An Intelligent, Hierarchical Framework for Adaptive Buffer Allocation in the CISCO Nexus Data Center Switches

A Deep Deterministic Policy Gradient (DDPG) Approach in a traditional Asynchronous Advantage Actor-Critic (A3C) DDQN-PER Architecture

Bharath Keshavamurthy and Imran Pasha

CISCO Systems, Inc.

June 2019

1 System Model

1.1 Definitions

- $N_P \triangleq$ The number of ports in the switch
- $N_Q \triangleq$ The number of queues per port in the switch
- Each port has a dedicated buffer space with capacity,

$$C_{local}^{P_i},\ \forall i \in \{0, 1, 2, \ldots, N_P - 1).$$

  When this dedicated buffer space is exhausted, i.e. allocated at any given time, the "**Intelligent Buffer Allocation Engine**" within the switch allocates the required amount of buffer space from the switch's global pool denoted by C_{global}.

- The size of the port-specific local pool and the global pool are design specific, i.e. varies from one switch variant to another. The engine proposed in this paper is agnostic to these design specifications as long as the switch complies with the basic framework outlined above.
- The arrival process of packets at each queue is modelled as a **Poisson Process** with rate $\lambda_{P_iQ_j} > 0,\ \forall i \in \{0, 1, 2, \ldots, N_P - 1\}$ and $j \in \{0, 1, 2, \ldots, N_Q - 1\}$ denoted by,

$$\{N_t,\ t \geq 0\}.$$

- There are two kinds of queues in every port - a High-Priority Queue with high service rate and a Low Priority Queue with low service rate. The service process at each queue is modelled as an **Exponential Process** with rates $\mu^{high}_{P_iQ_j} > 0$ and $\mu^{low}_{P_nQ_m} > 0$ where $\mu^{high}_{P_iQ_j} >> \mu^{low}_{P_nQ_m}$ for any $i, n \in \{0, 1, 2, \ldots, N_P - 1\}$ and $j, m \in \{0, 1, 2, \ldots, N_Q - 1\}$ such that $P_iQ_j \neq P_nQ_m$.

1.2 The Framework

- The queueing systems within the switch are modelled as **M/M/1 systems** with test scenarios covering varying degrees of load/utilization, i.e.,

$$\rho \triangleq \frac{\lambda_{P_iQ_j}}{\mu^{high|low}_{P_iQ_j}} \; \forall i \in \{0, 1, 2, \ldots, N_P - 1\} \text{ and } j \in \{0, 1, 2, \ldots, N_Q - 1\}. \quad (1)$$

- The adaptive, hierarchical, and intelligent buffer allocation procedure is modelled as a **Markov Decision Process (MDP)** defined as a 6-tuple, $(\mathcal{S}, \mathcal{A}, \mathcal{Y}, \alpha, \beta, \Pi)$ where, $\mathcal{S}$ denotes the state space, $\mathcal{A}$ denotes the action space, and $\mathcal{Y}$ denotes the observation space. Furthermore, the transition model α, emission model β, and the initial steady-state model Π of the MDP are unknown. This lack of process information encourages us to explore the use of Model-Free Learning Techniques to achieve optimal buffer allocation heuristics.

- We could also employ a Parameter Estimation algorithm based on the principles of **Maximum Likelihood Estimation (MLE)**, i.e.,

$$\hat{\rho}^* = \underset{\rho}{\text{argmax}} \; \mathbb{P}(\vec{Y} \mid \rho).$$

  One such technique is the well-known **Expectation-Maximization Algorithm** which can be employed to estimate the parameters of the underlying Markov model in the allocation process. But, as discussed in the subsequent sections of the document, the enormous state, action, and observation spaces make it impossible to apply standard MLE algorithms to estimate the model parameters. Therefore, Model-Free Learning techniques leveraging Artificial Neural Networks (ANNs) as **function-approximators** to estimate the action-value functions is the most optimal approach.

- The state space, the action space, the observation space, and the reward/cost metrics are detailed in the subsequent subsections.

1.3 The State Space

The state space of the underlying MDP is modelled as,

$$\begin{aligned}\mathcal{S} \equiv \{\vec{S} \mid \vec{S} = [\mathbf{X}\ \delta_{global}]^\intercal \mid \mathbf{X} \text{ is a } (N_P)\text{x}(N_Q+1) \text{ matrix with} \\ \vec{x}_{ij} \in \mathbf{X} \equiv [\{P_iQ_j\}_{min}\{P_iQ_j\}_{max}\{P_iQ_j\}_{alloc}\{P_iQ_j\}_{drop}], \\ \forall i \in \{0,1,2,\ldots,N_P-1\} \text{ and } j \in \{0,1,2,\ldots,N_Q-1\}\}.\end{aligned} \tag{2}$$

From the above definition of the state space, it is evident that the state space is huge: the exact number of total possible states depends on the design of the switch, i.e. the required minimum buffer space in the queue $\{P_iQ_j\}_{min}$, the maximum allowed buffer space in the queue $\{P_iQ_j\}_{max}$, and the size of the dedicated pool $C_{local}^{P_i}$ and the shared pool per port C_{global}.
For a switch with three ports and two queues per port, i.e. $N_P = 3$ and $N_Q = 2$, a state in the state space can be represented as $\vec{S} = [X\ \delta_{global}]^\intercal \in \mathcal{S}$ where, X can be written as

$$\begin{bmatrix} [\{P_0Q_0\}_{min}\{P_0Q_0\}_{max}\{P_0Q_0\}_{alloc}\{P_0Q_0\}_{drop}] & [\{P_0Q_1\}_{min}\{P_0Q_1\}_{max}\{P_0Q_1\}_{alloc}\{P_0Q_1\}_{drop}] & \delta_{local}^{P_0} \\ [\{P_1Q_0\}_{min}\{P_1Q_0\}_{max}\{P_1Q_0\}_{alloc}\{P_1Q_0\}_{drop}] & [\{P_1Q_1\}_{min}\{P_1Q_1\}_{max}\{P_1Q_1\}_{alloc}\{P_0Q_1\}_{drop}] & \delta_{local}^{P_1} \\ [\{P_2Q_0\}_{min}\{P_2Q_0\}_{max}\{P_2Q_0\}_{alloc}\{P_2Q_0\}_{drop}] & [\{P_2Q_1\}_{min}\{P_2Q_1\}_{max}\{P_2Q_1\}_{alloc}\{P_2Q_1\}_{drop}] & \delta_{local}^{P_2} \end{bmatrix}$$

where, $\delta_{local}^{P_i}$ denotes the leftover buffer space in the dedicated pool of port $P_i, \forall i \in \{0,1,2,\ldots,N_P-1\}$ and δ_{global} denotes the leftover buffer space in the global pool (switch-level).
The presence of this complex trickle-down allocation requirement along with varying degrees of priority among queues make this problem a very difficult one: warrants the need for an adaptive, hierarchical, intelligent allocation algorithm design. More on this later.

1.4 The Action Space

The action space of the underlying MDP is modelled as,

$$\mathcal{A} \equiv \{\vec{A} = [A,\ \Delta C_{global}]^\intercal \mid A = [a_{ij}]\}. \tag{3}$$

where, a_{ij} corresponds to Δs where, Δs corresponds to a change (increment or decrement in the allocated buffer units) with respect to the queues corresponding to a port or the change (increment or decrement in the leftover buffer units) with respect to the dedicated local pool of a specific port. Furthermore, ΔC_{global} denotes the change (increment or decrement in the leftover buffer units) with respect to the global pool of the switch.

1.5 The Observation Space

- Within the context of the design of a switch, it does not make sense to consider partially observable (incomplete observations and/or noisy observations), hence, we will not approach this problem as a **Partially Observable Markov Decision Process (POMDP)** one.

- In this problem, we assume the **system states are completely observable in a noiseless environment**.
- The observation space is modelled as,

$$\mathcal{Y} \equiv \{\vec{Y} \mid \vec{Y} = \vec{S},\ \vec{S} \in \mathcal{S}\}. \tag{4}$$

1.6 The Reward Metrics

- We propose to employ **continuous reward metrics** within our evaluation of the allocation process.
- The reward obtained by the MDP agent in an episode of interaction with the switch environment is modelled as $r_t \in \mathbb{Z}$ where,

$$r_t \triangleq \Big(- \Big\{ \sum_{i=0}^{N_P-1} \sum_{j=0}^{N_Q-1} (\{P_i Q_j\}_{drop})_t \Big\} \Big). \tag{5}$$

- The use of continuous rewards as opposed to discrete metrics will inadvertently result in slower convergence. However, the design offsets this by employing multiple workers which not only speeds up the training, but also makes it more diverse.

2 Motivation

- As we go through the state space, the action space, the observation space, and the reward/cost metrics, we realize that we're dealing with enormous state and action spaces whose size increases exponentially with an increase in the number of ports, queues, and buffer spaces corresponding to them.
- Owing to this **curse of dimensionality**, standard MDP techniques such as Value Iteration and Policy Iteration are rendered infeasible. For enormous state spaces, considering the fact that the transition, emission, and initial steady-state models of the underlying MDP are unknown, a model-free learning technique known as **Q-Learning** can be employed.
- However, computing the Q-values for every state-action pair in this enormous state and action space environment would be impossible considering the prowess of today's computational systems. **Deep-Q-Networks** (use an Artificial Neural Network (ANN) to evaluate the Q-value for a given state-action pair and pick the action with the largest Q-value for the given input state) offer a feasible solution in this scenario, however, the presence of an enormous action space causes the number of neurons in the output layer of the ANN to explode, which increases the number of training parameters, which in turn increases the training time.

- In order to solve the problem of increased training times, we employ an **Asynchronous Advantage Actor-Critic (A3C)** architecture employing **Double Deep-Q-Networks (DDQNs)** with a **Deep Deterministic Policy Gradient (DDPG)** Optimization used within the Actor network.

3 State-of-the-Art

1. J. Schulman, et. al., "High-Dimensional Continuous Control using Generalized Advantage Estimation", October 2018
2. V. Mnih, et. al., "Asynchronous Methods for Deep Reinforcement Learning, June 2016
3. T. Schaul, et. al., "Prioritized Experiential Replay", February 2016
4. T. Lillicrap, et. al., "Continuous Control with Deep Reinforcement Learning", February 2016
5. H. Hasselt, et. al., "Deep Reinforcement Learning with Double Q-learning", December 2015
6. R. Sutton and A. Barto, "Introduction to Reinforcement Learning", 2015
7. S. Ioffe and C. Szegedy, "Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift", March 2015
8. V. Mnih, et. al., "Playing Atari with Deep Reinforcement Learning", December 2013
9. D. Bertsekas, "Dynamic Programming and Optimal Control - Volumes 1 and 2", 2012
10. L. Ornstein and G. Uhlenbeck, "On the Theory of the Brownian Motion", September 1930

4 Solution Approach

An Asynchronous Advantage Actor-Critic Approach on a standard Double Deep-Q-Networks with Prioritized Experiential Replay architecture employing Deep Deterministic Policy Gradient in the Actor Network

4.1 Asynchronous Advantage Actor Critic (A3C)

- The design employs an **Off-Policy Learning Algorithm**.
- **The Actor Network**'s output is modelled as $\mu(\vec{S} \mid \theta^{\xi})$, $\forall \vec{S} \in \mathcal{S}$ and the actor is responsible for determining the action policy to be followed by the RL agent.

- **The Critic Network**'s output is modelled as $Q(\vec{S} \mid \vec{A}, \theta^Q),\ \forall \vec{S} \in \mathcal{S}, \forall \vec{A} \in \mathcal{A}$ and the critic is responsible for determining the "goodness" of the actor's action policy. The TD error from the critic drives the actor towards the most optimal action policy.

- In an Asynchronous Actor-Critic setting, **multiple DQN workers** having their own set of network weights interact with their own copy of the switch environment. This not only speeds up the training of the overall Actor-Critic architecture, but also makes the training more diverse because each worker logs in a myriad of experiences with the switch environment.

- The design incorporates a **Prioritized Experience Replay Buffer** to remove correlations among training data samples while training the neural networks and thereby facilitating faster convergence. Instead of sampling random experiences from the replay memory, the design samples experiences according to two different sampling variants - *TD-Error Prioritization* and *Stochastic Prioritization* [3]. We then compare the effort needed arrive at optimal solutions for the buffer allocation algorithm for these two experience sampling variants.

- The TD target for the Critic in episode t considering $\vec{S}_t \in \mathcal{S}$ and $\vec{A}_t \in \mathcal{A}$ is given by,

$$\psi_t \triangleq r_t + \gamma Q(\vec{S}_{t+1}, \xi(\vec{S}_{t+1}) \mid \theta^Q). \tag{6}$$

- The loss function of the Critic is determined by sampling experiences in a batch $M \equiv \{(\vec{S}_k, \vec{A}_k, r_k(\vec{S}_k, \vec{A}_k), \vec{S}'_k, \Gamma) \mid k \in \{0, 1, 2, \ldots, |M| - 1\}\}$ and determining the mean-square error between the target and the prediction (**off-policy**) as shown below.

$$L(\theta^Q) \triangleq \frac{1}{M} \sum_{k=0}^{|M|-1} (\psi_k - Q(\vec{S}_k, \vec{A}_k \mid \theta^Q))^2 \tag{7}$$

  Note that Γ refers to a flag indicating whether the allocation process has reached a terminal state or not - a design consideration. Also, note that the model is completely agnostic to the transitions (including the transitions to the terminal state).

- The loss function of the Critic can be optimized using optimizers from TensorFlow (**AdamOptimizer** or **GradientDescentOptimizer**). The optimization update step considering a simple GradientDescentOptimizer is,

$$\theta^Q_{n+1} = \theta^Q_n + \omega \nabla L(\theta^Q_n) \tag{8}$$

  where, ω represents the step-size which can be a fixed or a varying step-size, i.e. $\frac{1}{n}$. Note that ω is also known in AI literature as **the learning rate**.

- The Actor Network is optimized using a **Deep Deterministic Policy Gradient** technique wherein the update rule is obtained as shown below. The Bellman equation [4] helps us determine the value of a state-action pair termed **Action-Value Function** corresponding to a given policy $\pi : \mathcal{S} \rightarrow \mathcal{A}$ as defined by,

$$V^{\pi}(\vec{S}_t, \pi(\vec{S}_t)) = \mathbb{E}_{\vec{S}_{t+1}, r_t} \Big[r_t + \gamma \mathbb{E}_{\vec{A}_{t+1} = \pi(\vec{S}_{t+1})} \Big[V^{\pi}(\vec{S}_{t+1}, \vec{A}_{t+1}) \Big] \Big]. \quad (9)$$

In Q-Learning which is an off-policy learning procedure, the target policy is a greedy one, i.e. $\xi(\vec{S}_{t+1}) = \text{argmax}_{\vec{A} \in \mathcal{A}} \ Q(\vec{S}_{t+1}, \vec{A})$, we can write equation (9) as,

$$Q^{\xi}(\vec{S}_t, \xi(\vec{S}_t)) = \mathbb{E}_{\vec{S}_{t+1}, r_t} \Big[r_t + \gamma Q^{\xi}(\vec{S}_{t+1}, \xi(\vec{S}_{t+1})) \Big]. \quad (10)$$

Taking the gradient in equation (10) with respect to the Actor Network parameters θ^{ξ} and changing the expectation into an expectation over a separate behavior policy Ξ, we get

$$\nabla_{\theta^{\xi}} \xi = \mathbb{E}_{\Xi} \Big[\nabla_{\theta^{\xi}} \Big(Q^{\xi}(\vec{S}_t, \xi(\vec{S}_t) \mid \theta^{Q}) \Big) \Big] \quad (11)$$

where, θ^{Q} is incorporated because the design includes a Critic Network that determines the Q-value of state-action pairs estimating the "goodness" of the pair.
Furthermore, equation (11) can be written as,

$$\nabla_{\theta^{\xi}} \ \xi = \mathbb{E}_{\Xi} \Big[\nabla_{\theta^{\xi}} \Big(\xi(\vec{S}_t) \Big) \cdot \nabla_{\xi(\vec{S}_t)} \Big(Q^{\xi}(\vec{S}_t, \xi(\vec{S}_t) \mid \theta^{Q}) \Big) \Big]. \quad (12)$$

The optimization update step at the Actor is given by,

$$\theta^{\xi}_{n+1} = \theta^{\xi}_{n} + \omega \Big(\nabla_{\theta^{\xi}} \ \xi \Big) \quad (13)$$

which can be written as,

$$\theta^{\xi}_{n+1} = \theta^{\xi}_{n} + \omega \Big\{ \mathbb{E}_{\Xi} \Big[\nabla_{\theta^{\xi}} \Big(\xi(\vec{S}_t) \Big) \cdot \nabla_{\xi(\vec{S}_t)} \Big(Q^{\xi}(\vec{S}_t, \xi(\vec{S}_t) \mid \theta^{Q}) \Big) \Big] \Big\}. \quad (14)$$

- The DDPG A3C architecture is constructed in a standard **Double Deep-Q-Networks (DDQNs)** framework: one ANN is employed to estimate the Q-values for the state-action pairs, while another ANN is employed to select the best action, given the state. The separation of the selection and estimation steps is done within the design in order to mitigate the over-estimation of Q-values which is found to occur in conventional DQNs [5].

- **Batch Normalization** layers are incorporated into the design of both the Actor and Critic Networks in order to ensure that each dimension across the samples in a batch have unit mean and variance which in turn mitigates the covariance shift during training and ensures that the subsequent layers receive whitened input data [4].

- As mentioned earlier, the design incorporates a separate network to determine the target value. The target network is updated using a soft update step as given by

$$\theta' \longleftarrow \tau\theta + (1-\tau)\theta' \tag{15}$$

where, $\tau << 1$ is the **target tracking coefficient**. This soft target update greatly stabilizes the learning process and mitigates divergence of the critic network's weights during training [4].

- To allow for exploration within the design, we incorporate two exploration strategies:
  - A *Decaying Exploration Coefficient* employed in an ϵ-greedy action selection process
  - *System-Generated Ornstein-Uhlenbeck Noise* $\mathcal{U}$ is added to the actor policy ξ to create the exploration policy ξ' as shown below.

$$\xi'(\vec{S_t}) = \xi(\vec{S_t} \mid \theta^{\xi}) + \mathcal{U}. \tag{16}$$

  Given these two exploration strategies, performance evaluations are conducted to understand the best exploration policy in the given switch environment.

- **Generation of Ornstein-Uhlenbeck Noise**:
  Vasicek Model of the Ornstein-Uhlenbeck Process:

$$du_t = \phi(\nu - u_t)dt + \sigma dW_t \tag{17}$$

  where, ν is a constant, $\phi > 0$, $\sigma > 0$, and W_u represents a Brownian Motion process.
  Rewriting equation (17), we get

$$u_{t+\delta} = u_t + \phi(\nu - u_t)dt + \sigma dW_t. \tag{18}$$

  Note that the Wiener process, i.e. the Brownian Motion process W_t has **independent increments**: $W_{p+q} - W_p,\ p > 0$ and $q \geq 0$ is independent of $W_m,\ m < t$ and **Gaussian increments**: $(W_{p+q} - W_p) \sim \mathcal{N}(0, q)$ which can be written mathematically as,

$$dW_t \sim \mathcal{N}(0, dt) = \sqrt{dt}\mathcal{N}(0, 1). \tag{19}$$

  Using equation (19) in equation (18),

$$u_{t+\delta} = u_t + \phi(\nu - u_t)dt + \sigma(\sqrt{dt}\mathcal{N}(0, 1)). \tag{20}$$