

# Scala Job Scheduler

March 2018

## **Abstract**

We design a novel scheduling architecture PANDA (Policy Advisor Network and Decision Architecture) using multi-agent reinforcement learning model, specially for cloud environment with dynamically allocated computing resources and inhomogeneous requirements from consumers. This architecture consists of policy advisor with Bayesian hierarchical clustering, catch-release-wait agent action mechanism and reinforcement training, which lead to its many advantages over traditional schedulers.

# Contents

<b>Introduction</b>	<b>1</b>
<b>1 Previous Work</b>	<b>2</b>
<b>2 Policy Advisor Network and Decision Architecture (PANDA)</b>	<b>2</b>
2.1 Overview . . . . .	2
2.2 Model . . . . .	3
2.2.1 Policy Advisor Network . . . . .	4
2.2.2 Agent Model . . . . .	4
2.2.3 System Supervisor . . . . .	5
<b>3 Training</b>	<b>5</b>
3.1 Pre-Training . . . . .	5
3.2 Model Training . . . . .	6
<b>4 Discussion</b>	<b>6</b>

## Introduction

The aim of this project is to design an optimal scheduling algorithm for a scalable computing cloud, where computing resources are dynamically allocated to meet the diverse demands of an inhomogeneous set of consumers. Resources are not uniformly distributed, geographically or otherwise, as the nodes comprising the cloud are of variable type and processing power. Clients will submit job specifications (indicating the number and type of cores, ideal network topology, arrival time, required run time, memory size, etc.) to the scheduler, which should designate a time to run and a cluster of nodes that adheres to the specification. The algorithm should minimize expected average total time in system for all users, while maintaining fairness between jobs that place similar demands on the system. The algorithm should also be able to achieve optimal scheduling and adapting under rapidly changing cloud environment conditions, while reducing number of accounted metrics for scheduling optimization.

In such a dynamic, diverse system (given the sheer number of factors to account for), traditional static scheduling algorithms, for example, linear programming, can often be nullified by rapidly changing and sometimes unreliable resources. Therefore we elected to confront the problem with a reinforcement learning algorithm built on the aforementioned model. The algorithm is highly dependent on the system's partitioning into measurable (numerically describable) components, and typifying these components for efficient processing - thus the focus on separability and producing quantifiable descriptors of the system. What follows is a detailed specification of the resultant scheduling paradigm: first of the reinforcement model PANDA (Policy Advisor Network and Decision Architecture), and then training of this model and further discussions.

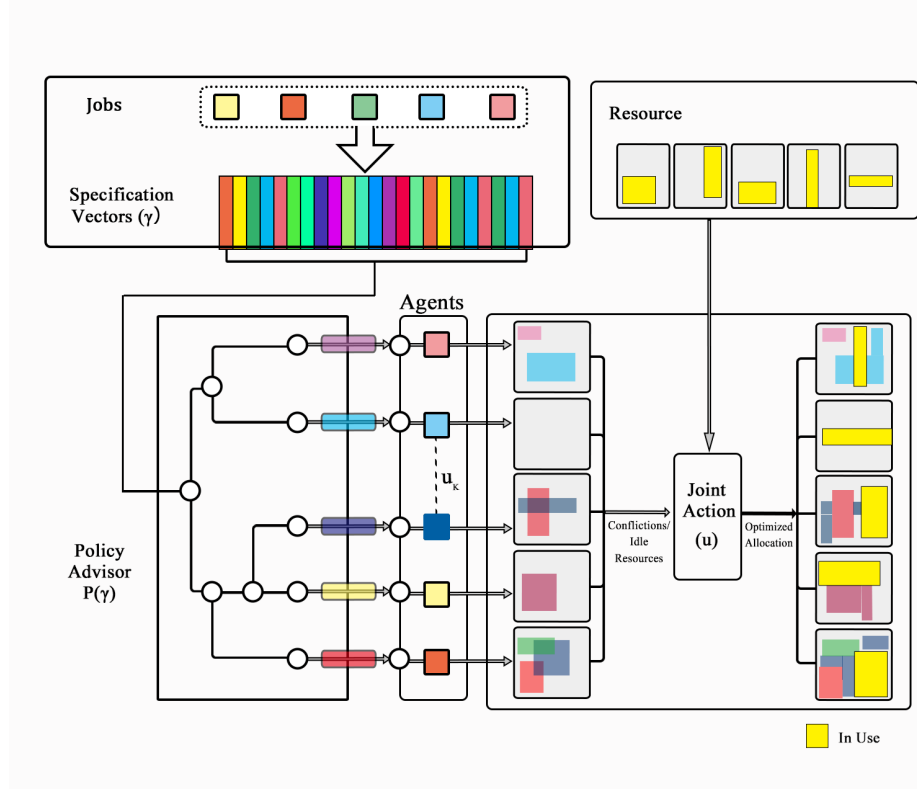
## 1 Previous Work

## 2 Policy Advisor Network and Decision Architecture (PANDA)

This proposed reinforcement architecture can handle both dynamic agent population and state space when processing diverse system. It can also handle static resource space by default. This architecture observes the total time of a user being in system as the only metric for scheduling optimization, by reducing other metrics to be about time in the first place. From this perspective, the PANDA architecture is simpler and more efficient than traditional static schedulers. It can counter greedy scheduling algorithm to a more extensive degree, which in return gives novel insight of general scheduling processes.

### 2.1 Overview

The diagram below shows the complete PANDA model, which represents one time step of the entire algorithm. Each parameter class is represented using a color residing in the box labeled agents. The square box represents a specification parameter class and there are a set of agents that exist within that class at any one time step.



Upon submission by user, a consumer (user) enters the scheduling process and submits its specifications as a parameter vector, which is given to the policy advisor as input. The policy advisor produces a policy parameter vector which are then used as the weights of the actor network. The actor network is then used as the mechanism that the representing scheduling agent will sample from in order to perform actions within the system. In the end, the consumer leaves the scheduling process by starting consuming (running on) assigned resources. Once finished, the consumer gives feedback to the scheduler for further training improvement.

There are three levels of action that are looked at, the agent action, the class action, and the system action. An agent action is produced by the actor network of an scheduling agent, a list of agent actions within the same class is a class action, and a list of class actions submitted to the system is a system action, which is referred to in the diagram as the joint action of all the agents.

## 2.2 Model

Given the nature of the dynamic and diverse system, the rigidity of other traditional scheduling solutions would be a critical problem in the efficiency and complexity of the scheduling. A reinforcement learning approach makes the system adaptable and flexible to changing conditions of the environment, which

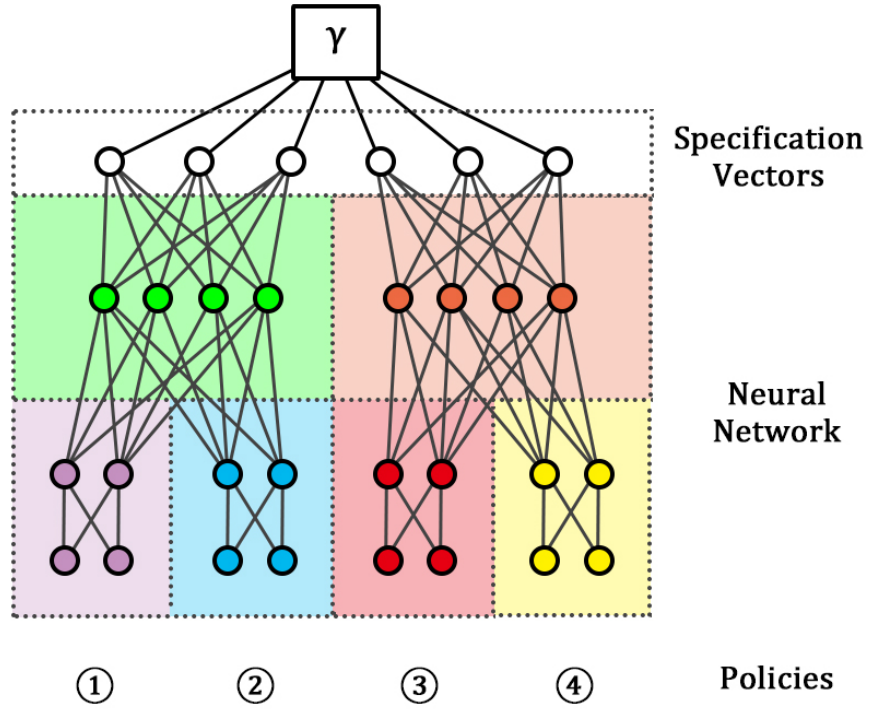
is very desirable. This model hopes to be able to give insight into multi-agent problems, as well as how to correct the relatively common sub-optimal solutions of greedy algorithms. Additionally the model hopes to show how reinforcement learning can be used usefully in combinatorial problem solving.

### 2.2.1 Policy Advisor Network

The policy advisor is the mechanism which consumers utilize to encode their specification into their representative scheduling agents. These agents then use the output given by the policy advisor corresponding to their class in order to parameterize their policy. The policy advisor is defined as follows.

**Definition 2.1.** The *policy advisor* is function  $\mathcal{P} : \Gamma \rightarrow \Theta$ , where  $\Gamma$  is the specification parameter space and  $\Theta$  is the policy parameter space.

**Topology** The diagram below shows a simplified advisor network topology. The colors represent the different classes of specification parameter vectors. The numeric labels are indicative of the specification parameter vectors.



### 2.2.2 Agent Model

#### Partial Observation and Attention Mechanism

**Catch-Release-Wait Mechanism (CRW)** At the end of each time step, agents submit a *joint action* to the system supervisor consisting of a sequence of elementary action units. These action units will be performed in order upon execution of the joint action. The basic action units are as follows:

1. Catch - the agent acquires an available resource to be assigned to its consumer. Caught resources are held until consumed or a *release* action is taken.
2. Release - the agent returns a held resource to the pool of available resources.
3. Wait - the agent delays consumption of resources until the subsequent time step. Wait actions may only occur once per joint action, and only as the last action in the sequence.

### 2.2.3 System Supervisor

Agent decisions are submitted as joint actions to a system supervisor, which resolves collisions between catch requests so as to maximize the net gain of all agents in the system. The gain function will be analytically constructed to provide functionality in the short-term, but will be replaced with a trained version once the reinforcement model has converged sufficiently for effective resolution. The supervisor then initiates consumption of collected resources, excluding those held by an agent whose joint action concluded with a wait operation.

The supervisor also tracks the progression of each specification through the system, and distributes reward to the respective agents according to a function of the collected data and potential consumer feedback. In the instance of the scalable cloud, reward will be computed upon completion of the job, at which point the total time spent in the system (the sum of wait time and run time) and any consumer feedback will be available and may be factored into the calculation. To maximize fairness, longer wait times would be permissible for jobs that placed larger demands on the system (in cores or core hours). Total time is the main factor to consider, as minimizing wait time will maximize system utilization, while reduced run time is a result of optimized resource selection. Both times would be normalized relative to expected values derived from the specification parameters and the state of the system.

## 3 Training

### 3.1 Pre-Training

Using Bayesian hierarchical clustering, consumers are classified into a certain consumer type represented by a numeric label (e.g. 1, 2). After classification, the consumer then enters their specification parameters and consumer type into

a policy advisor which then outputs policy parameters for a scheduling agent to use, for the duration of their search.

### **3.2 Model Training**

## **4 Discussion**

### **Hybrid Approach**



# Notation

$\gamma$	specification parameter
$\Gamma$	specification parameter space
$C$	a subset of consumers
$v$	a service
$V$	Set of services
$\rho(X)$	requirement function
$\mathbf{Req}_\rho(X)$	requirement operator with parameter set X
$\wedge$	and (joint)
$\cap$	intersection
$\psi(X)$	inspection function
$\phi$	acceptance requirement
$a$	an arriving process
$A, G$	assignment actions
$g$	an assignment process
$d$	a departure process
$D$	a departure action
$G$	a graph with vertices V, and Edges E
$\psi(X)$	inspection function
$\mathbf{Path}_G^{(n)}$	a path in graph G of length n
$\sqcup$	disjoint union
$\mathbf{Cycle}_G(v)$	cycle in graph G from vertex v to vertex v
$\mathcal{P}(\gamma)$	policy advisor