Scala Job Scheduler

February 2018

Contents

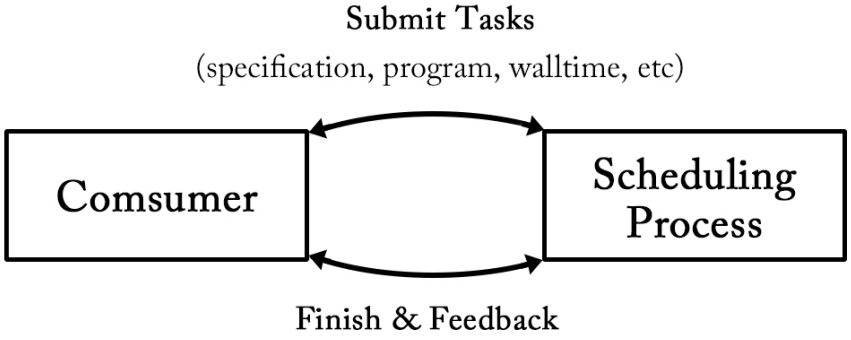
Introduction

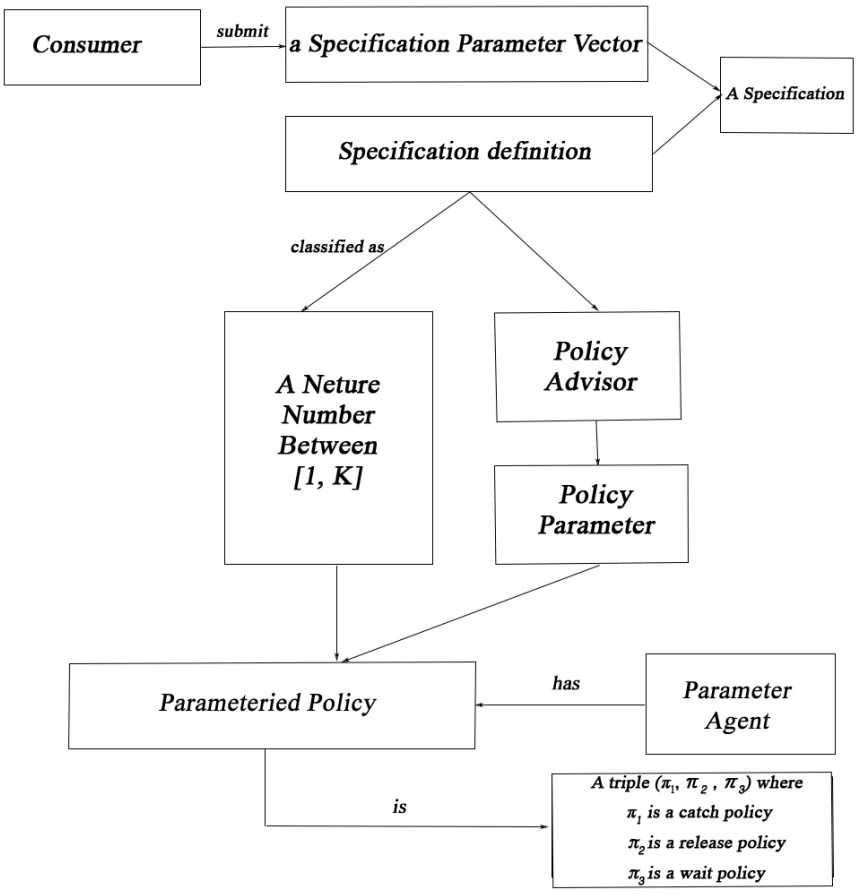
The aim of this project is to design an optimal scheduling algorithm for a scalable computing cloud, where computing resources are dynamically allocated to meet the demands of an inhomogeneous set of consumers. Resources are not uniformly distributed, geographically or otherwise, as the nodes comprising 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.

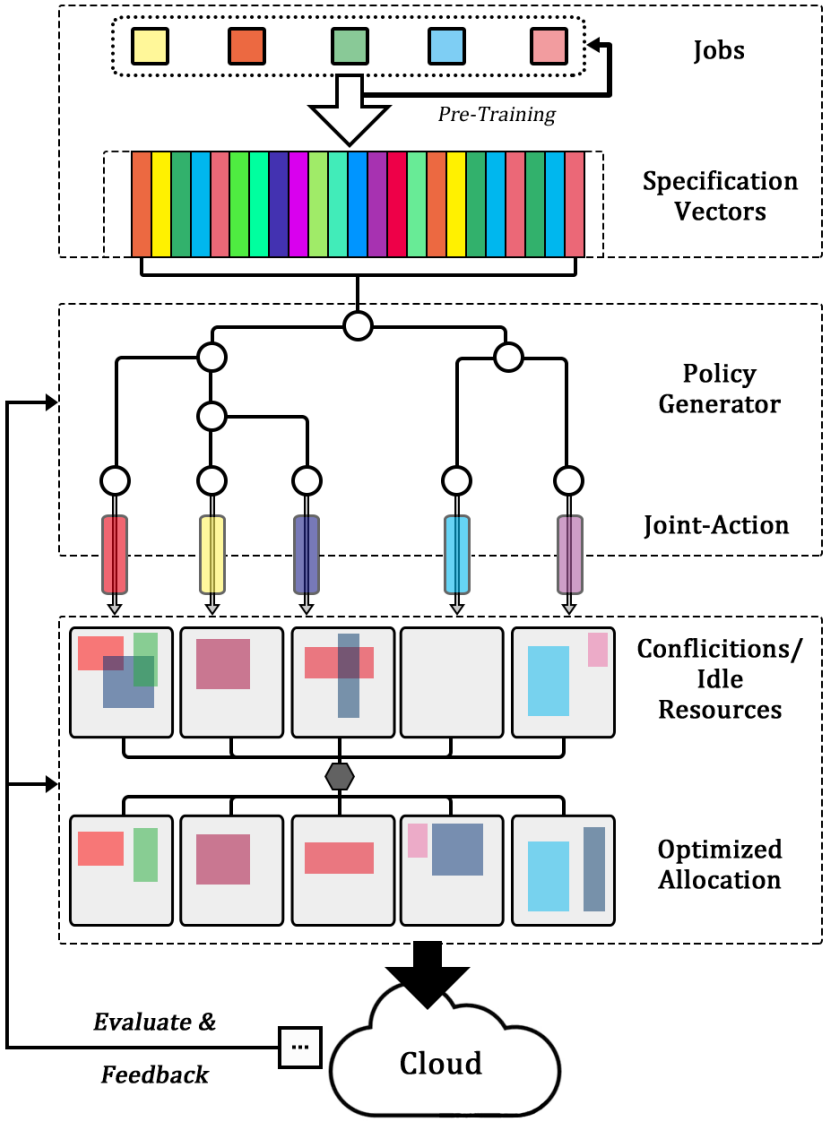
In order to build an algorithm capable of operating on a complex, heterogeneous system of resources and consumers, we must consider the process from an abstract yet granular point of view. In an attempt to do so we have defined the system in terms of simple mathematical objects, and constructed an algebra 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.

Part I
Scheduling Scheme

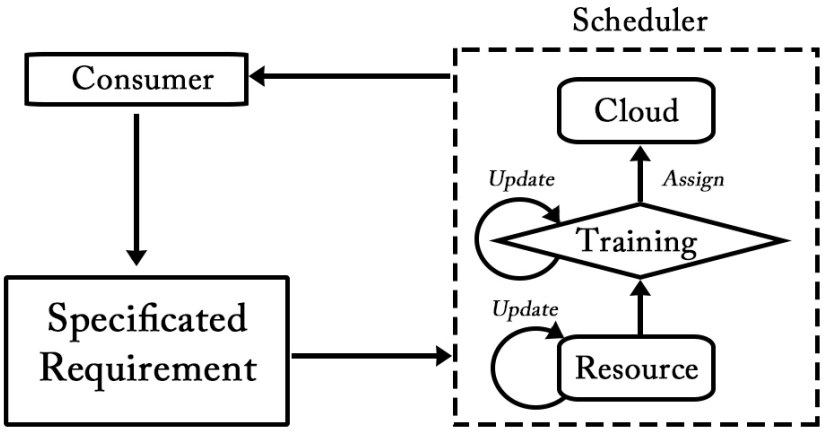



Consumer
submit
a Specification Parameter Vector
A Specification
Specification definition
classified as
A Neture Number Between [1, K]
Policy Advisor
Policy Parameter
Parameteried Policy
has
Parameter Agent
is
A triple (π_1, π_2 , π_3) where
π_1 is a catch policy
π_2 is a release policy
π_3 is a wait policy



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.



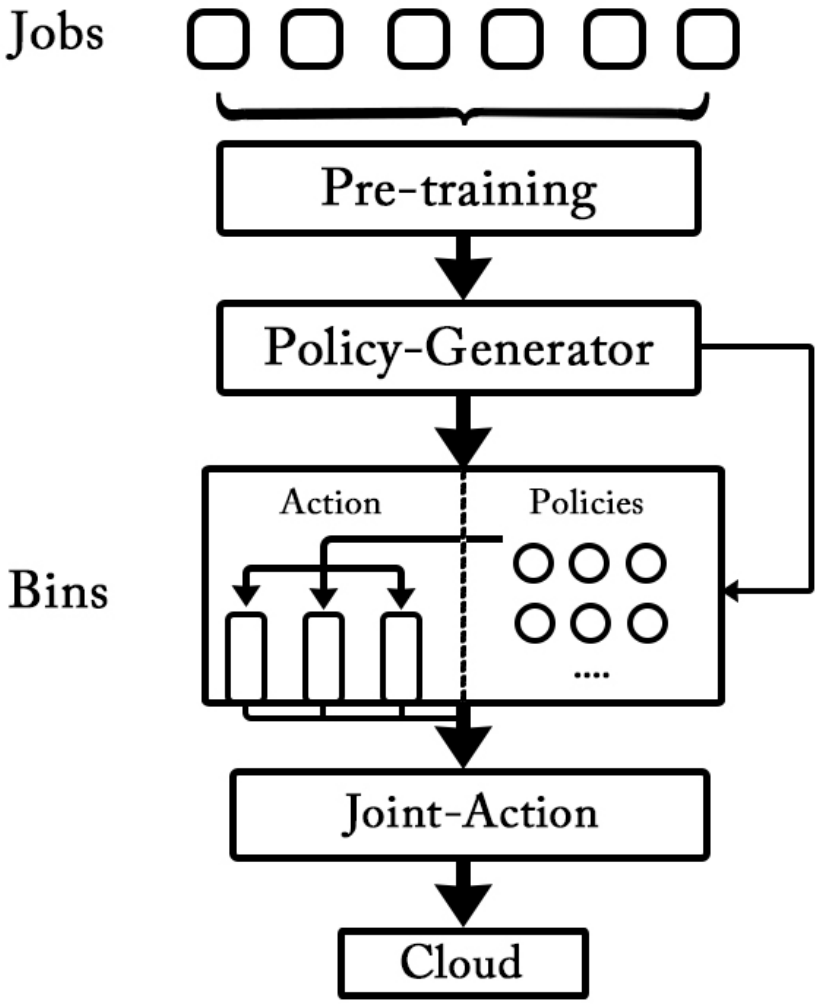
The scheduler first takes in the consumer specifications and inspects the current resource space. Then both the specifications and resource states are collected by the training module. During the reinforcement training, the training module and resource space are constantly being updated, while resources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

1 Specification

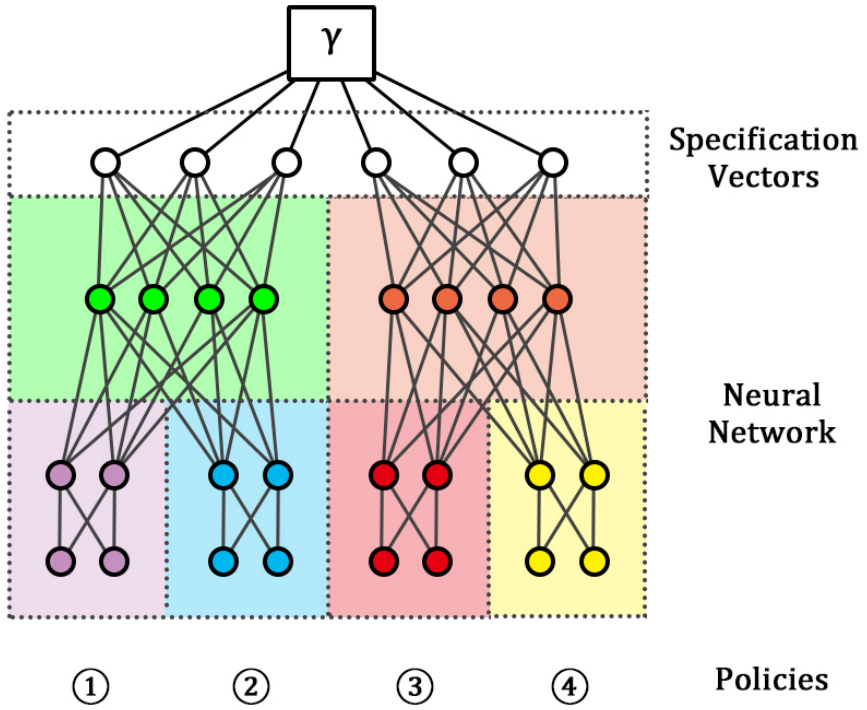
2 Consumer and Resource

3 Main Process

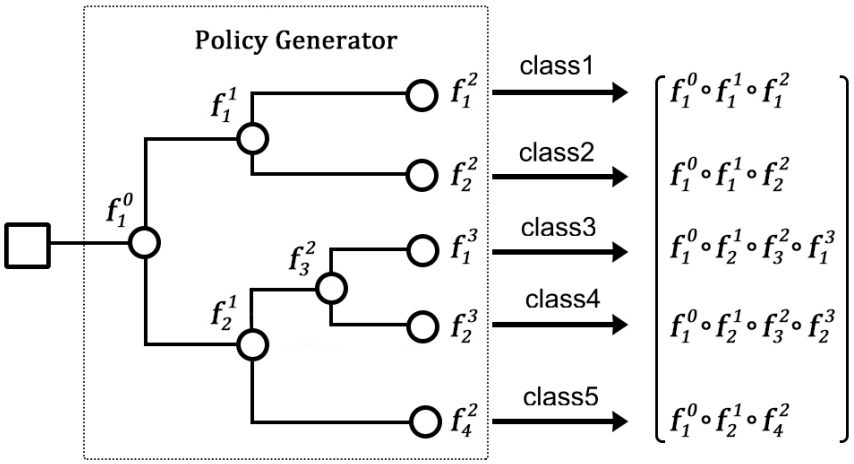
a. Main Training Process



b. Neural Network



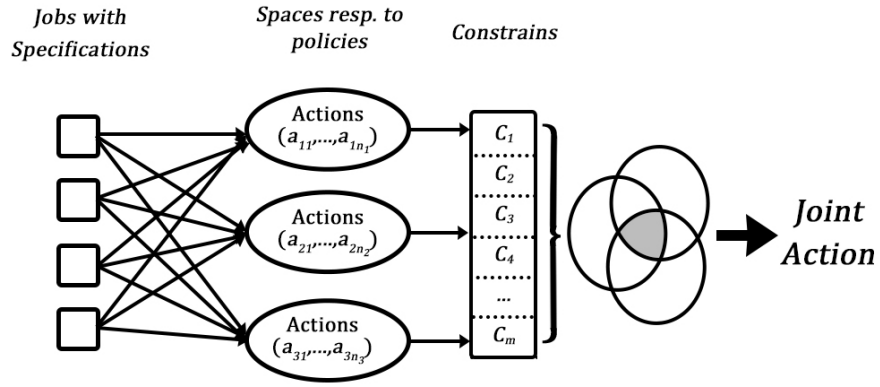
c. Policy Generator



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.

4 Consumer

Definition 4.1. 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 4.2. 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:

$$v := (R', \beta)$$

$$V = 2^R * ß$$

5 Requirements

Definition 5.1. 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 5.2. 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 5.3. 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 5.1. 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,

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

□

6 Specifications

Definition 6.1. 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 6.2. 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
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 $\mathcal{P} : \Gamma \to \Theta$, where Γ is the specification parameter space and Θ is the policy parameter space.

4 System Supervisor

Agent decisions are submitted to a system supervisor, which resolves collisions (conflicts) between requests

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