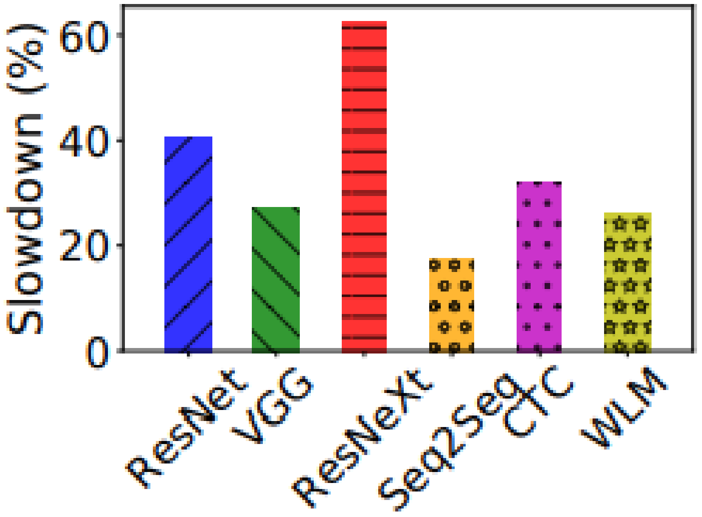
Case study 1: bin packing vs standalone execution

- Training speed & slowdown percentage

$$Slowdown = \frac{S\ of\ standalone\ - S\ of\ bin\ packing}{S\ of\ standalone}$$

S: Training speed (i.e., number of trained epochs per unit time)

- 30% performance degradation on average
- nearly 2x slowdown for training ResNet



Performance degradation
bin packing vs standalone

Background & Motivation – Case Study

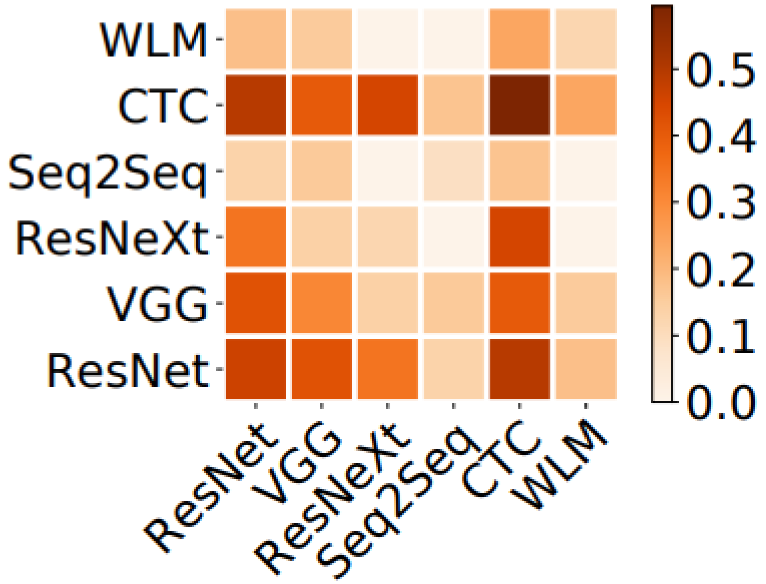
Case study 2: pair-wise interference level

- Co-locate each pair of jobs training two different models on 1 server
- Compute the sum of the slowdown percentages of the two jobs as the interference level

Background & Motivation – Case Study

Case study 2: pair-wise interference level

- Co-locate each pair of jobs training two different models on 1 server
- Compute the sum of the slowdown percentages of the two jobs as the interference level



Pair-wise interference (darker color indicates more severe interference)

Background & Motivation – Case Study

Case study 3: placement under representative policies

- 3 representative job placement policies

a) Load balancing: spreading workloads across servers evenly (Kubernetes)
b) Multi-resource bin packing (Google Borg)
c) Standalone execution

- 2 ML jobs

I. CTC
II. VGG-16/ResNeXt-110

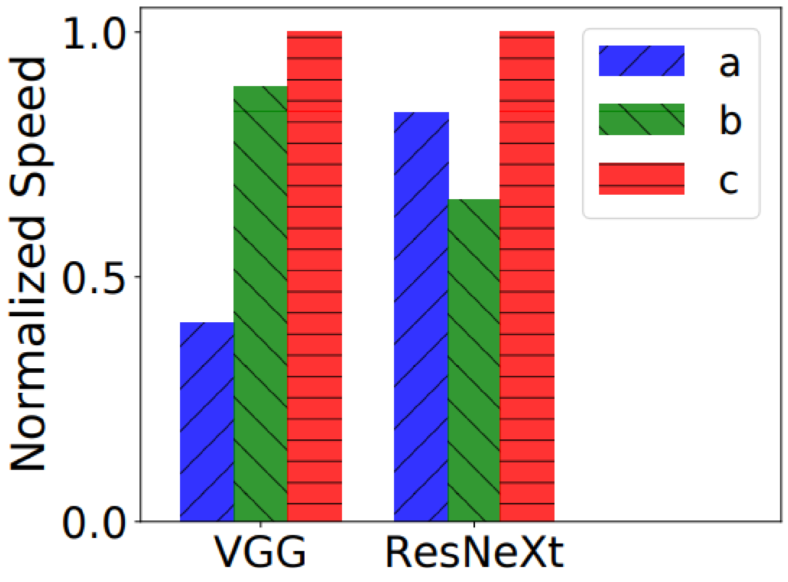
Placement under different schemes (diamonds represent parameter server and worker in job 1; squares represent those of job 2)

Background & Motivation – Case Study

Case study 3: placement under representative policies

- 3 representative job placement policies

a) Load balancing
b) Multi-resource bin packing
c) Standalone execution



Normalized training speed under 3 schemes

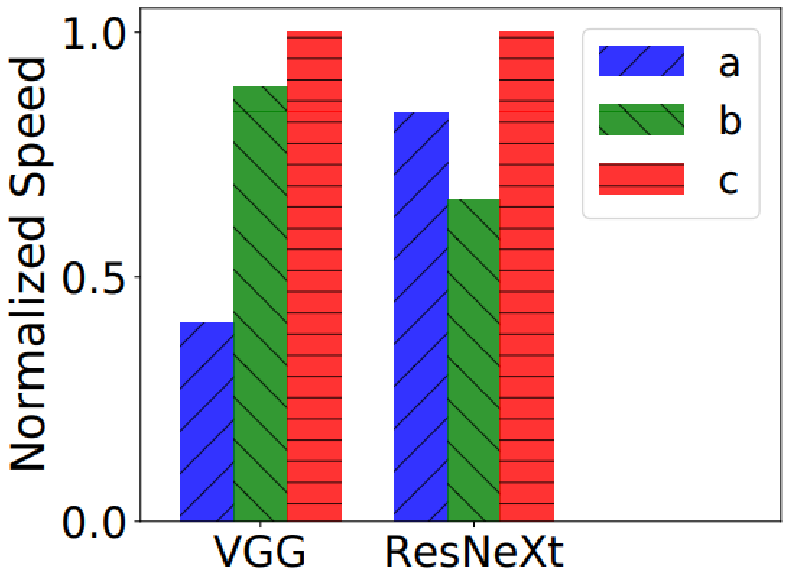
Background & Motivation – Case Study

Case study 3: placement under representative policies

- 3 representative job placement policies

a) Load balancing
b) Multi-resource bin packing
c) Standalone execution

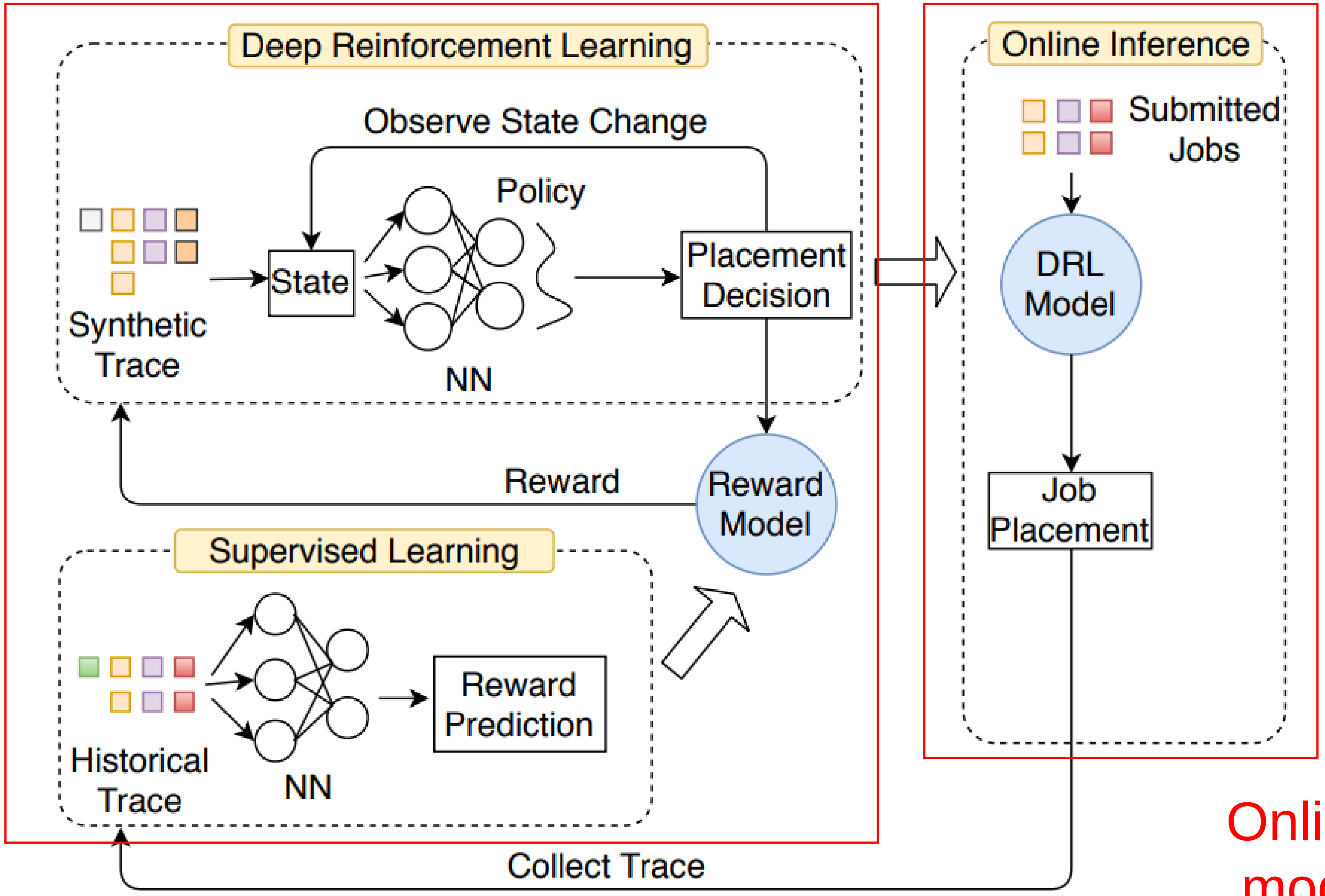
Severe interference between training ResNeXt and CTC together



Normalized training speed under 3 schemes

System Overview

Offline training



Online inference & model update

Harmony workflow

System Overview – Offline Training

- Produce a good model for online decision making
- Large historical traces containing enough samples may not always be available

- Two steps

➢ Reward model training
➢ DRL model training

System Overview – Offline Training

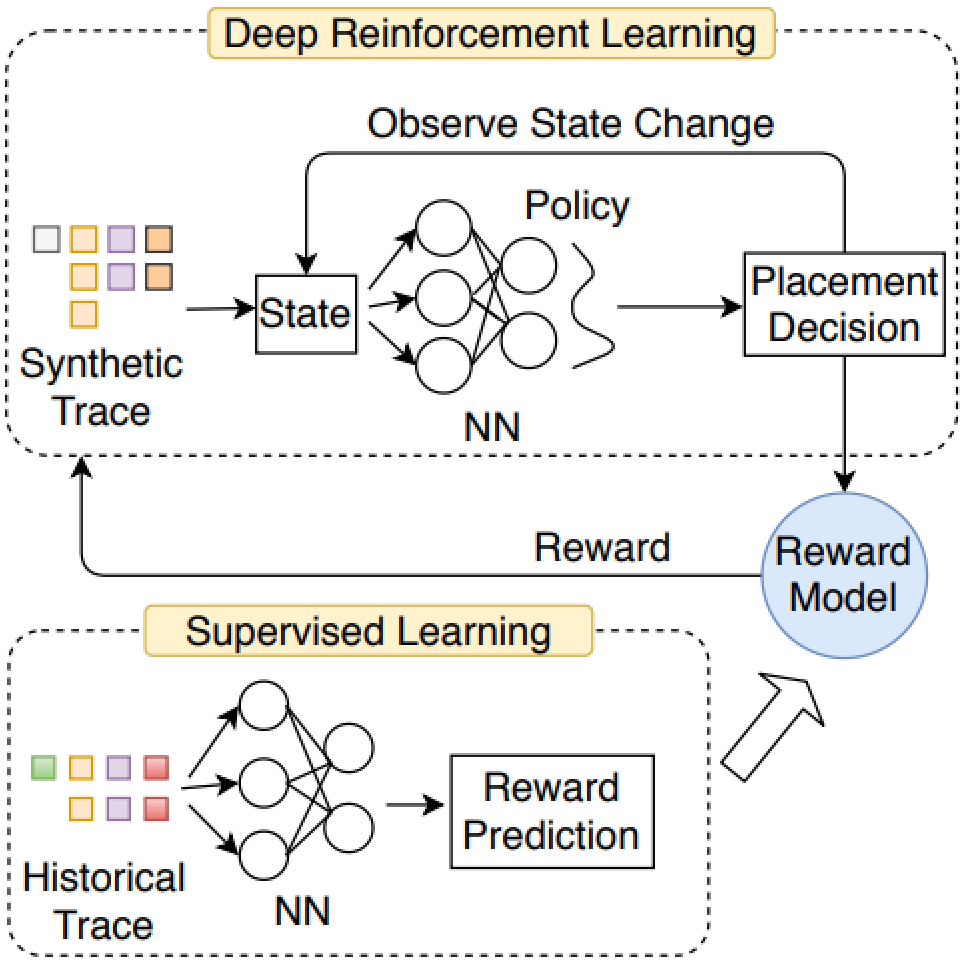
- Two steps

➢ Reward model training

With historical job traces, *Harmony* trains the reward prediction NN using supervised learning. The input includes job set information and placement state; the label is the reward (training speed) of each job.

➢ DRL model training

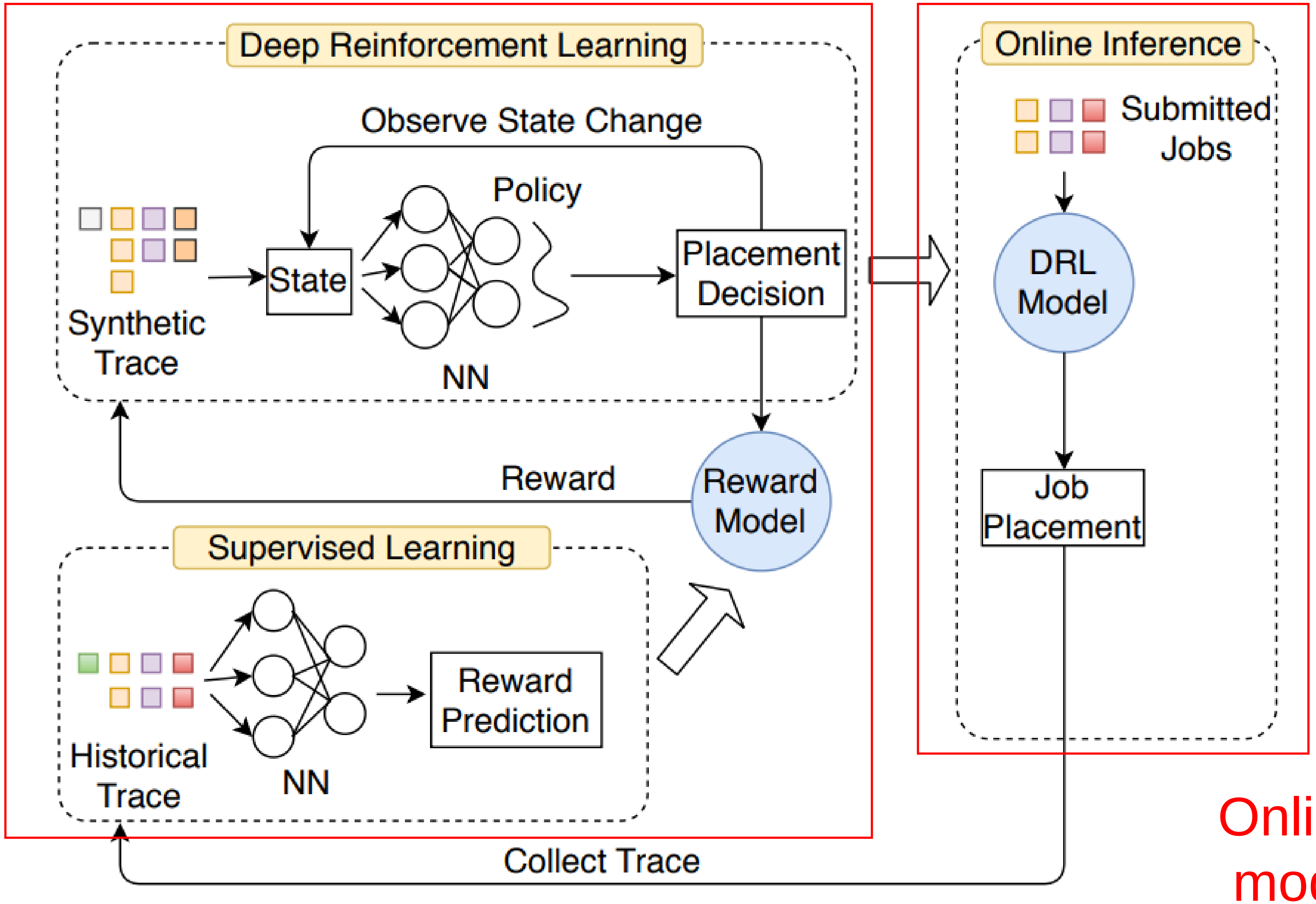
The DRL NN takes various job sets and cluster resource availability as input, and produces placement decisions for new jobs in the set



Offline Training

System Overview

Offline training



Online inference & model update

Harmony workflow

DRL Based Placement Policy

- State space

The input state is $s = (x, r, \vec{w}, \vec{p}, v, d)$

➢ x

An $N \times L$ binary matrix encoding the ML models trained by the jobs.

N: the maximal number of concurrent jobs in an interval

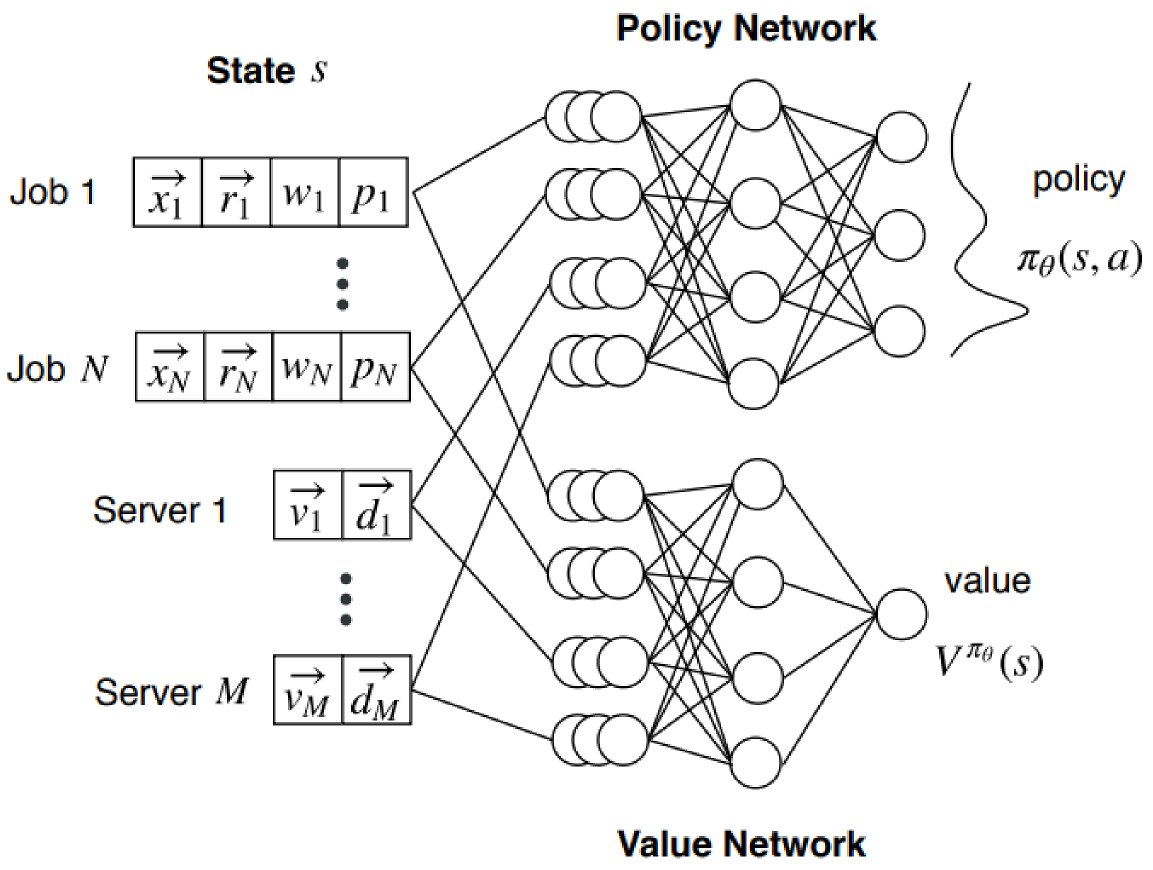
L: is the maximal number of models that can be trained in the cluster

➢ r

An $N \times 2(1 + K)$ matrix encoding worker/parameter server (PS) resource demands in the jobs. K: the number of resource types to compose a worker or a PS.

➢ $(\vec{w})\vec{p}$

An N-dimensional vector, in which the nth item is the number of workers (PSs) allocated to the nth job



DRL framework

DRL Based Placement Policy

- State space

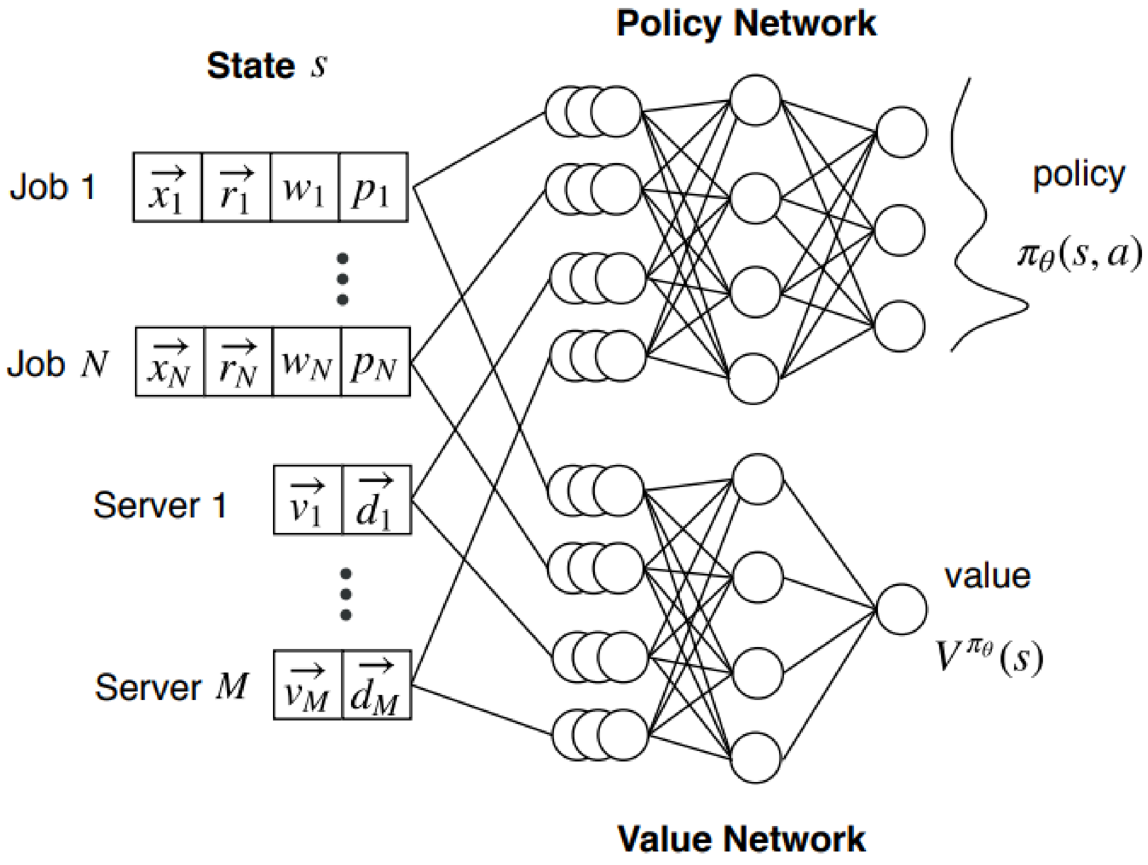
The input state is $s = (x, r, \vec{w}, \vec{p}, v, d)$

➢ v

An $M \times K$ matrix representing available amount of each type of resources on the servers. *M*: the number of physical servers.

➢ d

an $M \times 2N$ matrix encoding the placement of workers and PSs of the concurrent jobs on the servers.



DRL framework

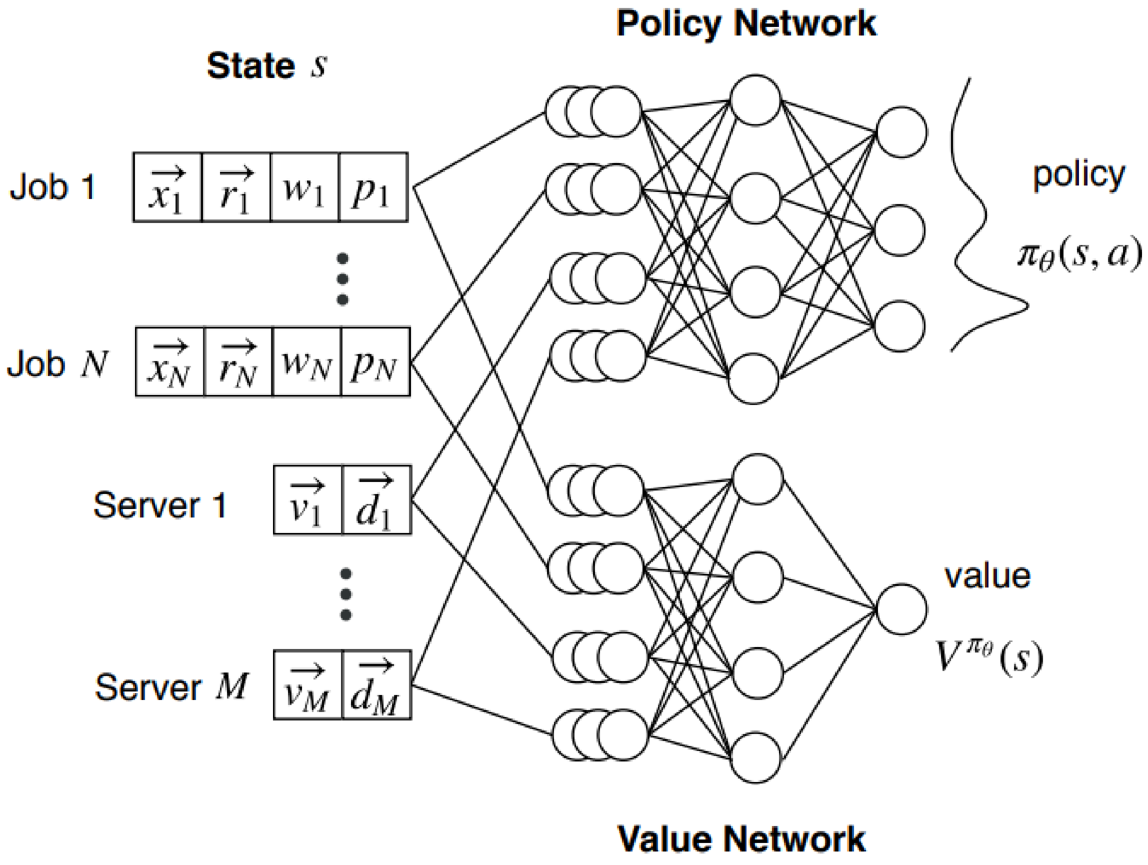
DRL Based Placement Policy

- Action space

After receiving s, the DRL agent selects an action a based on a policy $\pi(s, a)$ - a probability distribution over the action space.

- Reward

The reward r observed when action a is taken under state s is the sum of normalized training speeds of all concurrent jobs in the scheduling interval.



DRL framework

DRL Based Placement Policy

- Actor-critic
  - ➢ Ensure a much lower variance in the estimation of the policy gradient, such that policy learning is more stable.
- Exploration
  - ➢ Ensure that the action space is adequately explored; otherwise, DRL may well converge to poor local optimal policy.
- Experience replay
  - ➢ FIFO replay buffer with a fixed size
  - ➢ Avoid correlation in the sample sequence

Performance Evaluation

- Implementation
  - ➢ Testbed

A testbed with 6 GPU servers on docker containers connected by a Dell Networking Z9100-ON 100GbE switch. Each server has one 8-core Intel E5-1660 CPU, two GTX 1080Ti GPUs, 48GB RAM, one MCX413A-GCAT 50GbE NIC, one 480GB SSD and one 4TB HDD.

➢ Workloads

Randomly select a job from the 6 jobs and set its required number of workers and PSs randomly in [1, 3] to generate a job variant.

➢ Baselines

– Load Balancing (LB).

– Tetris: it uses multi-resource bin packing to place a worker/PS to avoid resource fragmentation.

– Least Interference First (LIF-Line, LIF-Quad): it builds a linear or non-linear interference model by assuming that task slowdown is a function of CPU and bandwidth usage.

Performance Evaluation

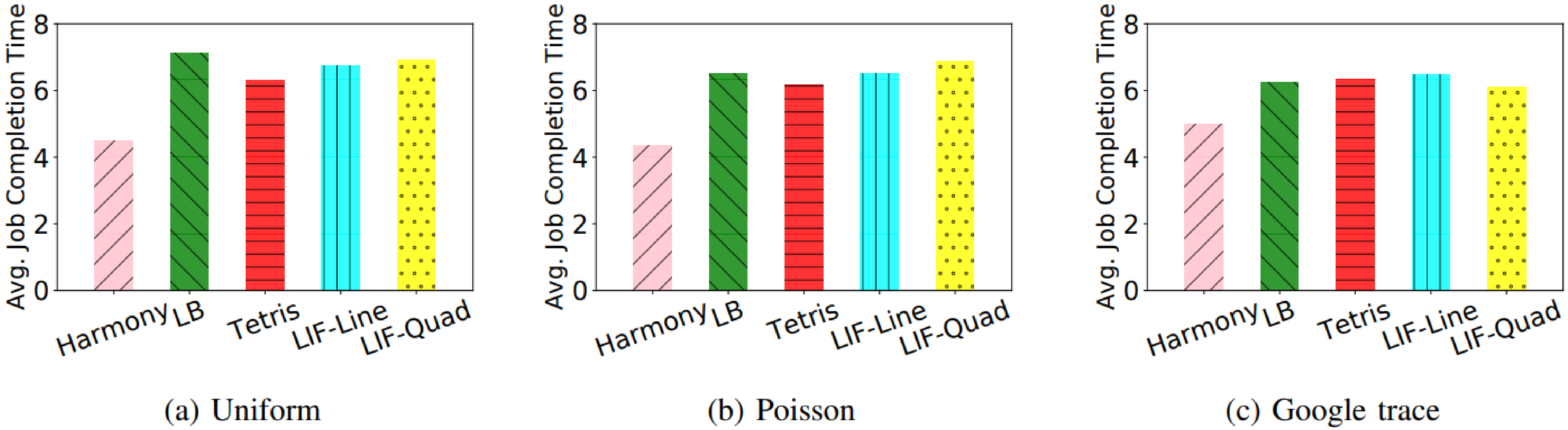
- Performance

➢ Three job arrival patterns

– Default uniform distribution

– A Poisson process with an arrival rate of 2 per scheduling interval.

– The job arrival process extracted from Google cluster traces, with downscaled arrival rates.



Performance comparison under three job arrival patterns

Contributions

- Identify severe performance degradation when sharing resources among ML workloads and propose a general design using DRL to schedule ML workloads.
- Adopt a number of training techniques to resolve issues that may prevent DRL from converging to a good ML job placement policy, including actor-critic algorithm, job-aware action space exploration and experience replay.
- Implement Harmony on Kubernetes and evaluated *Harmony* on a GPU cluster, with real ML jobs. The results show that *Harmony* outperforms commonly adopted scheduling policies by 25% in terms of average job completion time.

Any Questions?

Xingbo Fu
October 14, 2020