Neural Network Meets DCN: Traffic-driven Topology Adaptation with Deep Learning

Moewi Wang, Yong Cui, Shihan Xiao, Xin Wang, Dan Yang, Kai Chen, Jun Zhu
Introduction

Conventional wired data centers generally adopt a static network topology (e.g. Clos networks) leading to over-provisioning to handle worst case scenarios.

Topology-reconfigurable DCNs use network components such as Optical Circuit Switches (OCS) or Wireless Radio to build agile links that can be quickly reconfigured.

Modeling the global interactions between traffic and topology in a reconfigurable network is non-trivial, especially while considering user defined performance metrics.
xWeaver

- A traffic-driven deep learning system for learning the topology configuration in DCNs
- Uses deep learning to perform 2 tasks:
  a. Learn network traffic in DCNs
  b. Learn global interactions between traffic and topology
- Design Features:
  a. Can support optimization over conventional flow-level performance metrics and application level performance metrics
  b. Uses SCNN to automatically label high-score topologies with corresponding traffic demands
  c. Uses FPNN to capture interaction between traffic and topology configurations
Why Deep Learning?

- Heuristic approaches do not consider interactions between fixed and configurable parts of the network.

- High performance topologies for a given traffic demand share a set of critical links.

- CNNs are good at feature extraction, in this case, the critical links in the network.

---

Fig. 2. An example of traffic pattern attached to a specific topology configuration.
System Modules

Offline phase:

- **Scoring module**: Takes traffic-topology score as input and gives performance score based on optimization criteria
- **Labeling module**: Label historic traffic traces with corresponding high score topologies
- **Mapping module**: Learn the high-dimensional global mapping between traffic and topology

Online phase:

- Controller uses mapping module to periodically update OCS switch configuration
Traffic-driven training sample generation

Topology performance scoring:

- **Objective** is to learn a scoring function $\text{Score}(f,p)$ that maps topologies to scores based on a user-specified metric (for traffic trace $f$ and topology configuration $p$).
- **Neural networks** can be used to learn an approximate scoring function with tolerable accuracy loss.
- **Separate CNNs** can be used to extract features from traffic and topology since their patterns are unrelated.
High score topology sample generation

- Candidate topologies can be exponentially large for even small scale DCN
- Using high score topologies to learn traffic to topology mapping leads to better accuracy
- Use a heuristic search algorithm to generate high score topology samples
  \[ p_t = \arg \max_{p \in N_{\delta}(p_{t-1})} \text{Score}(f_t, p) \]
- Can lead to a local optimal score since topologies can have similar scores
- Beam search and random start to get out of local optimum
Traffic topology mapping learning

- Objective is to learn the mapping between input traffic demands and output topology configurations.
- Input feature extraction can be done using the already trained SCNN.
- Prior human knowledge embedding can be done using Conditional Random Fields.
- CRF input is the original output of the FPNN, while the CRF output is a new topology that is corrected by the prior human knowledge.

\[
P(y|x) = \frac{1}{\mathcal{Z}(x)} \prod_{c \in C} \phi(x, y|c)
\]

\[
\phi(x, y|c) = \prod_{c \in C} \exp(\sum_{k} \lambda_{k,c} f_{k,c}(x, y|c))
\]

- Uses MLE to find the topology \( y \) to maximize \( P(y|x) \) given the observed FPNN output \( x \) that satisfies all feature functions.
Traffic topology mapping learning
Performance Evaluation

Scoring module

Traffic-topology learning

(a) Learning performance of the optimal topologies

(b) Learning performance of the critical topology links
Performance Evaluation

Fig. 10. Learning performance for different traffic patterns.

(a) CDF of flow size over application traffic patterns
(b) Normalized CTD over application traffic patterns
(c) Optimality gaps for hot-spot traffic patterns
(d) CDF of normalized CTD over real data center traces

Fig. 11. Performance evaluation in an OCS-based testbed.

(a) OCS Testbed
(b) CTD of different traffic patterns
(c) CDF of flow completion time
(d) Throughput of reconfiguration
Scalability and Adapting to New Traffic Patterns

**Independent Learning:** FPNN is re-trained for every new traffic pattern

**Adaptive Learning:** FPNN is initialized for the first pattern and then keep updating the parameters for later traffic patterns

<table>
<thead>
<tr>
<th>Traffic pattern</th>
<th>(3, 50)</th>
<th>(3, 25)</th>
<th>(3, 0)</th>
<th>(1, 50)</th>
<th>(1, 25)</th>
<th>(1, 0)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Independent training (iter.)</td>
<td>23369</td>
<td>35783</td>
<td>20739</td>
<td>21502</td>
<td>20032</td>
<td>19341</td>
</tr>
<tr>
<td>Adaptive training (iter.)</td>
<td>23369</td>
<td>2431</td>
<td>2678</td>
<td>1440</td>
<td>2343</td>
<td>2536</td>
</tr>
</tbody>
</table>

**Table 1. Comparison of iterations required for different training processes.**

<table>
<thead>
<tr>
<th>Number of racks</th>
<th>10</th>
<th>100</th>
<th>200</th>
<th>500</th>
</tr>
</thead>
<tbody>
<tr>
<td>Running time of FPNN (CPU) (ms)</td>
<td>0.02</td>
<td>0.29</td>
<td>1.56</td>
<td>10.4</td>
</tr>
<tr>
<td>Running time of FPNN (GPU) (ms)</td>
<td>0.09</td>
<td>0.18</td>
<td>0.48</td>
<td>2.37</td>
</tr>
<tr>
<td>Running time of Weight-matching (ms)</td>
<td>0.02</td>
<td>0.15</td>
<td>0.94</td>
<td>24</td>
</tr>
</tbody>
</table>

**Table 2. Comparison of online running time over different network scales.**
Sensitivity and Robustness Analysis
Thoughts

Pros:

- Auto-labeling for training data
- Support for application level performance metrics
- Separate CNN modeling

Doubts:

- Can it optimize for multiple performance metrics at once? What if they are contradictory?
- Significant drop in throughput during reconfiguration (for about 300ms)