POLITECNICO
MILANO 1863

SCUOLA DI INGEGNERIA INDUSTRIALE
E DELL'INFORMAZIONE

Executive Summary of the Thesis

Experimental Assessment of Deep Reinforcement Learning Assisted Optical DC/HPC Network Reconfiguration Methods

Laurea Magistrale in Telecommunications Engineering - Ingegneria delle Telecomunicazioni

Author: Massimiliano Sica

Advisor: Prof. Massimo Tornatore

Co-advisor: Prof. S.J. Ben Yoo

Academic year: 2021-2022

1. Introduction

Data center (DC) and high performance computing (HPC) networks are at the roots of any cloud computing system and are responsible for interconnecting efficiently the different parts of a system architecture. In the recent years cloud computing has seen an impressive growth with services like AWS, Azure and Google Cloud Platform becoming the standard for almost any tradition IT service. In particular, the wide diffusion of machine learning has led to an increase in workload for the current data center systems which are now dealing with different types of traffic, with different quality-of-service requirements and an increased number of demands. Current data center networks rely on over provisioned static links which are designed to handle worst case scenarios. The current approach not only is very expensive to maintain, but is also inefficient since most of the current data center traffic is highly unpredictable and prone to oscillations. One of the most promising solutions is optical switch reconfiguration, which allows to provision paths depending on the current network situation with very fast switching times if compared to a classical electrical switch.

To drive optical switch reconfiguration several integer linear programming and heuristic methods have been tried, however they tend to show limited scalability and poor generalization capabilities. To solve the above problem I am going to present a deep reinforcement learning (DRL) [4] based optical reconfiguration method using the experimental test bed in Fig. 1. Deep reinforcement learning allows to learn patterns that humans would not be able to find, and generalize to different scenarios without explicit training. The specific goal is to show that optical reconfiguration can indeed improve the training performance of distributed machine learning (DML) workloads in case of network congestion. To give an idea of how relevant this goal is in [3] the authors claim that "Since 2012, the amount of compute in the largest AI training jobs has been increasing exponentially with a 3.4-month doubling time, 50x times faster than the pace of Moore's Law". By setting up the test bed with the proper number of servers, a real time network monitoring system and a routing algorithm I was able to show a 5x training time decrease for the deployed distributed computer vision algorithm. In addition using a self-supervised mod-

ule I was able to improve the convergence of the DRL agent and visit the failure state 29% less times.

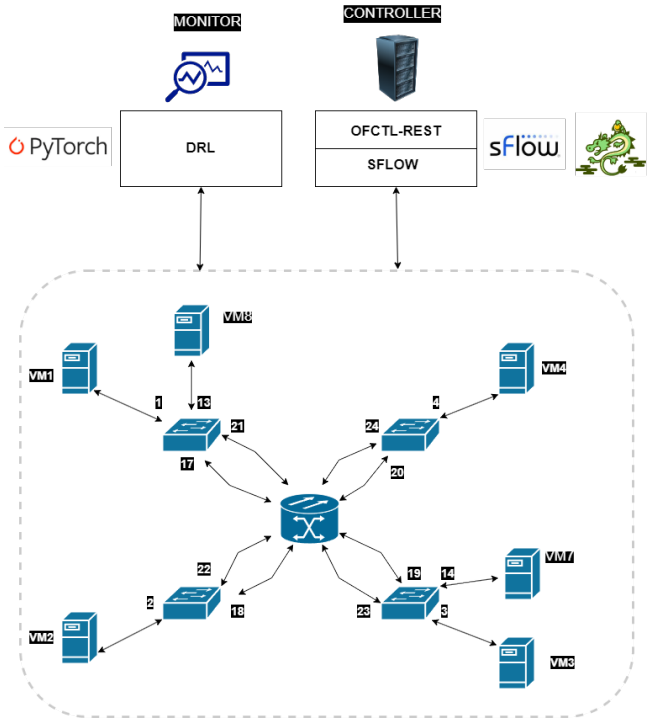

Figure 1: System Architecture

2. Current State of the Art

The problem of optical reconfiguration has been treated in literature in multiple occasions. The initial approach taken by most researchers was to write an integer linear programming (ILP) for reconfiguration, however the complexity of these methods does not scale efficiently with the size of the current data center infrastructure, as pointed out in [5]. Also heuristic algorithms have been proven to be non adequate to solve the task since they are often very far from optimal [5]. In this context, people started exploring machine learning based solutions that have the potential to scale properly without considerable human intervention. The two main papers that inspired my work are [5] and [2]. Diving deeper in [5], the authors were able to develop a convolutional neural network architecture capable of finding an optimal or near optimal reconfiguration scheme given as input the current demand matrix and the topology of the network. The overall architecture ends up being quite complex and based on three separate modules responsible for:

- Scoring a traffic demands matrix and topology couple
- Labeling the historical traffic traces with topologies with high scores
- Mapping module to actually map a demand matrix to a topology

In [2] the authors developed a DRL based framework for avoiding Quality-of-Service (QoS) deterioration of applications running within a data center. Whenever a certain application's QoS deteriorates the DRL is triggered and the workloads on the overloaded links are reconfigured. QoS is defined in terms of throughput, latency and packet loss combined together in:

$$QS(t) = T_p - \sum_{s=1}^{S}(k_1^2 La_s + k_2^s Pl_s) \qquad (1)$$

Where T_p is the throughput, La_s is the latency and Pl_s is the packet loss of reconfigured workloads at time t with QoS s, while k_1 and k_2 are weighted factors. The authors make use of OpenFlow to periodically poll the SDN agents and gather the statistics needed for computing the QS function and for checking the QoS status. The authors tested their application using some real world workloads belonging to different categories with very different needs (online serving, scientific computing, offline backup) and obtained significant results up to 6.9% network latency improvement with respect to heuristic reconfiguration methods.
However, DRL itself and in particular Deep Q-Learning tends to follow a trial and error approach which leads to many failures during training and a very long, data-hungry training. To be more specific a failure in our case could mean a network collapse. So the authors of [1] have developed a self-supervised reversibility-aware algorithm that can help improve the training performance of the agent and lead to less visits to the failure state and faster convergence.

3. Solution Development

The algorithm that I developed for this master thesis is based on multiple components:

- Workloads: Iperf and distributed machine learning
- Traffic monitoring
- Routing
- Deep Q-learning agent

The distributed machine learning algorithm is the main workload for which we want to optimize the training time, the Iperf traffic instead is used to congest the network and trigger the reconfiguration. For this experiment I am using a 9 Gbit/s UDP Iperf traffic because TCP traffic is not suitable for congesting the network because of congestion control. The DML is deployed over VM1, VM2, VM3 and VM4 which are shown in Fig. 1. VM7 and VM8 are used respectively as iperf server and client.
In order to route the traffic in the network and understand the level of congestion of each link, I had to make use of a network monitoring technology. Sflow ended up being the best tool for the task since it can be easily configured inside the top of the rack switches and in the controller and offers a rest API which allows to query multiple traffic metrics in real time. By means of these metrics I estimate the current IP to IP traffic matrix in bits per second.
Now that the traffic matrix has been estimated using Sflow, the top of the rack switches need to know where to route the incoming packets. To do so I developed a routing algorithm that for every node pair combination finds the 5 shortest paths and picks the one with more bandwidth available. Once the path is chosen I make use of the Ryu ofctl rest API running in the controller to send OpenFlow messages to install correct flow tables in the switches.
Last but not least the Deep Q-Learning-based agent has been developed in order to generate a new topology whenever the link congestion levels go above 80% for any link where the distributed machine learning algorithm is running. The new topology is generated by re-configuring the optical circuit switch shown as a cylinder in figure 1. The agent takes as input the current state represented by the number of congested links where the DML is running and outputs a score for each available action. The state space is made up by 4 states (see figure 4) which are one-hot encoded before being fed to the agent's neural network. Once we have the scores for every action we pick the one with the highest one and send a reconfiguration command to the optical switch via a TCP socket using SCPI. The action space is made up by 4 actions a0,a1,a2,a3 which allow for mobility between the different states (see figure 4)

The agent structure is as follows:
1. input layer: 4x15 ReLU activated
2. normalization layer
3. hidden layer: 15x30 ReLU activated
4. normalization layer
5. output layer: 30x4 ReLU activated

The whole algorithm is represented by the flowchart in Fig. 2

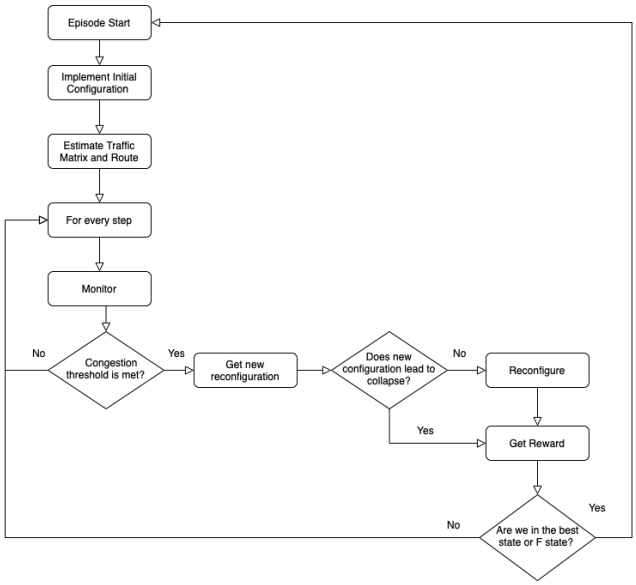

Figure 2: Algorithm Flow Chart

The first step is to set up the initial optical switch configuration and flow tables needed to start the DML. The initial configuration is state 3 in the Markov chain represented in Fig. 4 and its topology is shown in Fig. 3. The reason why there are three congested links is because a 9Gbit/s Iperf is generated between VM8 and VM7 and it is going to congest all the links in the network where the DML is running. Given that the DML follows a ring all-reduce communication pattern ($0 \rightarrow 3 \rightarrow 2 \rightarrow 1 \rightarrow 0$) all the 3 links are going to be at the same time used by the DML and congested by the Iperf. The second step is to estimate the traffic matrix using the metrics accessible through the Sflow rest API and to install the flow tables on the different top of the rack switches. Now that the initial setup is done I start monitoring the status of the links and if more than 80% of the bandwidth is used over a link occupied by DML traffic I trigger reconfiguration. The new reconfiguration schema is chosen by the DRL agent from the ones available in the action space which allow to move to either state 3, 2, F or 0 (the optimal

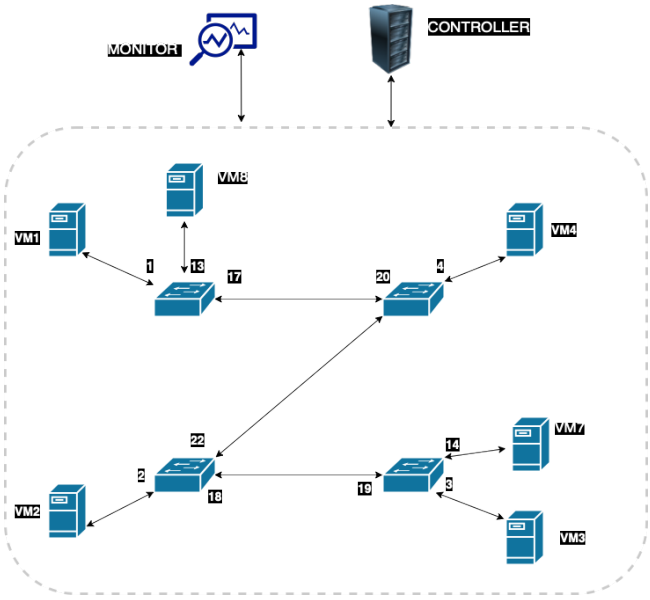

Figure 3: State 3

one). However, the action leading to the F state leads to a network failure. If that specific action is chosen I am going to ignore the action and collect a reward of -0.125 without actually implementing it. The reason is that I want to avoid unnecessary network crashes which may create problems at the application level (the DML does not restart properly). Instead if the action is legit then I collect the reward using the formula in Eq.3
In Fig 4, the Markov chain representing my decision process is shown. Every state represents the number of congested links where the machine learning algorithm is sending traffic. It is important to consider only the links being used by the DML since the overall performance of the project depends only on its completion time. The reward function A(t) where t is a specific time step of the episode is defined as:

$$A(t) = non\_conforming\_dml(t) \\ \_ \frac{non\_conforming\_dml(t+1)}{total\_links} \quad (2)$$

From Fig. 5 we can see that convergence is reached around epoch 250. It is worth reminding that 0.125 is the best reward attainable by the agent while -0.125 is the worst. In Fig. 6 we can see the evolution of the agent's training loss changing along the x-axis which represents the different training epochs. The agent is trained every five episodes. An episode is terminated if either the F state or the 0 state are visited or 10 reconfigurations have been implemented.

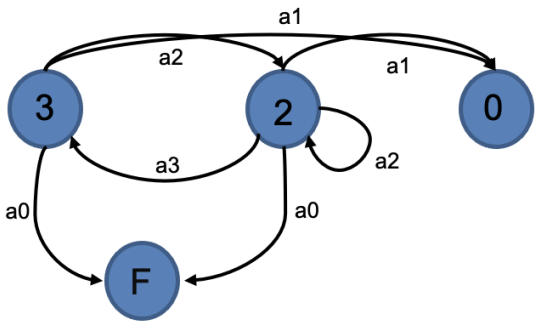

Figure 4: Markov Chain

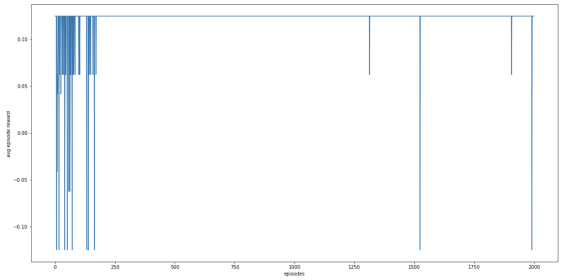

Figure 5: Rewards per episode

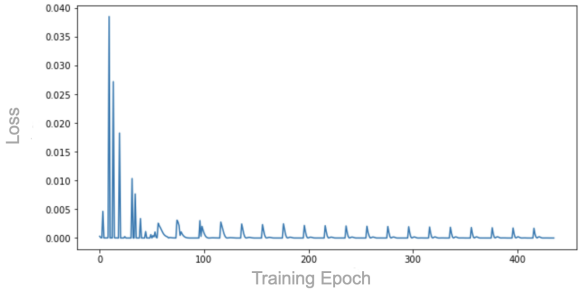

Figure 6: DRL loss evolution

4. Conclusions

For this thesis I developed a deep reinforcement learning based optical reconfiguration algorithm to improve the training time of a distributed machine learning algorithm running over 4 nodes in an experimental test bed in presence of network congestion. I was able to demonstrate a 5x times training time improvement by generating a new topology via optical switch reconfiguration and

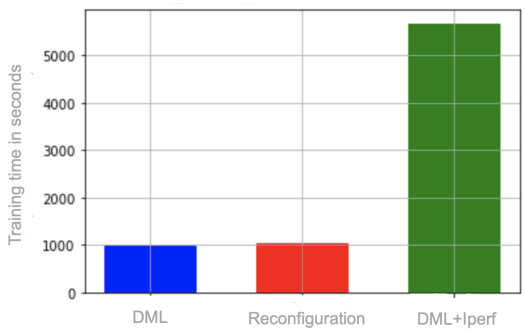

Figure 7: DQN performance over 20 epochs

proper routing to separate completely the DML from the congesting traffic flow.
Fig. 7 shows the comparison in terms of training time between the DML running on the virtual machines without any other kind of disturbance (blue), the DML running with Iperf and optical reconfiguration enabled (red), and the DML running with Iperf without optical reconfiguration enabled (green). The actual values are:

1. Only DML : 9778 seconds (16 minutes)
2. DML and 9 Gbit/s UDP Iperf: 5676 seconds (94 minutes)
3. Reconfiguration: 1026 seconds (17 minutes)

The result is very relevant since it provides a proof of concept that distributed machine learning workloads can have their training time improved thanks to optical reconfiguration. However, the training of our agent can be improved using the theory in [1].
First of all we need to update the reward function to:

$$A(t) = non\_conforming\_dml(t) - \frac{non\_conforming\_dml(t+1)}{total\_links} - SS \quad (3)$$

Where SS is a variable estimated by the reversibility network which is defined as:

- one fully connected layer 4x10 ReLu activated
- one fully connected output layer 20x1 Sigmoid activated

SS is representing the degree of reversibility of each action, and its goal is to penalize actions which are irreversible (for example going to the F state has to be strongly penalized). One may argue that terminating the episode both when state F or 0 is reached is going to teach the agent to avoid the optimal action leading to state 0 (optimal state). To avoid this issue the SS penalty is not applied to the action leading to the optimal state.
The comparison between the regular DQN agent and the self-supervised aided one is shown in Fig. 8, and the comparison in terms of visits to the failure state is reported in Fig. 10. In Fig. 9 the loss of the self-supervised network is shown using the same x-axis as Fig. 6, since the DRL and the self supervised network are trained together. It is worth pointing out that the self-supervised loss is not showing the ideal behavior (smooth convergence to a value), despite decreasing and converging around the value of 0.2, the convergence is not very stable. This could be a starting point for a future investigation.

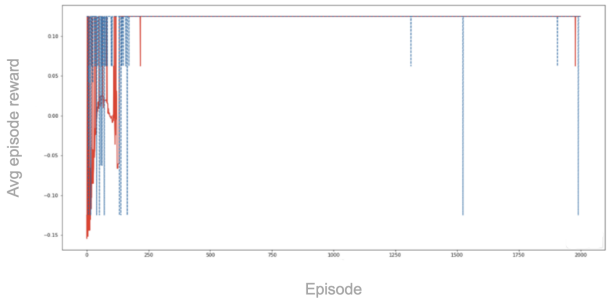

Figure 8: Reward evolution DQN vs reward evolution DQN with self-supervised module

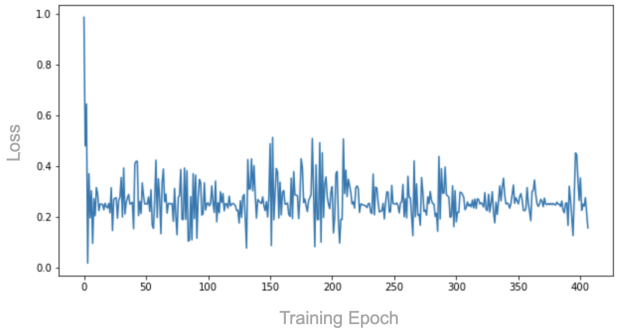

Figure 9: Self-supervised loss

The main limitations of my works are related to scalability both in terms of number of nodes and applications running in the test bed. A future

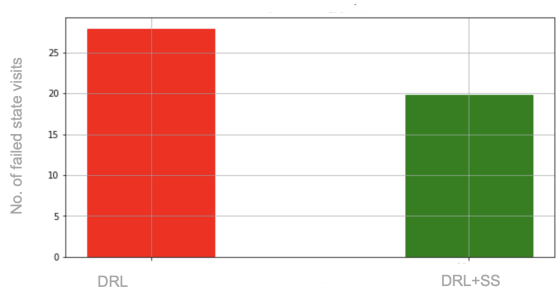

Figure 10: Failure state visits per algorithm

improvement could be to deploy other applications on the test bed (Hadoop, media streaming etc...) to check the performance variations, and to increase the state space by simulating my algorithm over an actual data center network to deal with the possible scalability issues that may arise from using DQN.

5. Acknowledgements

I would like to thank Prof. Massimo Tornatore who helped me out during my Master's and gave me the possibility to go to pursue my thesis at UC Davis. A special thanks goes to Dr. Sandeep Kumar Singh who followed me closely during my work in Davis and taught me how to make it through my first research experience. I would also like to thank Prof. Roberto Proietti and Prof. S.J. Ben Yoo for hosting me in their lab and providing me with funding for my time in California.
On a personal level I would like to thank my family for supporting me throughout my entire journey in both the bachelor's and the master's. I could not have reached this goal without your support.
A special thanks goes to U.U.
To conclude I would like to thank my university colleagues with whom I have shared ups and downs and long days and nights working on a multitude of projects. Thanks to your friendship and positive attitude I was able to keep myself strong and focused throughout my studies.

References

[1] Nathan Grinsztajn, Johan Ferret, Olivier Pietquin, Matthieu Geist, et al. There is no turning back: A self-supervised approach for reversibility-aware reinforcement learning. *Advances in Neural Information Processing Systems*, 34:1898–1911, 2021.

[2] Xiaotao Guo, Fulong Yan, Xuwei Xue, Bitao Pan, George Exarchakos, and Nicola Calabretta. Qos-aware data center network reconfiguration method based on deep reinforcement learning. *Journal of Optical Communications and Networking*, 13(5):94–107, 2021.

[3] Mehrdad Khani, Manya Ghobadi, Mohammad Alizadeh, Ziyi Zhu, Madeleine Glick, Keren Bergman, Amin Vahdat, Benjamin Klenk, and Eiman Ebrahimi. Sip-ml: high-bandwidth optical network interconnects for machine learning training. In *Proceedings of the 2021 ACM SIGCOMM 2021 Conference*, pages 657–675, 2021.

[4] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *nature*, 518(7540):529–533, 2015.

[5] Mowei Wang, Yong Cui, Shihan Xiao, Xin Wang, Dan Yang, Kai Chen, and Jun Zhu. Neural network meets dcn: Traffic-driven topology adaptation with deep learning. *Proceedings of the ACM on Measurement and Analysis of Computing Systems*, 2(2):1–25, 2018.