Scala Job Scheduler

March 2018

Abstract

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

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 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 capable of adapting to and achieving optimal scheduling in highly variable cloud environments, while reducing the 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 such as 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 specifically adapted to this and similar environments. The algorithm is highly dependent on the system's partitioning into measurable (numerically describable) components, and typifying these components for efficient processing - thus we focused 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.

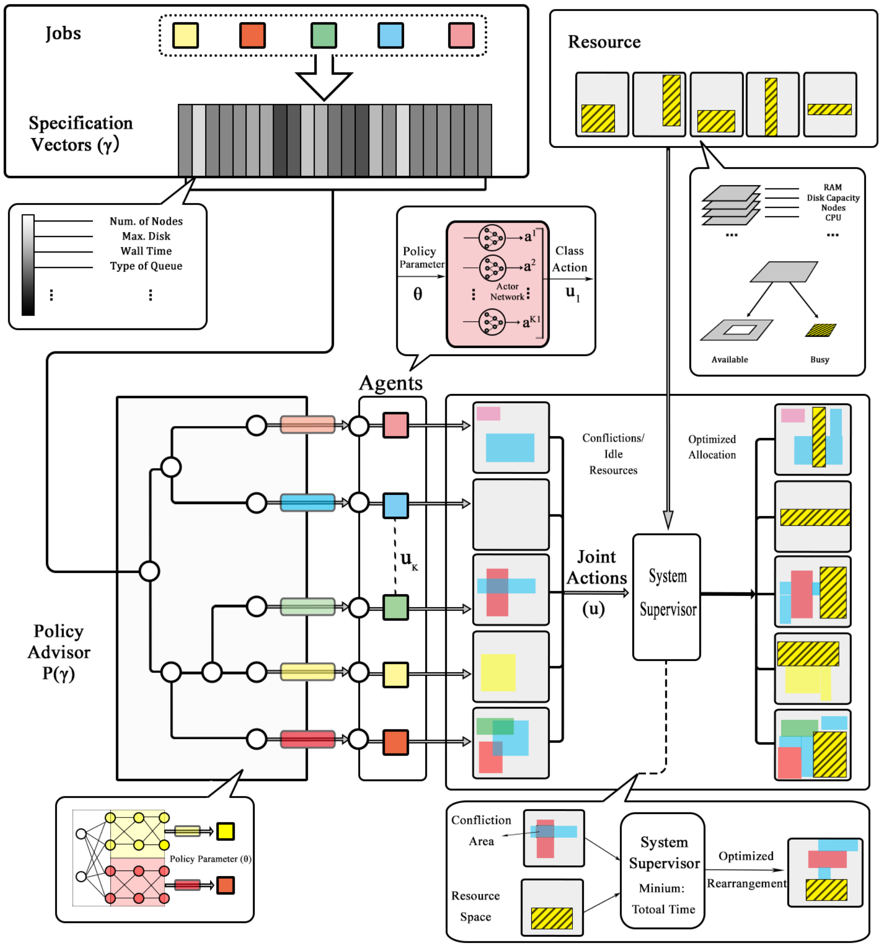
1 Previous Work

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 rigidness 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.

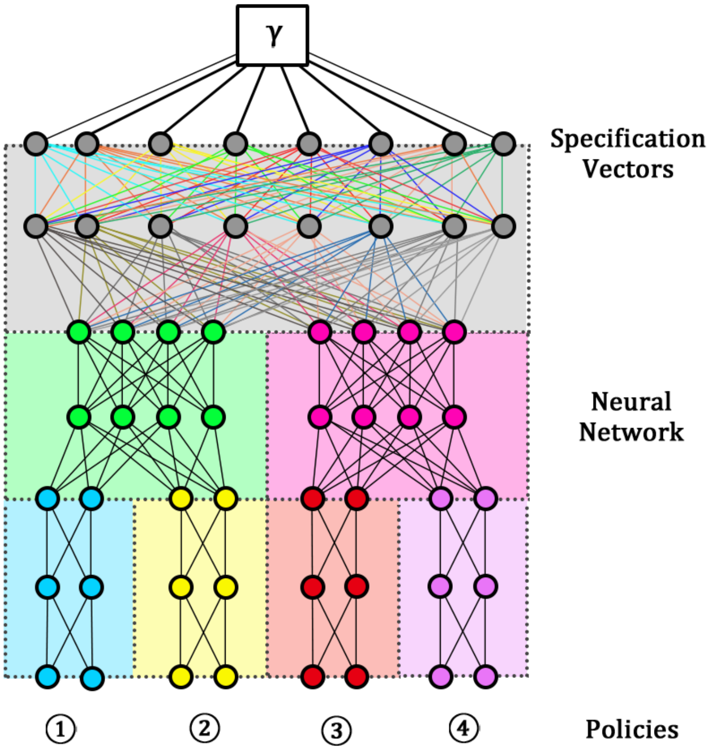
2.2.1 Policy Advisor Network

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 $\mathcal{P} : \Gamma \to \Theta$, where Γ is the specification parameter space and Θ is the policy parameter space.

Remark. The policy advisor is a neural network, $\mathcal{P}_{\mathbf{w}}(\gamma)$, initialized with a hierarchical topology. This is done with respect to the expected clusters emerging in Γ.

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

Agents and Actor Networks Each consumer that enters the system is assigned an agent, which operates according to the policy output by the advisor network. The goal of an agent is to maximize reward collected by the PAN by obtaining the optimal set of resources that satisfies the consumer's specification, which occurs through an iterative process executed on each time step. The process is as follows:

1. Given the state of the system, its policy parameters, and its assigned specification, each agent selects an action to take based on a stochastic decision model. The action is a composite structure of atomic action units, and will be described in detail later in this section.

2. The action is submitted to the system supervisor, which regulates large-scale behavior and controls the supply of reward to the PAN. Actions received from all agents are then interpreted and applied to the environment, potentially inducing a state transition (between state categories, as states are clustered in the same manner as consumer specifications).

3. Data on all consumers are updated, and reward is calculated and delivered to the PAN. Policies are revised accordingly, and a form of supervised learning is employed to reconfigure the network itself to better conform to the new policies.

Agent function is driven by an actor network, which maps each potential component of the action to a real number measuring its expected value to the consumer given the current state type of the system. The parameters output by the PAN serve as weights for the action network, and as are the PANDA's means of manipulating agent activity. Here it is important to note that this algorithm differs from traditional reinforcement learning algorithms in that reward is dispensed to a PAN rather than the reinforcement agents. The weights on the agent's action networks themselves are not trained - instead, training is concentrated on the advisor network that generates the weights from consumer specifications. In fact, agents and their networks have finite lifetimes, as they are disposed of upon becoming unemployed, with collected data extracted and incorporated into the next iteration of the PAN.

Remark. An agent is said to be *unemployed* when the consumer to which it has been assigned has departed from the system. As agents are specifically designed to cater to a particular consumer, unemployed agents have minimal value to the scheduler beyond the information they have gathered over their lifetimes.

Partial Observation and Attention Mechanism

Agent Actions At the end of each time step, agents submit an action to the system supervisor consisting of a three elementary action units. These action units will be performed in the order listed below upon execution of the 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. Catch requests are submitted as a Boolean vector, with each component corresponding to an item in the pool of observed resources.

2. Release - the agent returns a held resource to the pool of available resources. Release requests are also submitted as a Boolean vector, with each component corresponding to an item in the pool of held resources.

3. Wait - the agent delays consumption of resources until the subsequent time step. Wait requests are submitted as a Boolean scalar, as only one wait action may be performed per time step.

The action network output is mapped to a Boolean vector by applying the Gibbs softmax function and sampling from the resultant Boltzmann distribution.

2.2.3 System Supervisor

Agent decisions are submitted as actions to a system supervisor, which resolves collisions between catch requests[1]. The supervisor then initiates consumption of collected resources, excluding those held by an agent whose action included 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.

[1] Collision resolution methods currently being considered are allocating the resources in question on a first come, first served basis, and allocating them on the basis of need, determined from the output of the agent's actor networks.

3.2 Model Training

As was previously mentioned, the only metric that will be observed is the total time a consumer is in the system. With this in mind, we will first give a first attempt at a definition of an optimal scheduler. Let c_n represent the nth consumer leaving system. Firstly, we will observe a stochastic process, $\mathcal{D} = \{(T_n, \delta_n, \mu_n); n \in \mathbb{N}\}$, where T_n is the time between c_n and c_{n-1} times of occurence, δ_n is the total time consumer c_n was in the system, and μ_n is the utility of consumer c_n.

Definition 3.1. Let $\mathcal{D} = \{(T_n, \delta_n, \mu_n); n \in \mathbb{N}\}$ be a stochastic process and $\mathbf{c} = (c_1, \dots, c_n)$ be a sequence of consumers that have left the system. A scheduling policy is said to be sufficient if

$$\max_{i \in \underline{n}} \{\delta_i\} \leq S_n \leq \sum_{i \in \underline{n}} \delta_i$$

where $S_n = \sum_{i=1}^{n} T_i$.

Remark. The intuition following this definition is that a good scheduler will, depending on the resources, have the nth consumer departure greater than maximum total time among $\mathbf{c}$, if the resoures provided can admit an embarassingly parallel situation for consumer use, and less than the sum of all the total times in the system among $\mathbf{c}$.

The data that will be collected on the system are those corresponding to the arrival, scheduling, and departure process. The data set will be of the form

$$T = \{(\gamma_i, W_i, \vec{t_i}, \vec{\delta_i}, \mathcal{R}_i, r_i)\}_{i=1}^{n}$$

where γ_i is a specfication parameter vector, W_i are the weights used for γ_i, $\vec{t_i} = (t_1, t_2) \in \mathbb{R}^2_+$ such that t_1 is the arrival time and t_2 is the departure time of γ_i, $\vec{\delta_i} = (\delta_w, \delta_u) \in \mathbb{R}^2_+$ such that δ_w is the wait time and δ_u is the use time of γ_i, $\mathcal{R}_i$ is the set of resources assigned to γ_i, and r_i is the reward given to γ_i.

From this data set we will be able to determine the data for the arrival, scheduling, departure processes. Additionally, any information that we will need to derive for PANDA will be derived from this data.

The model will be trained using the data collected on the agents in the system and then using the rewards and updated policy parameters from each individual agent to update the weights of the PAN. Each agent will collect reward at the end of each episode in the system (when a successful scheduling has occurred). This training will most likely go under a slow convergence considering the parameters being trained are the weights of the actor networks of the agents. Currently, we are constructing methods for translating rewards to other specification and policy parameters by using measure-preserving transformations.

This method is explored as a way for overcoming the problem of learning how to distribute rewards for different policy parameters. This will hopefully lead to faster convergence, with respect to the PAN.

4 Discussion

Hybrid Approach

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