Scala Job Scheduler

February 2018

Contents

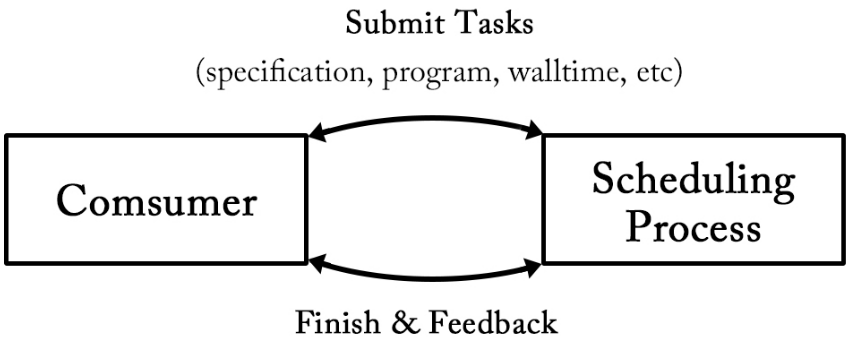
1 Introduction

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, 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 maximize throughput (efficient use of resources) while maintaining fairness, with wait times minimal and consistent between jobs that place similar demands on the system.

In order to build an algorithm capable of operating on a complex, heterogeneous system of resources and consumers, we must consider the process from an abstract yet granular point of view. In an attempt to do so we have defined the system in terms of simple mathematical objects, and constructed an algebra over those objects to describe their interaction. By decomposing the scheduling process into its constituent parts, we were able to describe each component of the process in terms of this algebra and thus devise a mathematical model for the process as a whole.

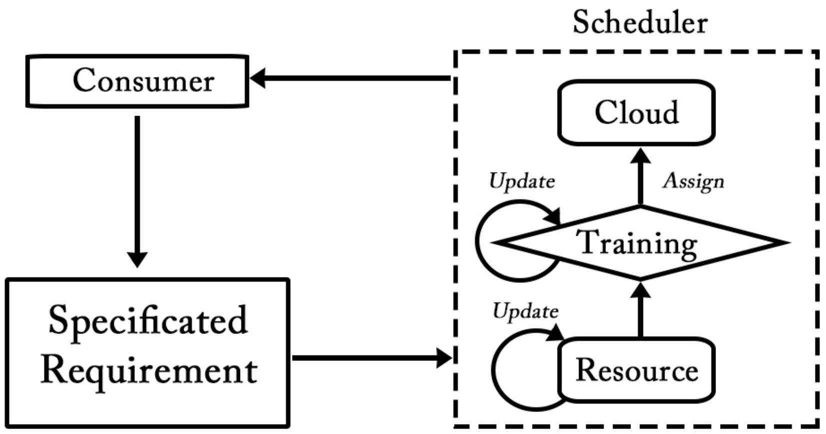
Due to the difficulty of processing such a dynamic, diverse system (given the sheer number of factors to account for), we elected to confront the problem with a reinforcement learning algorithm built on the aforementioned model. The algorithm is highly dependent on the system's partitioning into measurable (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 environment and scheduling process, component by component, and then the construction of the reinforcement model from those components.

1.1 General Process



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.

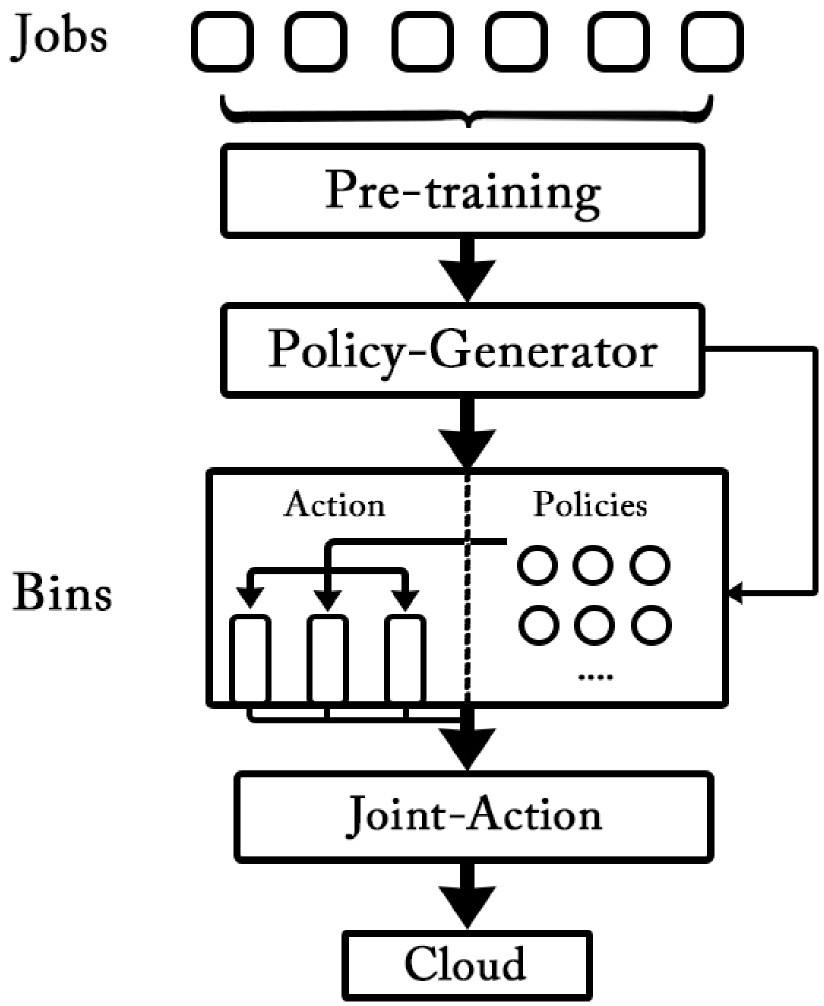
Example: a job is submitted with specifications such as arrival time, maximum run time, required number of nodes or cores, memory size, CPU/GPU, etc. After it finishes running, it gives feedback, such as wait time, actual run time, user reward, etc, to the scheduler.



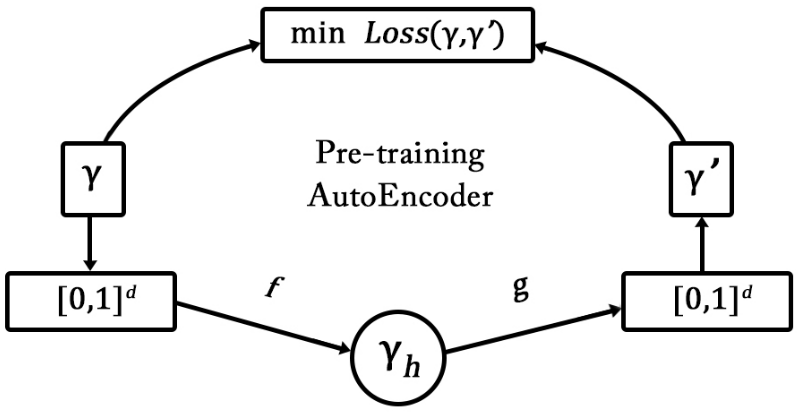
The scheduler first takes in the consumer specifications and inspects the current resource space. Then both the specifications and resource states are collected by the training module. During the reinforcement training, the training module and resource space are constantly being updated, while resources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

1.2 Consumer

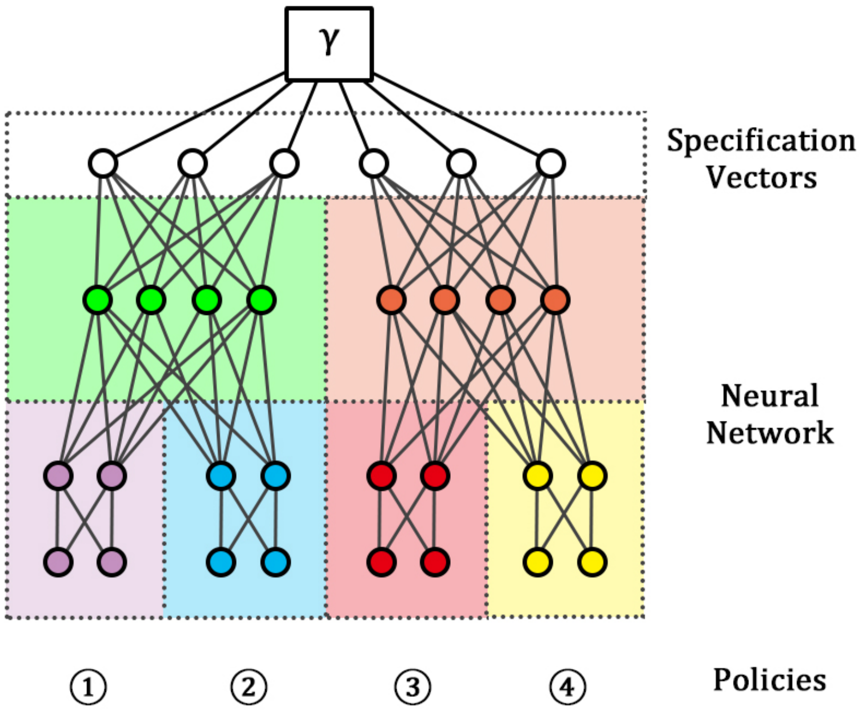
1.3 Scheduling Process



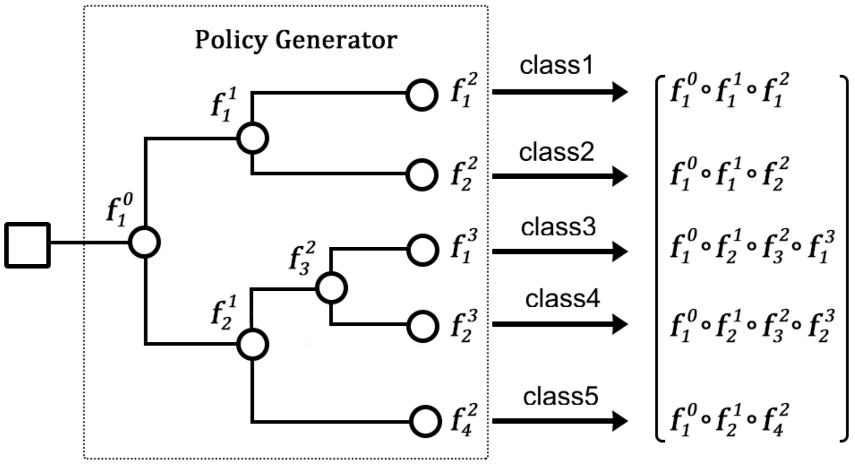
a. Pre-Training



b. Nerual Network



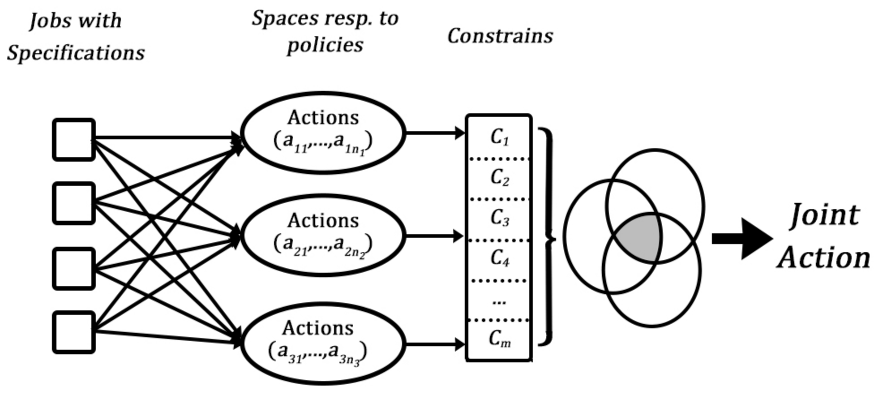
c. Policy Generator


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


Jobs with Specifications
Spaces resp. to policies
Constrains
Actions $(a_{11},\ldots,a_{1n_1})$
Actions $(a_{21},\ldots,a_{2n_2})$
Actions $(a_{31},\ldots,a_{3n_3})$
C_1
C_2
C_3
C_4
...
C_m
Joint Action

There are three agent actions: catch, wait and release. An agent can either catch unoccupied resources to assign to a consumer, make a consumer wait for a certain amount of time, or release the resources occupied by a consumer. Based on their corresponding policies, multiple agents can have different actions on one consumer. Then these actions will jointly apply to this consumer in a time order.

4.

Part I
Consumers, Requirements and Specifications

2 Consumer

Definition 2.1. A *consumer*, c, is a triple (k, ϕ, t), where $k \in \mathbb{N}$, ϕ is a specification, and $t \in \mathbb{R}^+$. The consumer C is the subset of consumer c which defined such that:

$$c = (\phi, \sigma_m)$$
$$C = 2^{\phi} * [0, \sigma_m)$$

Definition 2.2. Let *service* v be the available resource can be provide which related with a subset of resource R' and a boolean variable β can be defined such that:

$$v := (R', \beta)$$
$$V = 2^R * ß$$

3 Requirements

Definition 3.1. A *requirement* is a function $\rho : X \to \mathbb{B}$, where X is a set and $\mathbb{B} = \{0, 1\}$. The set X is referred to as the required set of ρ.

Definition 3.2. A requirement ρ is said to be *separable* if and only if it may be written as $\rho(x) = \prod_{i \in I} \rho_i(x)$, where $\forall i \in I$, $\rho_i(x)$ is a requirement with required set X.

Definition 3.3. A *requirement operator* is a mapping, $\mathbf{Req} : \mathbf{Set} \to \mathbf{Set}$, such that:

$$\mathbf{Req}_\rho(X) := \{\ x \in X \mid \rho(x) = 1\ \}.$$

Proposition 3.1. Given a requirement $\rho = \rho_1 \cdot \rho_2$, where ρ_1 and ρ_2 are requirements with required set X, then:

$$\mathbf{Req}_\rho(X) = \mathbf{Req}_{\rho_1}(X) \cdot \mathbf{Req}_{\rho_2}(X)$$

Remark. The binary operation $\cdot$ between two requirements is the same as the symbol, $\wedge$, used in boolean algebra to represent the join, *and*, between two boolean statements. Likewise, $\cdot$ operating on two sets is the intersection operation $\cap$.

Proof. The proof of this proposition is very straightforward. Let X be a set and ρ be a requirement with required set X. Then,

$$\begin{aligned}
\mathbf{Req}_\rho(X) &= \{\ x \in X \mid \rho(x) = 1\ \} && (1)\\
&= \{\ x \in X \mid \rho_1(x) \cdot \rho_2(x) = 1\ \} && (2)\\
&= \{\ x \in X \mid \rho_1(x) = 1 \, \mathit{and} \, \rho_2(x) = 1\ \} && (3)\\
&= \{\ x \in X \mid \rho_1(x) = 1\ \} \cdot \{\ x \in X \mid \rho_2(x) = 1\ \} && (4)\\
&= \mathbf{Req}_{\rho_1}(X) \cdot \mathbf{Req}_{\rho_2}(X) && (5)
\end{aligned}$$

□

4 Specifications

Definition 4.1. A set, X, is said to be inspectable if and only if there exists a function, $\psi : X \to \prod_{i \in I} X_i$, where $X \neq X_i, \forall i \in I$. This function is referred to as an *inspection function* of X.

Remark. The inspection function may also be expressed as, $\psi(x) = (\psi_i(x))_{i \in I}$.

Definition 4.2. A *specification* is a requirement, $\phi : X \to \mathbb{B}$, such that the following conditions hold:

1. The required set, X, is inspectable.
2. There exists a requirement, $\varphi : \prod_{i \in I} X_i \to \mathbb{B}$, such that $\rho = \varphi \circ \psi$, where ψ is an inspection function of X, with $X \neq X_i, \forall i \in I$. The requirement, φ, is referred to as an *acceptance requirement* of X.

Theorem. Given a separable specification, ϕ, with an acceptance requirement that is mutually independent, $\varphi(x_i)_{i \in I}$, then following isomorphism holds:

$$\mathbf{Req}_\phi(X) = \prod_{i \in I} \mathbf{Req}_{\varphi_i}(X_i)$$

Part II
Resources

Part III
Scheduling Process

5 Process Decomposition

Definition 5.1. let a represent the arriving process $a = (c, t_a)$. Then the assignment action A defined as:

$$A = C * [0, \infty)$$

Definition 5.2. let g represent the assignment process, $g = (c, \sigma_w, v)$. Then the assignment action G can be defined as:

$$G = C * [0, \sigma_m) * V$$

Definition 5.3. let d represent the departure process $d = (c, t_d)$, we also defined $\gamma(c) = min \sigma_r, \sigma_m$. Then the departure action D defined as:

$$D = C * [t_a, t_a + \sigma_w + \gamma(c)]$$

Part IV
Reinforcement Model

6 Policy Routing

6.1 Policy Graph

Definition 6.1. Let $G = (V, E, src, tgt)$ be a graph. A *path of length* n in G, denoted $p \in \mathbf{Path}_G^{(n)}$, is a head-to-tail sequence:

$$p = (v_1 \overset{a_1}{\to} v_2 \overset{a_2}{\to} \cdots \overset{a_{n-2}}{\to} v_{n-1} \overset{a_{n-1}}{\to} v_n)$$

The *set of all paths on* G is defined such that:

$$\mathbf{Path}_G := \bigsqcup_{n \in \mathbb{N}} \mathbf{Path}_G^{(n)}$$

Definition 6.2. let $\mathbf{Path}_G(v)$ be the all path from v back to v.
The *set of all path on* G is defined such that:
$\mathbf{Path}_G(s, t) = \mathbf{Path}_G(s, v) \sqcup \mathbf{Cycle}_G(v) \sqcup \mathbf{Path}_G(v, t)$.
The $\mathbf{Cycle}_G$ is defined as that:
$\mathbf{Cycle}_G(v) = \mathbf{Path}_G(v, v)$