UVA CS 6316: Machine Learning : 2019 Fall
Course Project: Deep2Reproduce @
https://github.com/qiyanjun/deep2reproduce/tree/master/2019Fall

CuSH: Cognitive Scheduler for Heterogeneous High Performance Computing System

Citation: Giacomo Domeniconi, Eun Kyung Lee, and Alessandro Morari. 2019. CuSH: Cognitive ScHeduler for Heterogeneous High Performance Computing System. In Proceedings of DRL4KDD 19: Workshop on Deep Reinforcement Learning for Knowledge Discovery (DRL4KDD).

Reproduced by: Vanamala Venkataswamy, Swaroopa Dola

12/06/2019

Motivation

Resource Management is everywhere

Cluster scheduling

Video streaming

Virtual machine placement

Congestion Control

Internet telephony

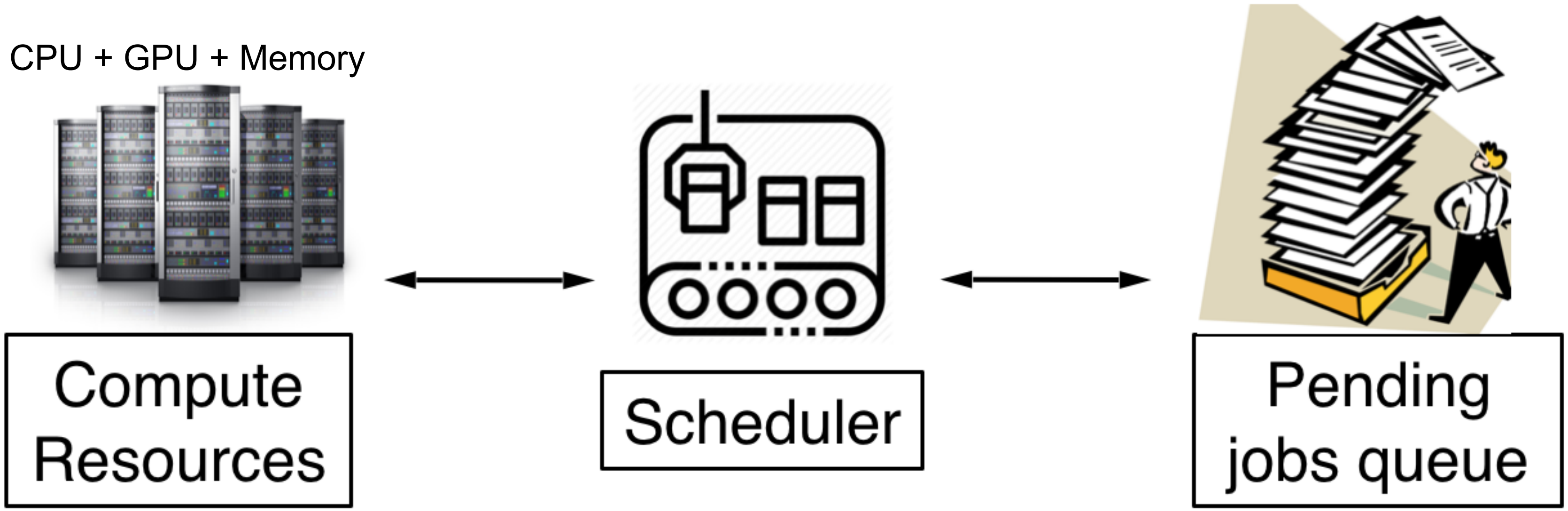
. . . . . .

source: [6]

Motivation

An Example online multi-resource allocation problem, e.g. CPU, memory.

Many factors to consider: NP-hard



source: [6]

Motivation

- Increased complexity of resource management algorithms.

- Resource manager/scheduler should account for data locality, levels of parallelism, frequency of collective synchronizations etc.

- Cost model complexity for a job scheduler
  - job dimensions
  - queue size
  - execution time and available resources
  - job locality - shared datasets

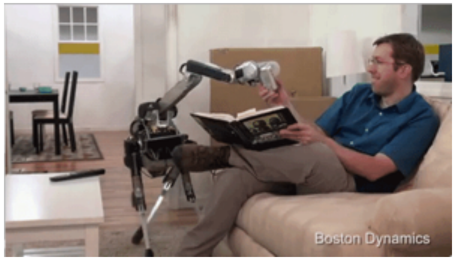
Motivation

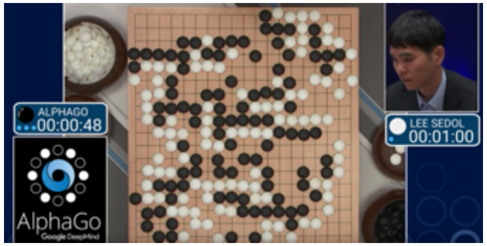
Current approach

- Assume a simple system model
- Come up with a set of heuristics
- Iteratively test and tune the heuristics in real system

Alternative approach

- Reinforcement Learning: Learning via interacting with the environment.



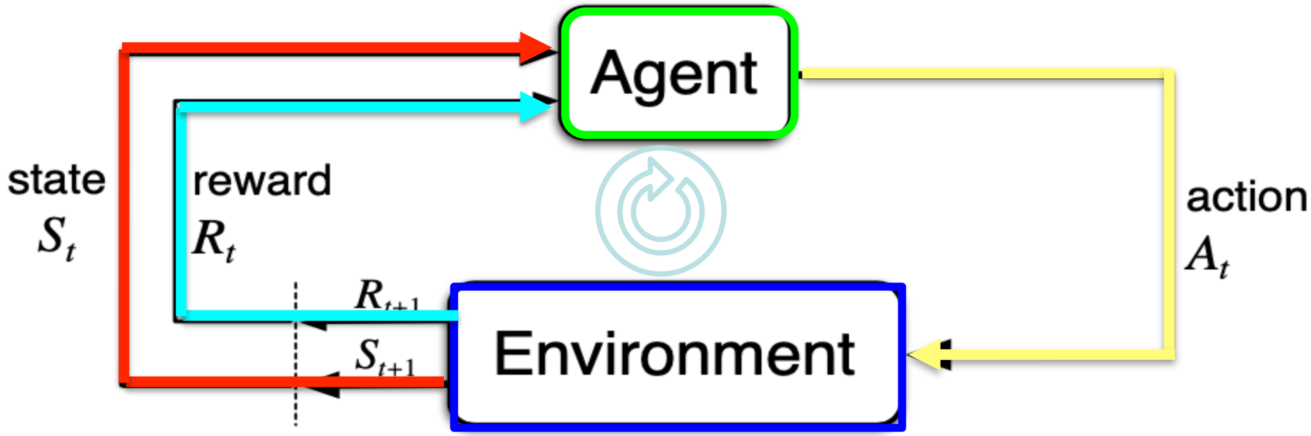


Background - Schedulers

- Job placement policies
  - First Come First Serve (FCFS)
  - Shortest Job First (SJF)
  - Dominant Resource Fairness (DRF)
  - Least Attained Service (LAS)
  - more...

- Traditional Heuristics based schedulers use a combination of these policies to maximize/minimize some objective function.

Background - Reinforcement Learning



Sutton and Barto: The agent-environment interaction in reinforcement learning.[5]

In RL, the data is not **Independent and Identically Distributed**. The outcome depends on the previous state(s) and action(s).

Reward Hypothesis: All goals can be described as maximising expected cumulative reward.

Related Work

- **DeepRM**: Uses RL for cluster scheduling by modeling the cluster state using image-like representation. (2016)
- **Gandiva**: Utilizes domain-specific knowledge to improve latency and efficiency of training DL models in a GPU cluster. (2018)
- **Decima**: Uses RL for scheduling job in Tensorflow like framework. Decima heavily focuses on DL jobs that have DAG like dependencies, optimizing for placing DAG tasks on the cluster. (2019)

Claim / Target Task

- **Claim**: Reinforcement Learning agent can learn better scheduling policies for given cluster constraints than heuristics based schedulers.
- **CuSH**
  - Employs DNN and Reinforcement Learning to achieve optimal performance.
  - Learns to make better scheduling choices by training on a dataset that contains jobs history, available resources and performance characteristics.

An Intuitive Figure Showing WHY Claim

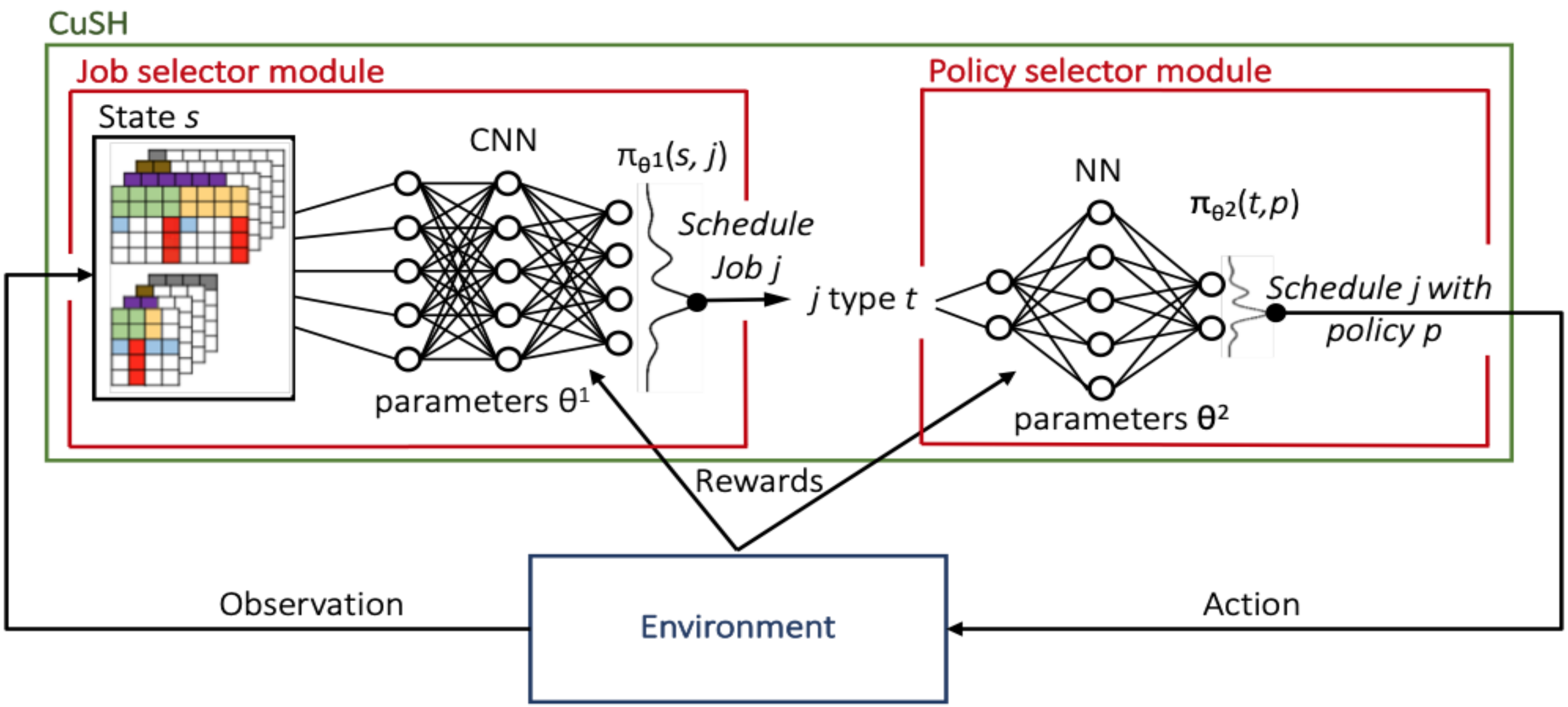


Figure 1: CuSH hierarchical agents trained via RL process.

Reference: CuSH paper

Proposed Solution

- Scheduler as two level system i.e two separated DNNs for job selection and policy allocation.

- The reward function dynamically adjusts based on application.

- RL environment as cluster with
  - N - no. of jobs
  - R - resources
  - Sr - Fixed no.of resource per node
  - Q - jobs to be scheduled

Proposed Solution

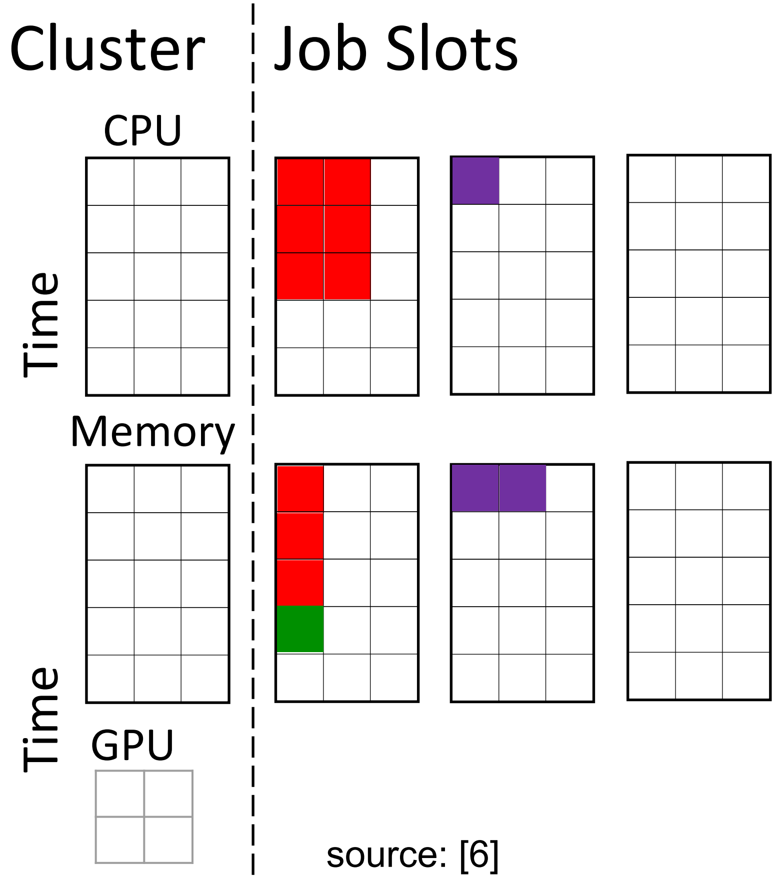
- Two allocation policies
  - Depth-first policy: assign requested resources utilizing as few nodes as possible.
  - Breadth-first policy: assign requested resources utilizing as many nodes as possible

- Two different workloads
  - Compute intensive
  - Network intensive

Cluster scheduling problem setting

- Allocate multiple resources
- Resource requests are known
- Non-preemptive jobs

Goal: Minimize averaged normalized turnaround time



source: [6]

Implementation

Two main scheduler modules

1. **Job selector module (JSM)**
2. **Policy selector module (PSM)**

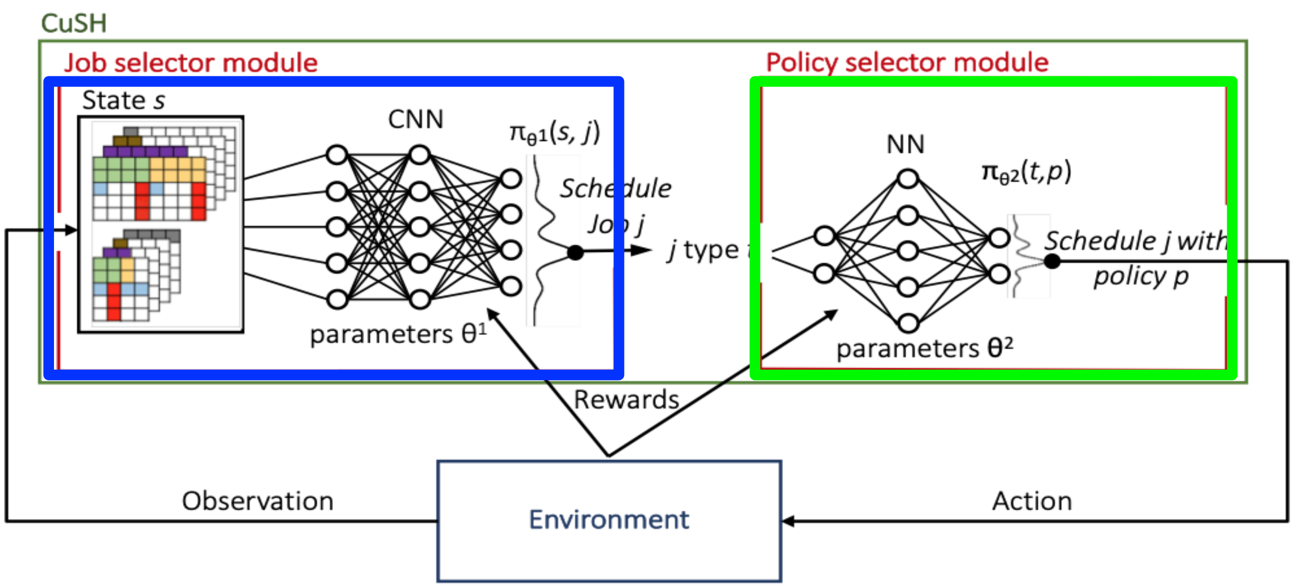


Figure 1: CuSH hierarchical agents trained via RL process.

RL - Observe

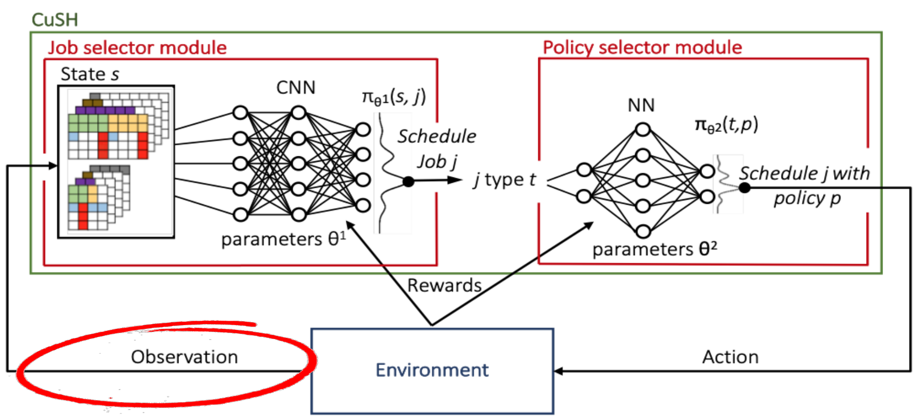
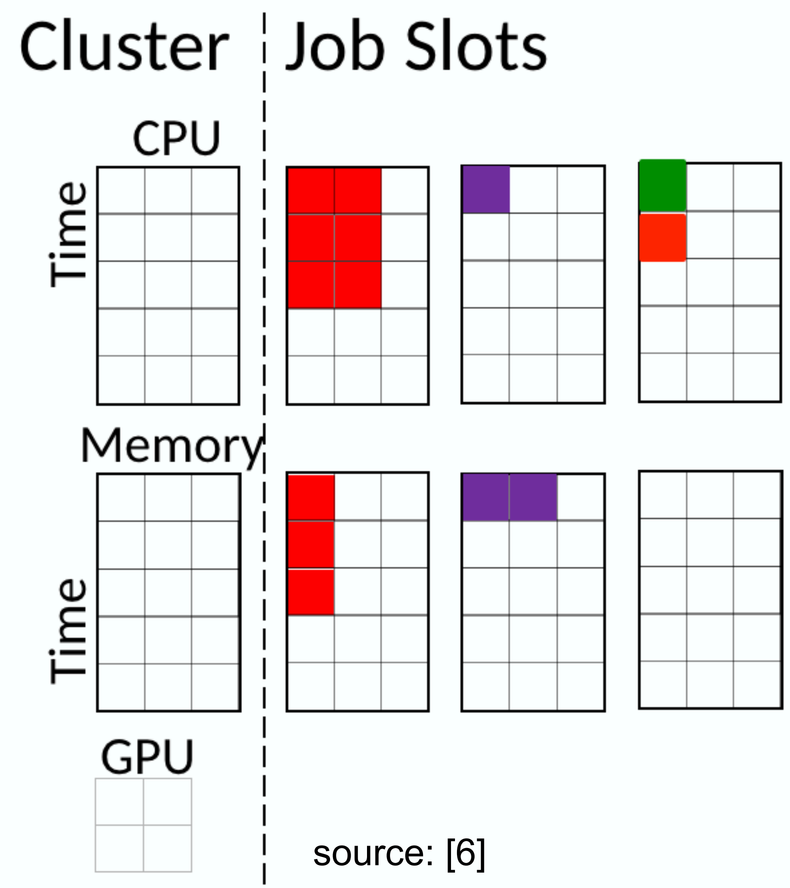


Figure 1: CuSH hierarchical agents trained via RL process.



source: [6]

RL - Take action

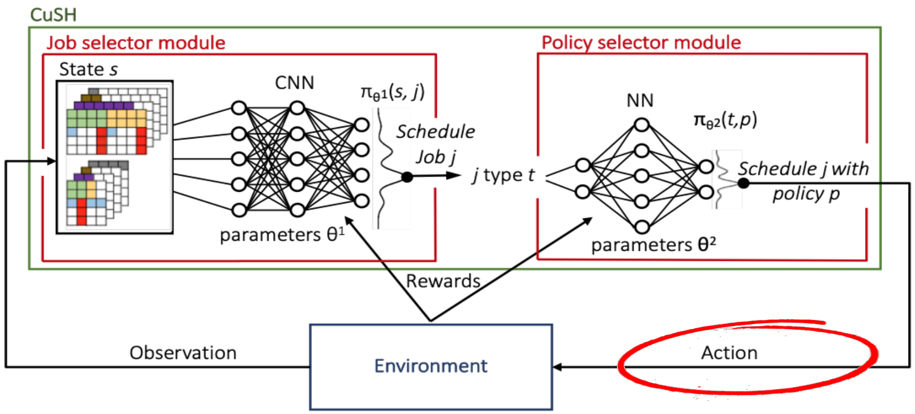
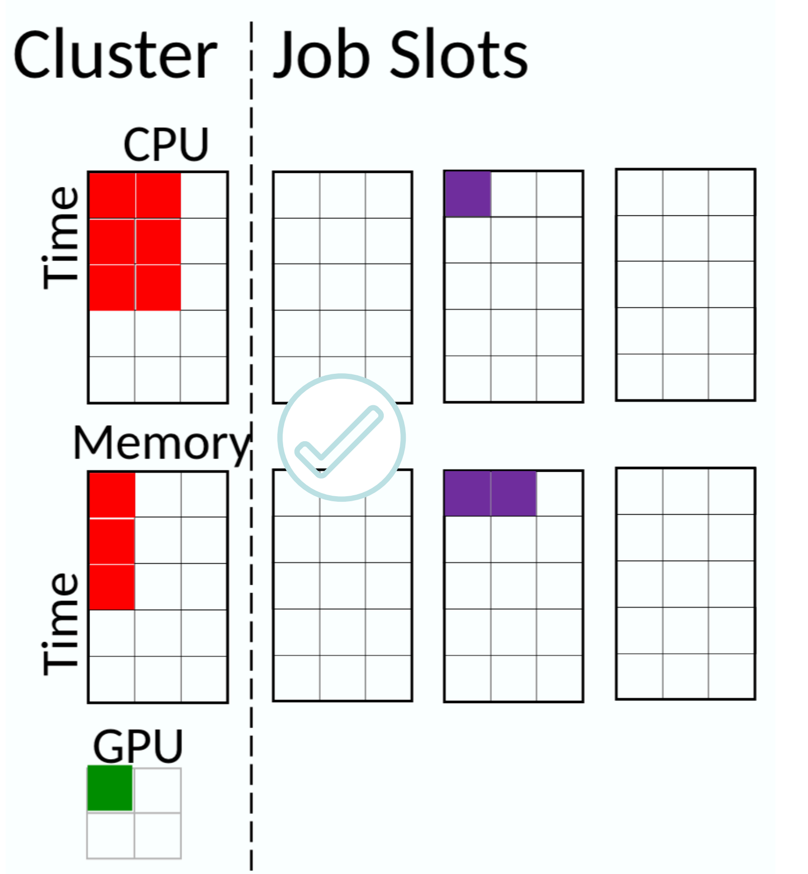


Figure 1: CuSH hierarchical agents trained via RL process.



Naively considering all possible state/action pairs will be exponential cost.

Solution: Sequential allocation

RL - Take action

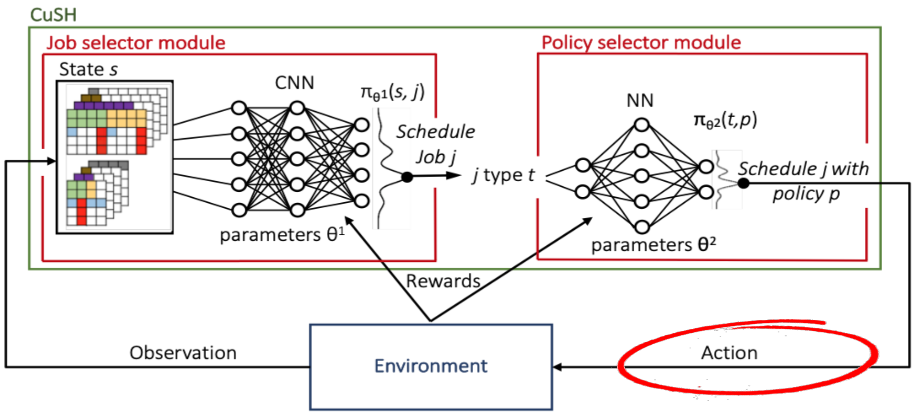
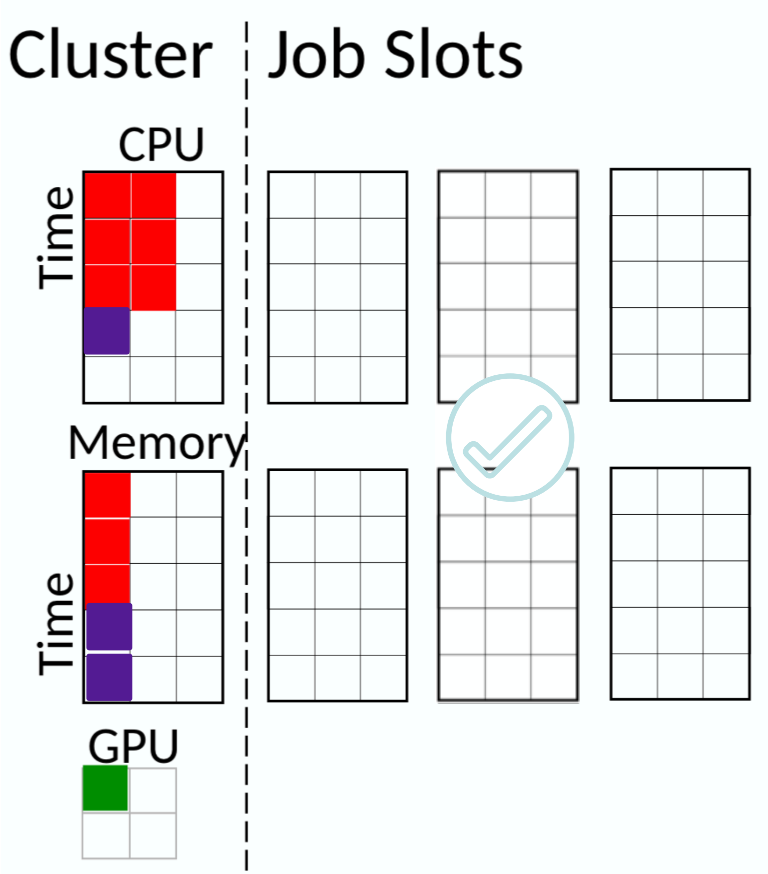


Figure 1: CuSH hierarchical agents trained via RL process.

Naively considering all possible state/action pairs will be exponential cost.

Solution: Sequential allocation



RL - Take action

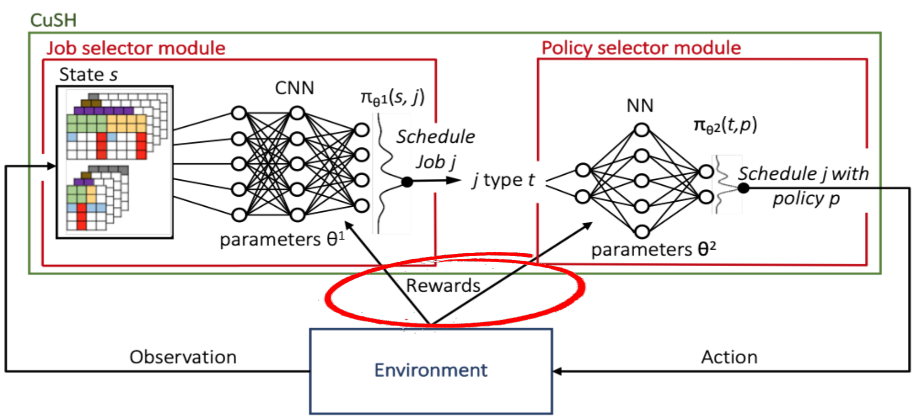
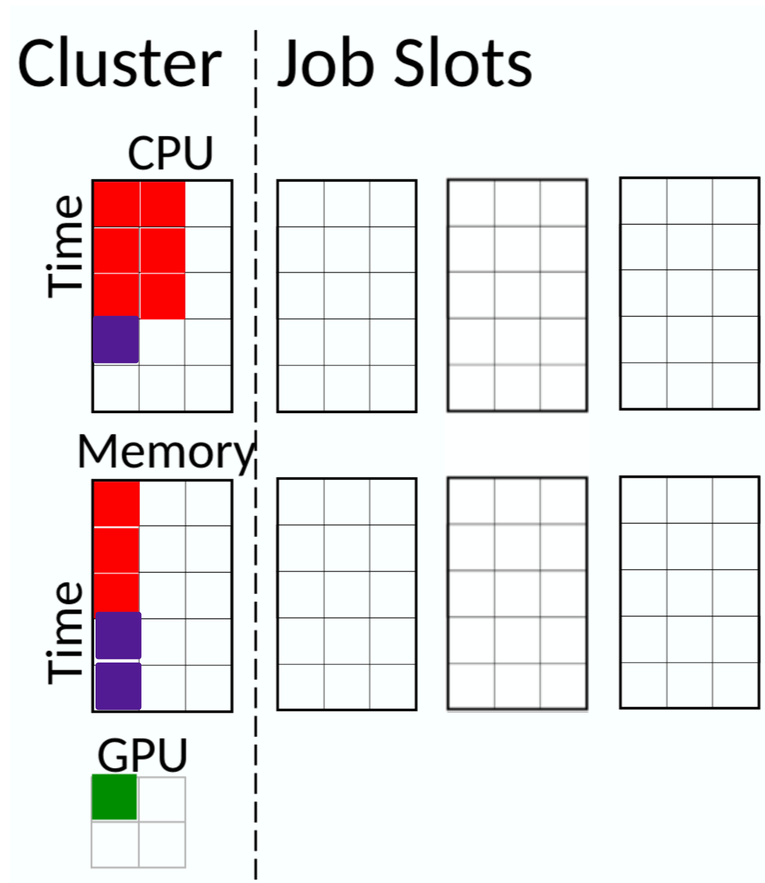


Figure 1: CuSH hierarchical agents trained via RL process.



Get Reward:

A penalty of (*-1/job_len)* for every unfinished job

Implementation

Algorithm 1 Training process

for each epoch e **do** in parallel:
  if $e \mod K = 0$ **then**
    $\theta^1 \leftarrow \text{allreduce}(\theta^1, AVG)$ *//synchronize JSM*
    $\theta^2 \leftarrow \text{allreduce}(\theta^2, AVG)$ *//synchronize PSM*
  $\Delta\theta^1 \leftarrow 0$ *//gradient reset*
  $\Delta\theta^2 \leftarrow 0$ *//gradient reset*
  for each jobset js **do**
    for each Monte Carlo simulation s **do**
      $\{s_i^1, a_i^1, r_i^1, s_i^2, a_i^2, r_i^2\}, r_g \leftarrow \text{dosimulation}(js)$ *//run simulation, each i is one of the L steps of the simulation.*
      $v_t^1 \leftarrow (\sum_{i=t}^{L} \gamma^{i-t} r_t^1) r_g$ *//calculate JSM returns*
      $v_t^2 \leftarrow r_t^2$ *//calculate PSM returns*
    for each step in simulation i **do**
      $b_t \leftarrow \frac{1}{L} \sum_{i=1}^{L} v_t^i$ *//baseline*
      $b_t \leftarrow \text{allreduce}(b_t, AVG)$ *//synchronize baseline*
      $\Delta\theta^1 \leftarrow \Delta\theta^1 + \alpha^1 \sum_t \nabla_{\theta^1} \log \pi_{\theta^1}(s_t^1, a_t^1)(v_t^1 - b_t^1)$
      $\Delta\theta^2 \leftarrow \Delta\theta^2 + \alpha^2 \sum_t \nabla_{\theta^2} \log \pi_{\theta^2}(s_t^2, a_t^2) v_t^2$
  $\theta^1 \leftarrow \Delta\theta^1$ *//update JSM parameters*
  $\theta^2 \leftarrow \Delta\theta^2$ *//update PSM parameters*

source: cuSH paper

The $\theta 1$ are network parameters of the JSM and $\theta 2$ parameters of the policy selector.

Then $\pi\theta 1$ and $\pi\theta 2$ as the JSM and PSM networks, respectively.

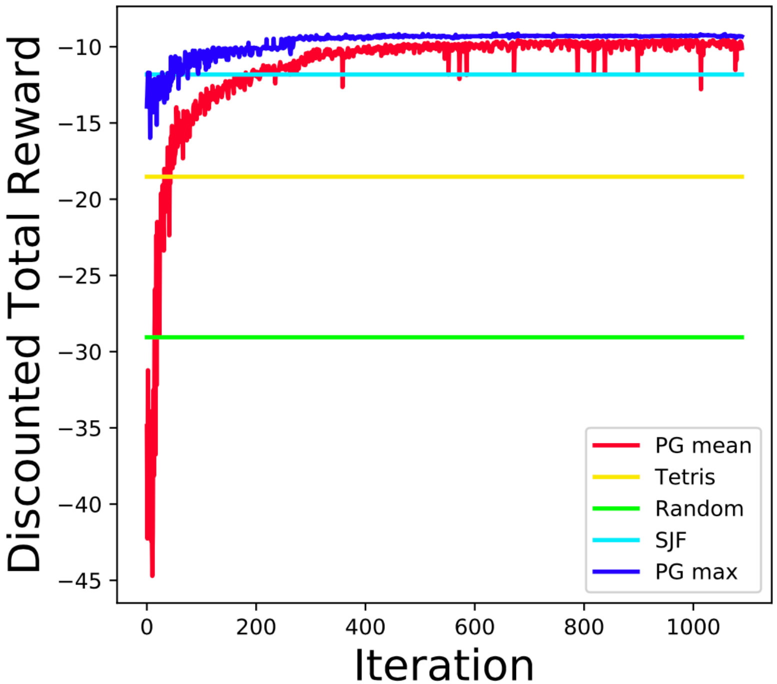
Experimental Strategy

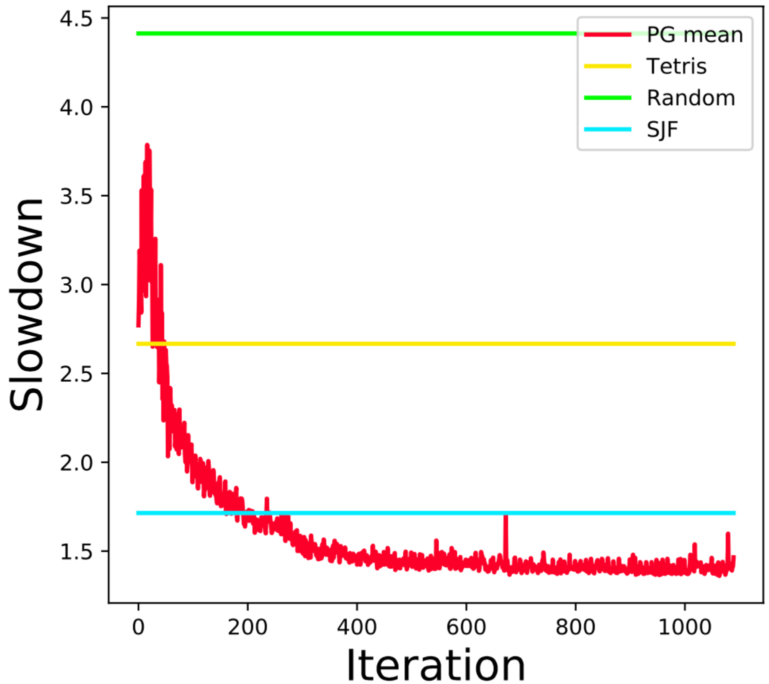
- CuSH: Code not open sourced
- DeepRM [2][6] by MIT is open source. CuSH is based on this work.

CuSH vs. DeepRM

Differences	CuSH	DeepRM
Architecture	JSM and PSM	JSM
Key Metric	averaged normalized turnaround time	average job slowdown
Input format	Wait time for jobs in queue	Binary matrices
Job duration	Bounded	Unbounded
Resource Locality	Yes	No
Workload Type	Yes	No

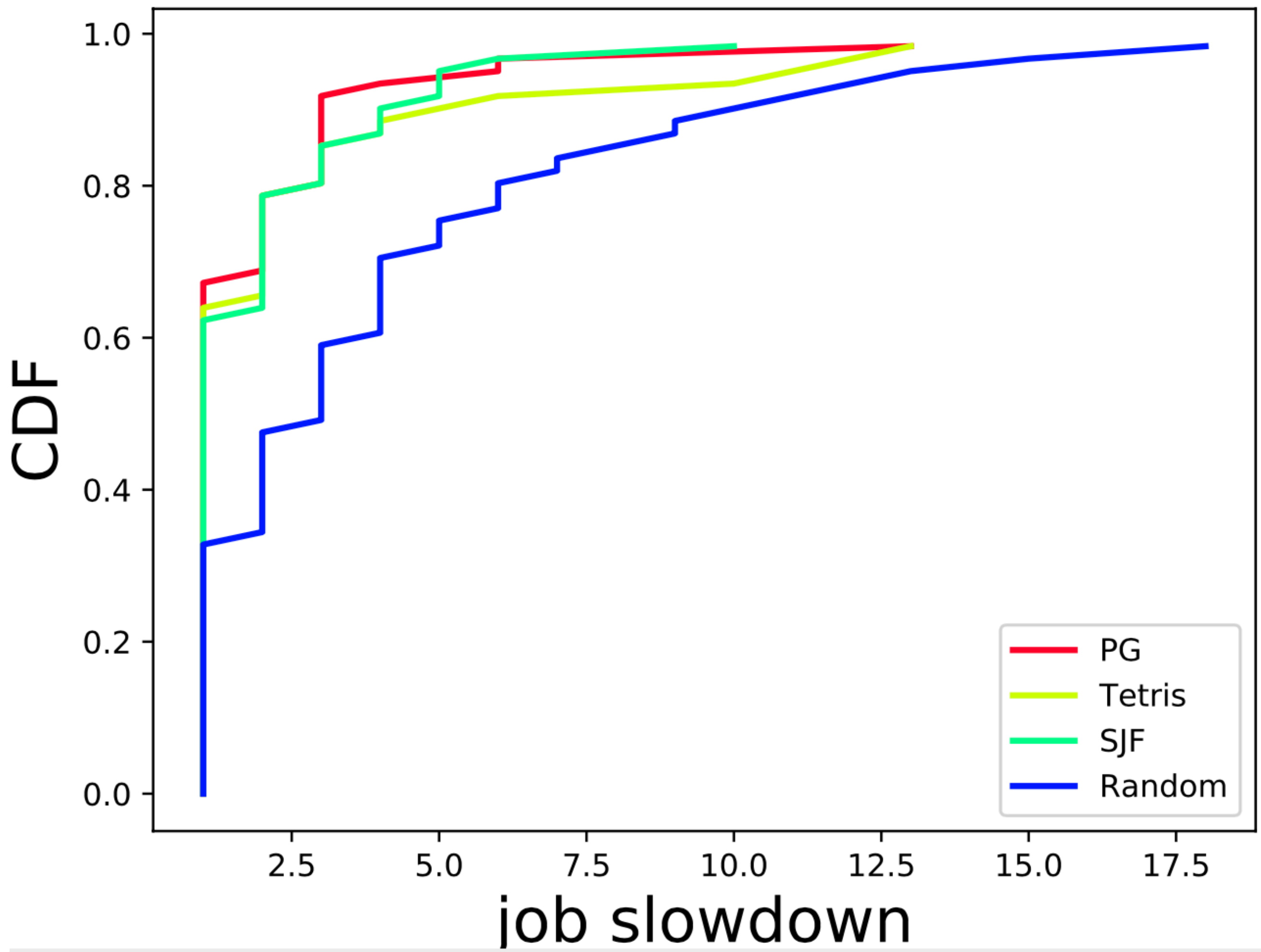
Experimental Results





Graphs generated by presenters, using DeepRM [2] code

Experimental Results



Graphs generated by presenters, using DeepRM [2] code

Conclusion and Future Work

- Resource management using traditional Heuristics based scheduling does not always give best schedule.

- Better job scheduling subject to constraints can be achieved using DNN and RL.

- CuSH - an RL implementation outperforms the best heuristic-based approaches, delivering up to 19% lower normalized turnaround time.

Work distribution

Paper selection	Vanamala, Prof. Qi
Paper review slides	Equal contribution
Experiments	Individually ran DeepRM code and generated results/graphs
Slides with results	Equal contribution

Questions?

Thank You for your attention!

References

[1] Learning Scheduling Algorithms for Data Processing Clusters. Hongzi Mao, Malte Schwarzkopf, Shaileshh Bojja Venkatakrishnan, Zili Meng, Mohammad Alizadeh. SIGCOMM-2019.

[2] Resource Management with Deep Reinforcement Learning. Hongzi Mao, Mohammad Alizadeh, Ishai Menache, Srikanth Kandula. HotNets-2016.

[3] Gandiva: Introspective Cluster Scheduling for Deep Learning. Wencong Xiao et. al. OSDI-2018.

[4] High-Performance Job-Shop Scheduling With A Time-Delay TD(lambda) Network. Zhang, W., Dietterich, T. G., (Advances in Neural Information Processing Systems. 1996.

[5] Richard S. Sutton and Andrew G. Barto. Reinforcement Learning: An Introduction. 2nd Edition. 2015.

[6] https://github.com/hongzimao/deeprm

Backup Slides

Data Summary

Input for the NN

- Merge cluster and queue into one matrix representation.
- The cluster nodes are concatenated together along the x-axis, forming R matrices of N x Sr ×T size, which is the same size of the representation of the waiting jobs.
- The input of the job scheduler module is a three-dimensional matrix of size (□R N · Sr) × (T) × (Q + 1)

Implementation

1. Job selector module (JSM)

- Current state as input image: jobs in the cluster, waiting jobs, resources.
- a CNN using 16 2x2 filters, stride=1 and without padding followed by a ReLU, batch normalization and a softmax layer to predict probability for each action.

Implementation

2. **Policy selector module (PSM)**

- goal of selecting which policy to use to allocate a job that has to be scheduled.
- The module is trained with policy gradient.
- The value return **vt** is only based on the local action and its reward value **rt**.
- The return is the locality penalty **(vt =rt =pj)**, that is calculated using the projected workloads data.

Data Summary

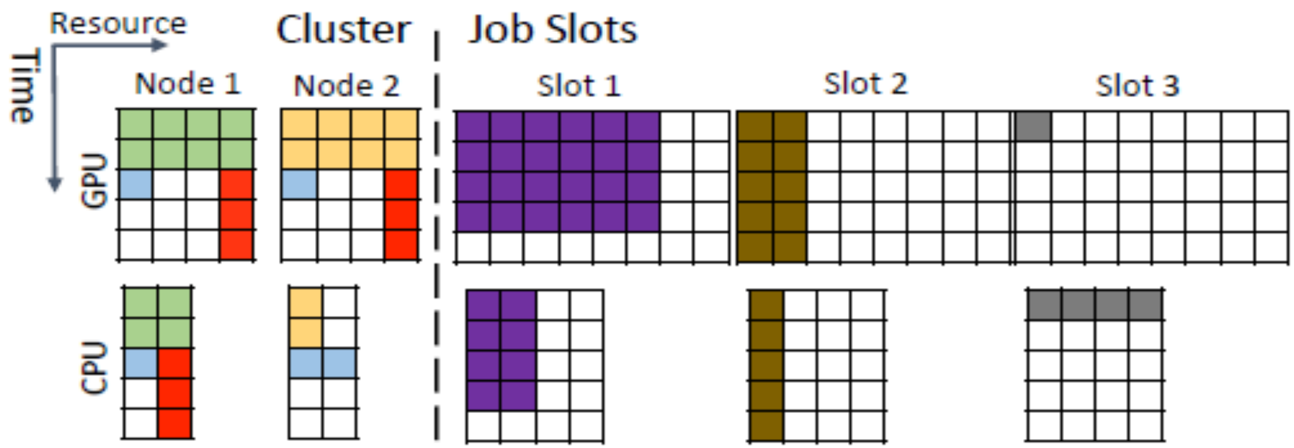


Figure 3: Example of environment (state) representation. This representation shows a cluster of 2 nodes (each one with 2 CPUs and 4 GPUs) and a queue (job slots) of 3 waiting jobs.

source: cuSH paper

Conclusion and Future Work

- Current model requires workload type to be specified by the user.
- Better approach would be to use dynamic scheduling
  - unspecified job types can be classified as “unknown”
  - After few executions, the job type can be automatically classified with a ML model

DeepRM Architecture

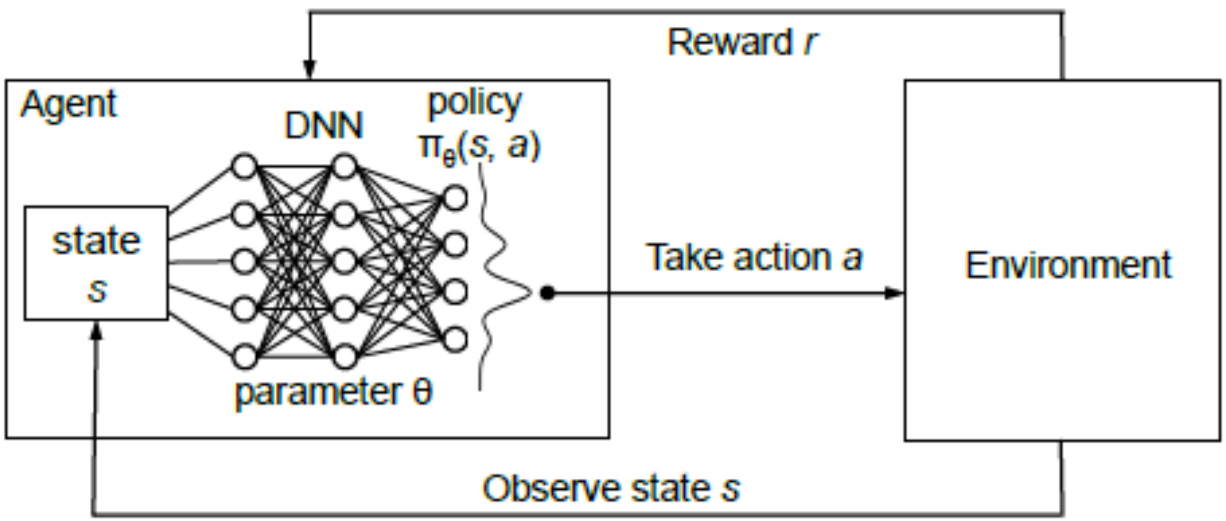


Figure 1: Reinforcement Learning with policy represented via DNN.

Source: DeepRM [2] paper