Scala Job Scheduler

March 2018

Abstract

We present a novel scheduling architecture PANDA (Policy Advisor Network and Decision Architecture) built on a multi-agent reinforcement learning model, specially designed for inhomogeneous and elastic cloud environments where resources are delivered on-demand to an inconstant stream of consumers. To combat environmental instability, the architecture makes use of a policy advisor network (PAN), designed with a hierarchical topology yielded from a Bayesian hierarchical clustering analysis on consumer specifications, computing resources, and environmental states. Decision-making is decentralized, but scheduling agents operate in a collaborative environment and share a joint return distributed to the PAN, which translates the clustered consumer specifications into weights on the actor networks driving agent decisions. This design results in many advantages over traditional schedulers, among them improved adaptability, optimized resource selection, and robustness in the face of an otherwise intractable environment.

Introduction

The aim of this project is to design an optimal scheduling algorithm for an elastic computing cloud, where computing resources are dynamically allocated to meet the demands of a broad range 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 at times unreliable resource pools. 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 the training of this model and further discussions.

1 Previous Work

Traditional scheduling theory has been thoroughly studied, with different approaches such as deterministic and stochastic models and wide range of system implementations [1]. In recent years, along with the significant advances of deep learning, multi-agent, multi-task learning in non-stationary environments have started to attract attention [2]–[7]. Combining these two areas would provide a solution mechanism to the dynamic scheduling problem. Several similar scheduling studies concerning the dynamic environments [8], [9] showed that this issue has been of growing importance in large-scale HPC and cloud platforms. However, most of the previous studies have not taken into account the full range of the changing factors in a highly distributed cloud environment, while maintaining good scheduling performance.

2 Preliminary Data Processing

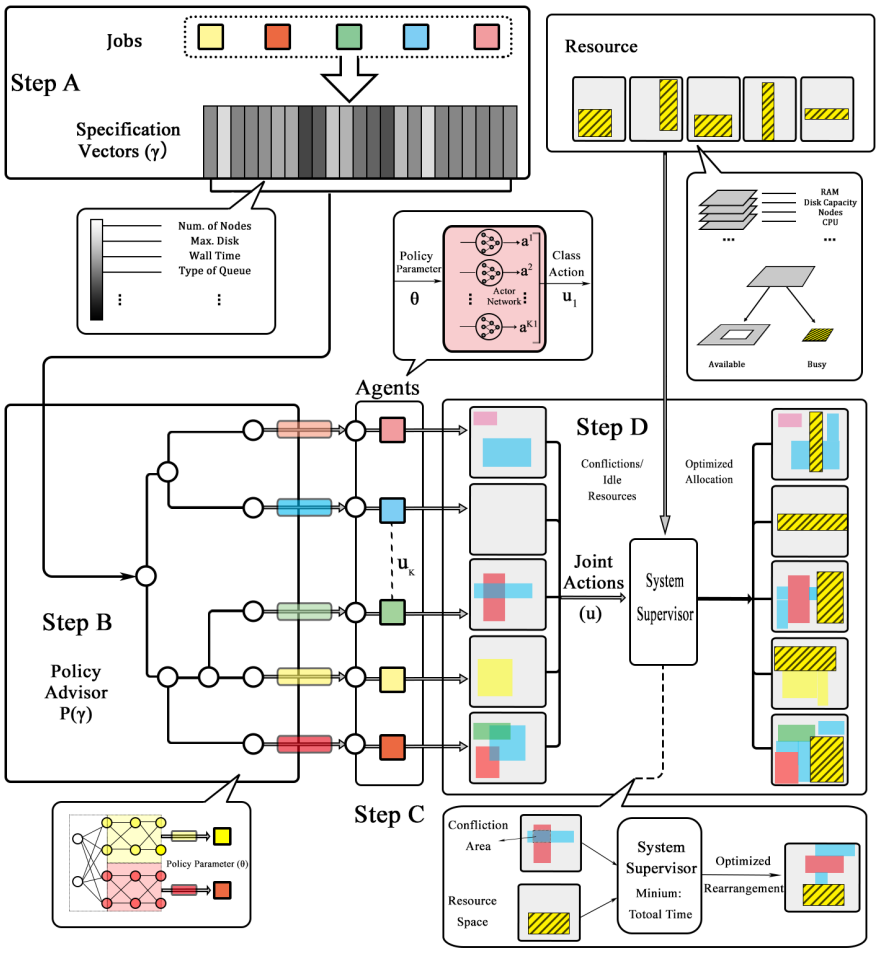
In the interest of improving efficiency and accelerating convergence of the reinforcement model, aspects of the environment are grouped utilizing a probabilistic approach to agglomerative hierarchical clustering called Bayesian hierarchical clustering [10]. Using this method, consumers, resources, and states are classified into types, each representable by a numeric label (e.g. 1, 2). After classification, consumer specification parameters and consumer types are fed into a policy advisor network, which then outputs policy parameters for a scheduling agent to use for the duration of their search. Resource and state types are reserved for components of the agent decision model, where they can greatly improve algorithmic performance simply by reducing their respective spaces to tractable sizes.

3 Policy Advisor Network and Decision Architecture (PANDA)

This proposed architecture can handle both dynamic agent population and state space when processing diverse systems. 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. From this perspective, the PANDA 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 into general scheduling processes.

3.1 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.



. Figure 1: General Scheduling Process (PANDA)

(A) **Submission**
Upon submission by a user, a consumer enters the scheduling process and submits its specifications as a parameter vector. The parameter vector is classified as being a certain consumer type (some positive integer we refer to as the consumer's class label), and then both the label and the parameter vector are given to the policy advisor as input.

(B) **Construction**
Using the class label and the parameter vector, the policy advisor outputs the actor network parameters for the consumer's representative scheduling agent.

(C) **Allocation**
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.

(D) **Consumption**
In the end, the consumer leaves the scheduling process by consuming (running on) assigned resources. Once finished, the consumer gives feedback to the scheduler for further training improvement.

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

3.2 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 useful in combinatorial problem solving.

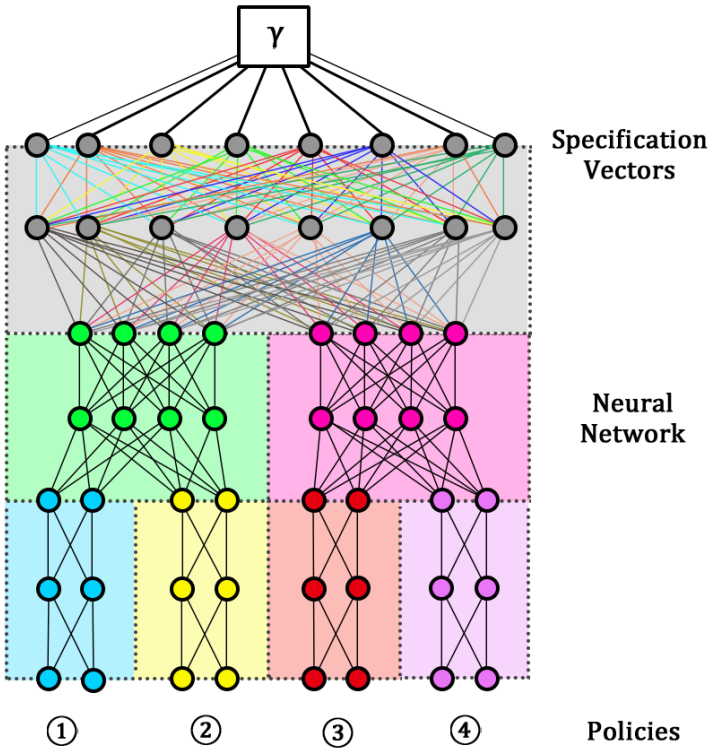
3.2.1 Policy Advisor Network (PAN)

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 a 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.

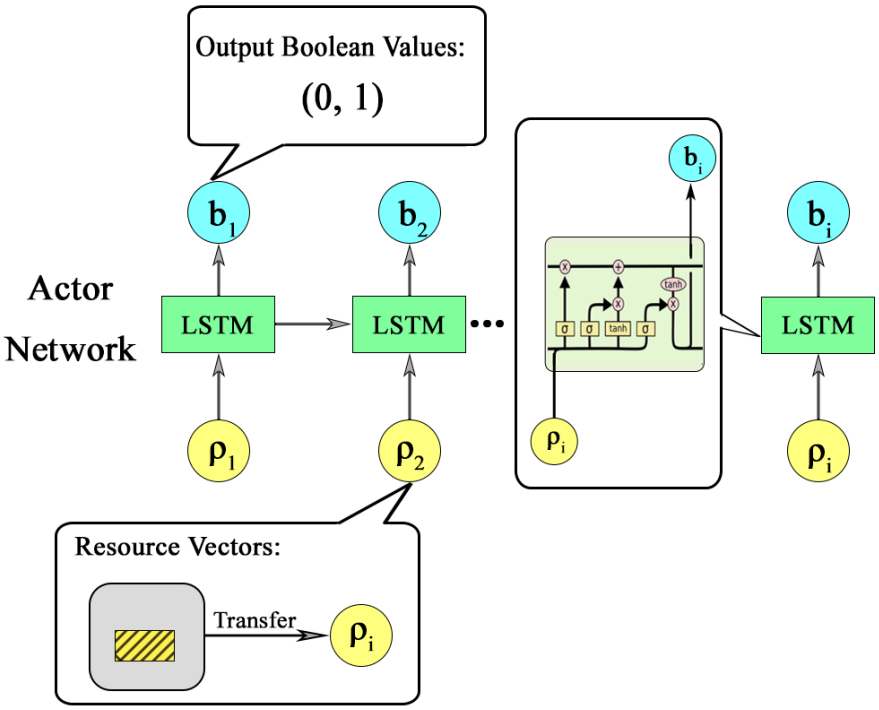
Topology The diagram below shows a simplified PAN topology. The colors represent the different classes of the specification parameter space. The numeric labels are indicative of the specification parameter vectors.



. Figure 2: Advisor Topology

3.2.2 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. Resource collection occurs through the following iterative process, executed on each time step:



. Figure 3: Actor Network

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 decision-making is driven by a neural network known as an actor network, uniquely configured to meet the demands of the agent's consumer. Each actor network is a function $\alpha : S \times A \to \mathbb{R}$, such that $\alpha_\theta(a|s)$ is a measure of

the expected value of the action a to the consumer. The parameters output by the PAN serve as weights for the actor network, and as such 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 directly the reinforcement agents[1]. This is necessitated by the unpredictability of consumer influx and consumer demands, as the agent population fluctuates accordingly, and agent policies are tailored to consumer specifications to optimize performance and adaptability. With training focused on the PAN, learning is centralized despite a decentralized decision process, eliminating what would have been a restrictive reliance on agent longevity or forms of continuous evolution. Agents and their networks thus have finite lifetimes, and may be 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.

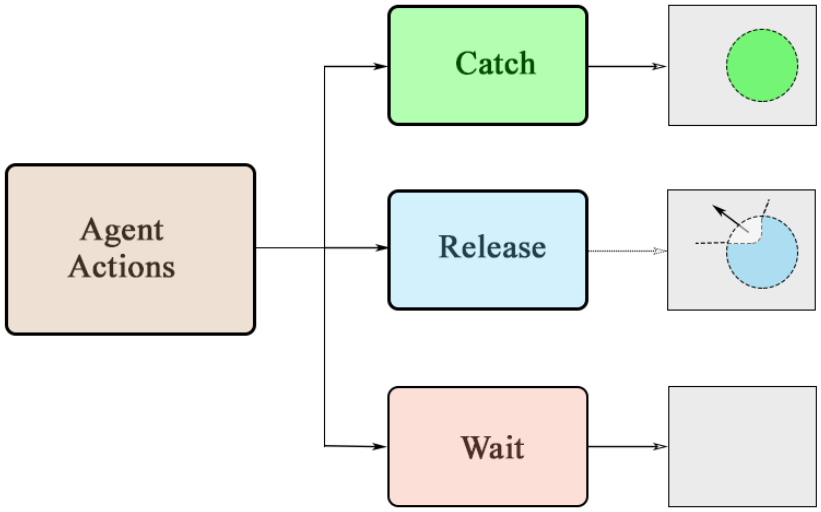
Partial Observation and Attention Mechanism In an effort to reduce the time complexity of the agent decision process, agent observation is limited to subset of viable resources rather than the entire resource space. This form of agent perception is not unlike that of humans, who tend to fixate on objects of interest and filter out ostensibly irrelevant features of their environment. While previously developed selective attention mechanisms such as the one outlined in [11] have largely been applied to visual environments, the concept may be adapted to analogous computational systems, where agents must make partial observations of the environmental state. To mimic a human-like attention mechanism, the agents thus apply an aptly-named filter function $\mathcal{F} : \mathbf{Set} \to \mathbf{Set}$ to sets of resources as they become available for consumption. Mathematically speaking, $\mathcal{F}$ maps a set of resources X to some $X' \subseteq X$, such that X' contains only those elements of X whose expected value to the consumer surpasses a certain threshold.

The implementation of this function hinges on how the relative value of a resource may be estimated without compromising efficiency or relying on static numerical thresholds, which could compromise performance when operating within a dynamic environment. A potential solution employs the categorical representation of resources by assigning a weight to each resource type based on its conformity to the specification. As each resource category occupies some

[1]The model is not unlike those used in meta-learning, where high-level procedures are used to approximate effective policies for new tasks (see [2], [7]). In this instance, however, the goal is to generate an optimal policy based on a consumer specification, rather than approximate one to reduce learning time.

location in N-dimensional space, given by the mean vector representation of its constituents, a weighted average of these locations may be computed to estimate the range of acceptable locations for a resource given the consumer's specification. Resources within a certain distance limit (perhaps some function of the variance in type location) would pass the filter, and be integrated into the agent's pool of observed resources. An alternative approach could be to accept entire categories based on their weights, and compute a threshold based on the type of environmental state, but the former would permit finer filtering.

Agent Actions At the end of each time step, agents submit an action to the system supervisor consisting of 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:



. Figure 4: Agent Action

(A) **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.

(B) **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.

(C) **Wait**
T he 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 actor network output is mapped to a Boolean vector by applying the softmax function and sampling from the resultant categorical distribution.

3.2.3 System Supervisor

Agent decisions are submitted as actions to a system supervisor, which resolves collisions between catch requests[2]. Once conflicts are resolved, the agent actions are executed in order of submission. The supervisor then initiates consumption of collected resources, excluding those held by an agent whose action included a wait operation.

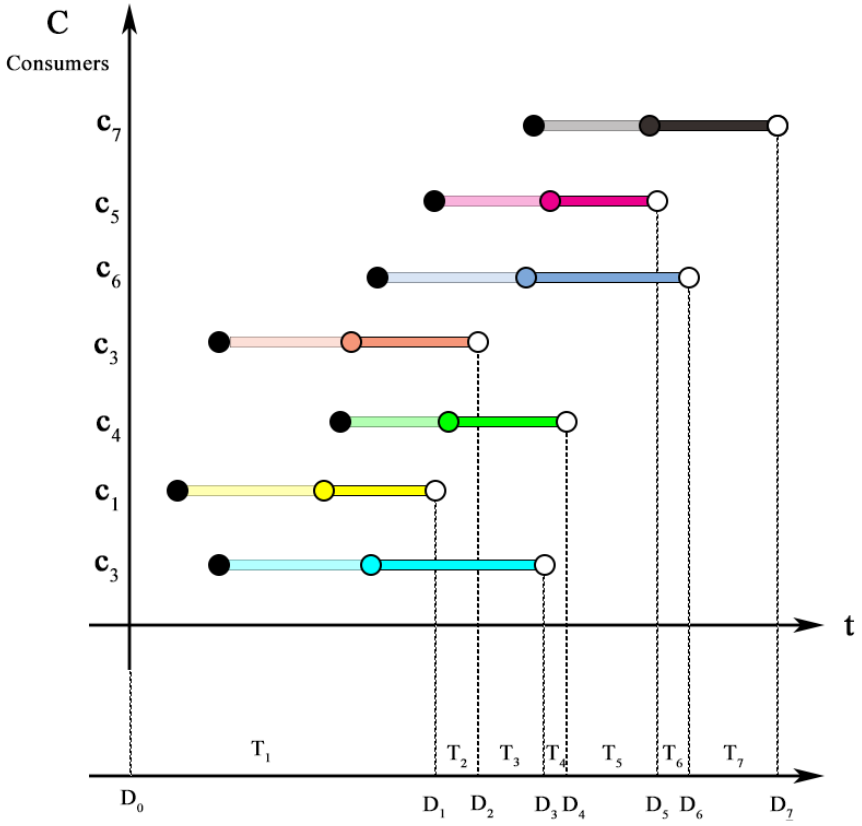
The supervisor also tracks the progression of each specification through the system, and distributes reward to the respective agents according to a function of the collected data and potential consumer feedback. In the instance of the scalable cloud, reward will be computed upon completion of the job, 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 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.

[2]Collision resolution methods currently being considered are (a) allocating the resources in question on a first come, first served basis, and (b) allocating them on the basis of need, determined from the output of the agent's actor networks. Effective resolution could likely be achieved through trained communication protocols such as those outlined in [4], but the centralized approaches would be simpler to implement and would not require additional training.

4 Training

4.1 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 what constitutes a valid schedule. 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 occurrence, δ_n is the total time consumer c_n was in the system, and μ_n is the utility of consumer c_n.



. Figure 5: Consumer Departure Process

Definition 4.1. Let $\mathcal{D} = \{(T_n, \delta_n, \mu_n); n \in \mathbb{N}\}$ be a stochastic process and $\mathbf{c} = (c_1, \ldots, c_n)$ be a sequence of consumers that have left the system. A scheduling

policy is said to be *valid* if

$$D_n \in \left[\max_{i \in \underline{n}} \{\delta_i\}, \sum_{i \in \underline{n}} \delta_i \right] \subseteq \mathbb{R}_+$$

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

Remark. The distance of S_n from the lower and upper bound represent the degree of parallel uses of the resources, the lower bound being embarrassingly parallel and the upper bound being strictly sequential.

The intuition following this definition is that a valid scheduling policy will effectively , depending on the resources, force the sum of interdeparture times to fall within these bounds. If D_n falls above the upper bound then the scheduling policy was *invalid*. Therefore, the desired objective in a schedule is to minimize the expected lower bound of the interval, and maximize the expected difference of D_n from the upper bound of the interval.

Here is a list of some metrics for scheduling and our reasons for believing they are dependent on time.

(A) **Flow Rate** (**Throughput**)
The flow rate is the number of jobs that the system can process within a given time interval. This is dependent on the interdeparture times, the maximum total time and the sum of all total times a job is in the system. A minimization in expected interdeparture times will lead to more jobs being completed in a given time interval, meaning a higher flow rate.

(B) **System Utilization**
If the system is being utilized to its full capacity it would mean that it could complete more jobs within a given time interval than if it was not being fully utilized, which is along the lines of maximizing flow rate.

(C) **Fairness**
Fairness is a subjective metric because it is completely dependent on the agents claiming unfairness. Additionally most literature tries to maximize fairness, however, this seems odd for the fact that there is no optimal level of fairness, there is just equilibrium on total time in the system among agents. In which case the goal would be to minimize deviation from equilibrium, hence minimizing unfairness by minimizing variance among total times in the system. It is much easier to see what is not fair than what is fair, therefore it should ultimately be up to the users instead of the creators. If they feel that the scheduling is unfair then they should be given the option to say so. This has the benefit of being able to directly know what constitutes a "fair" scheduling

algorithm. Additionally, we can be sure that all agents would like to be in the system for as short of a time as possible, hence minimizing the expected total time in system would, at worst, have no effect on the fairness and would, at best, render a less unfair system.

Given this definition there are some reward functions that will be reviewed and chosen based on performance of the scheduler. At the agent level the goal is to minimize the total time that their job is in the system, at the class level it is to minimize unfairness among agents within the class, and at the system level it is to minimize the sum of interdeparture times.

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 specification 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 required by the PANDA will be derived from this data set.

The model will be trained using the data collected on the agents in the system, 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 under go 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.

5 Discussion

The major advantages of using PANDA for job scheduling is its potential for online scheduling. This provides a more streamline scheduling experience while rendering a cohesive system that relies on less metrics than a typical static scheduler. A general multitude of simulations will be run in order to locate conflicts that may arise with our approach to this problem, allowing us to fine

tune individual issues which can prevent problems that may arise in the metadata before receiving the data.

Hybrid Approach In order to integrate PANDA into a real-time system we will gradually transition from an static model. We could implement a cake-cutting algorithm [12], [13] that will use a discrete envy-free protocol, equipped for handling any number of agents.

Remark. This approach will be dealt with on a cautionary level because the integrity of scheduling data will be affected as different scheduling algorithms are used. We will take this as an opportunity to gather large amounts of data that exceed the currently available data in literature and given out by other companies (e.g. Google Cloud Platform and Amazon Web Services).

6 Future Work

In future work, we will be establishing more formal and concrete definitions on scheduling policies (better than what there currently is). We will be exploring this by utilizing literature on measure-preserving dynamical systems, decentralized partially observable Markov decision processes, and competitive/cooperative multi-agent systems. Additionally, we will be observing and analyzing the real-time data we receive from the system, in order to guide the future of the architecture in the right direction. We will also be keeping the integrity of the data as our highest priority, as for it to have no dependence on the scheduling system that is used. This choice should only affect observations such as inter-departure times and the average total time in the system for a consumer.

References

[1] M. L. Pinedo, *Scheduling: theory, algorithms, and systems*. Springer, 2016.

[2] M. Al-Shedivat, T. Bansal, Y. Burda, I. Sutskever, I. Mordatch, and P. Abbeel, "Continuous adaptation via meta-learning in nonstationary and competitive environments", *arXiv preprint arXiv:1710.03641*, 2017.

[3] S. Omidshafiei, J. Pazis, C. Amato, J. P. How, and J. Vian, "Deep decentralized multi-task multi-agent rl under partial observability", *arXiv preprint arXiv:1703.06182*, 2017.

[4] J. Foerster, Y. Assael, N. de Freitas, and S. Whiteson, "Learning to communicate with deep multi-agent reinforcement learning", in *Advances in Neural Information Processing Systems*, 2016, pp. 2137–2145.

[5] A. Tampuu, T. Matiisen, D. Kodelja, I. Kuzovkin, K. Korjus, J. Aru, J. Aru, and R. Vicente, "Multiagent cooperation and competition with deep reinforcement learning", *PloS one*, vol. 12, no. 4, e0172395, 2017.

[6] T. Bansal, J. Pachocki, S. Sidor, I. Sutskever, and I. Mordatch, "Emergent complexity via multi-agent competition", *arXiv preprint arXiv:1710.03748*, 2017.

[7] K. Frans, J. Ho, X. Chen, P. Abbeel, and J. Schulman, "Meta learning shared hierarchies", *arXiv preprint arXiv:1710.09767*, 2017.

[8] W. Tang, Z. Lan, N. Desai, and D. Buettner, "Fault-aware, utility-based job scheduling on blue, gene/p systems", in *Cluster Computing and Workshops, 2009. CLUSTER'09. IEEE International Conference on*, IEEE, 2009, pp. 1–10.

[9] D. Carastan-Santos and R. Y. De Camargo, "Obtaining dynamic scheduling policies with simulation and machine learning", in *The International Conference for High Performance Computing, Networking, Storage and Analysis (Supercomputing)*, 2017.

[10] K. A. Heller and Z. Ghahramani, "Bayesian hierarchical clustering", in *Proceedings of the 22nd international conference on Machine learning*, ACM, 2005, pp. 297–304.

[11] V. Mnih, N. Heess, A. Graves, *et al.*, "Recurrent models of visual attention", in *Advances in neural information processing systems*, 2014, pp. 2204–2212.

[12] A. D. Procaccia, "Cake cutting algorithms", in *Handbook of Computational Social Choice, chapter 13*, Citeseer, 2015.

[13] H. Aziz and S. Mackenzie, "A discrete and bounded envy-free cake cutting protocol for any number of agents", in *Foundations of Computer Science (FOCS), 2016 IEEE 57th Annual Symposium on*, IEEE, 2016, pp. 416–427.