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

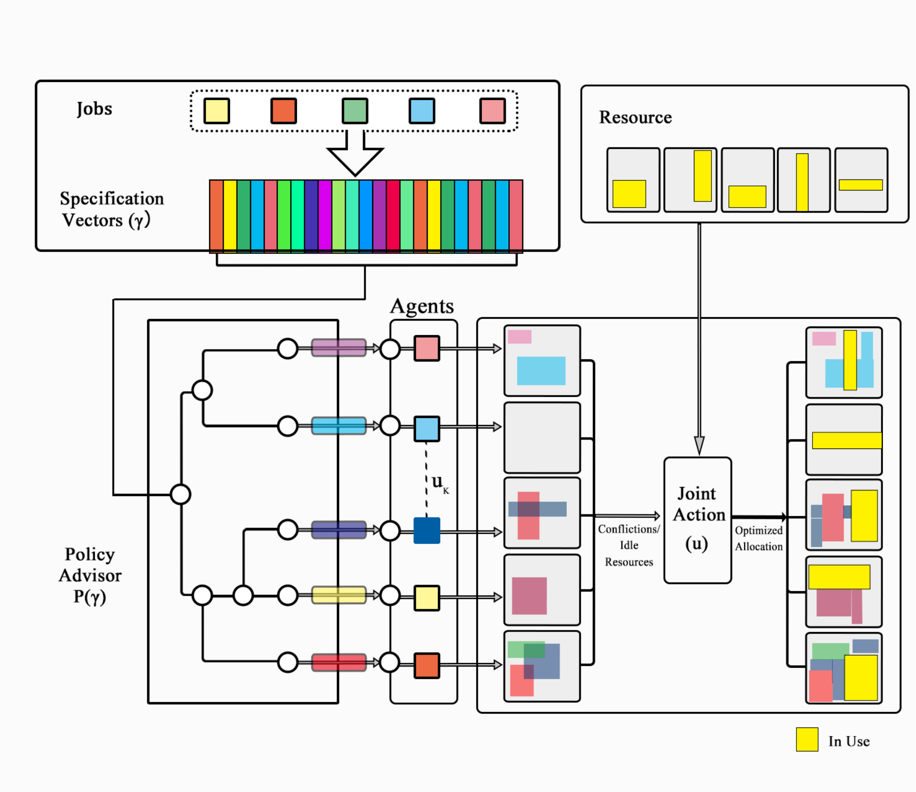
Introduction

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, required run time, 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.

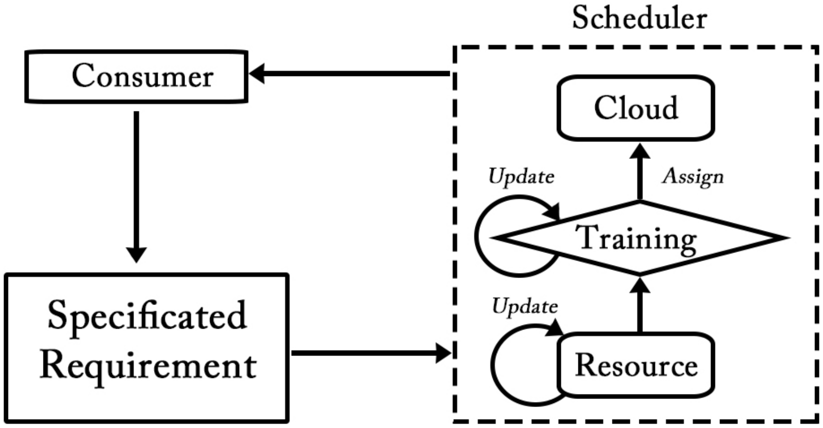
1 Previous Work

2 Policy Advisor Network and Decision Architecture (PANDA)



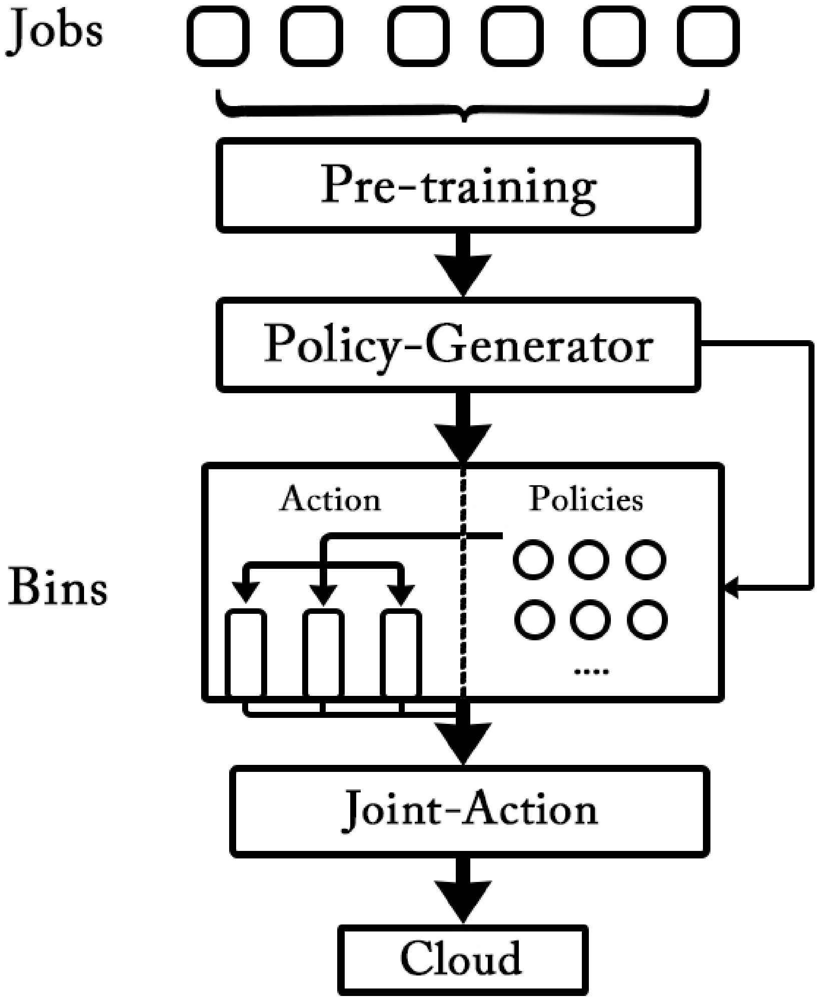
Upon submission by user, a consumer (job) enters the scheduling process with its specifications. It leaves the scheduling process by starting consuming (running on) assigned resources. Once finished, the consumer gives feedback to the scheduler for further training improvement.

Example: a job is submitted with specifications such as arrival time, maximum run time, required number of nodes or cores, memory size, CPU/GPU, etc. After it finishes running, it gives feedback, such as wait time, actual run time, user reward, etc, to the scheduler.

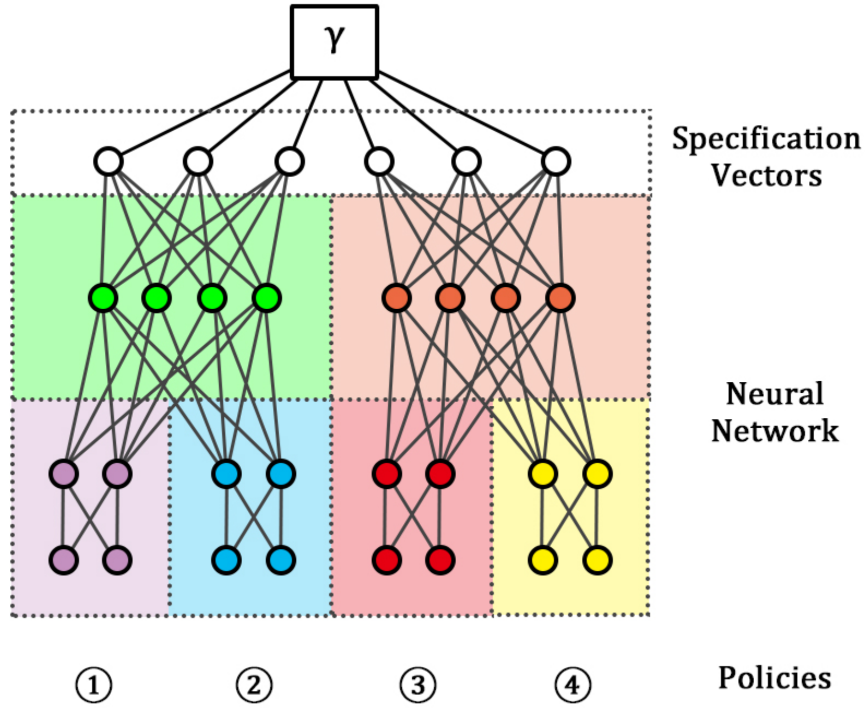


The scheduler first takes in the consumer specifications and inspects the current resource space. Then both the specifications and resource states are collected by the training module. During the reinforcement training, the training module and resource space are constantly being updated, while resources are being assigned or released and consumers entering or leaving the scheduling process.

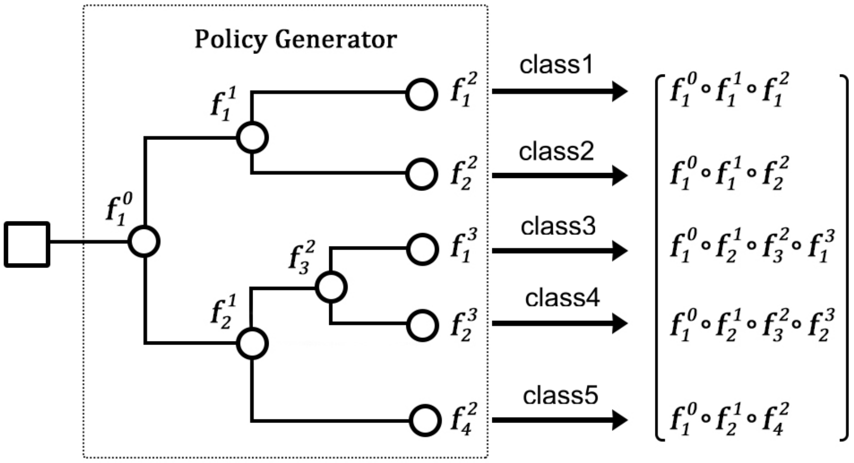
a. Main Training Process



b. Neural Network



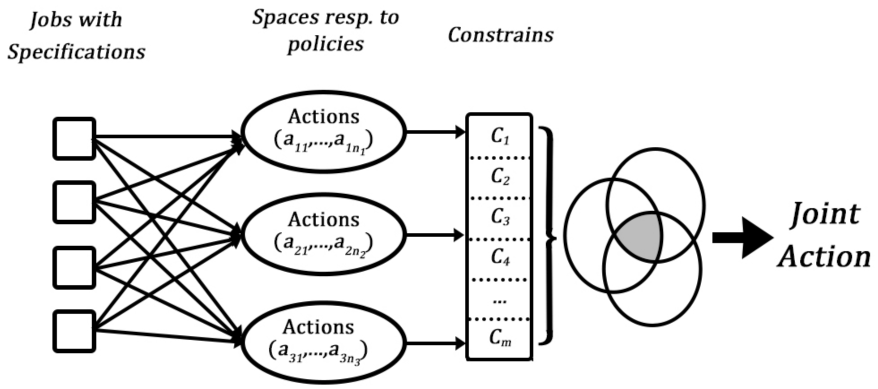
c. Policy Generator


Policy Generator
f_1^0
f_1^1
f_2^1
f_3^2
f_1^2
f_2^2
f_1^3
f_2^3
f_4^2
class1
class2
class3
class4
class5
$f_1^0 \circ f_1^1 \circ f_1^2$
$f_1^0 \circ f_1^1 \circ f_2^2$
$f_1^0 \circ f_2^1 \circ f_3^2 \circ f_1^3$
$f_1^0 \circ f_2^1 \circ f_3^2 \circ f_2^3$
$f_1^0 \circ f_2^1 \circ f_4^2$

Bayesian Hierarchical Clustering

d. Bins

e. Joint Actions



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.

Part I
Reinforcement Model

1 Data Collection

The data collected on consumers is as follows:

2 Consumer Classification

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 Policy Advisor

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

4 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.

Notation

γ	specification parameter
Γ	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
ϕ	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