Atlas: Automate Online Service Configuration in Network Slicing

Qiang Liu, University of Nebraska-Lincoln
Nakjung Choi, Nokia Bell Labs
Tao Han, New Jersey Institute of Technology
Emerging Applications

- Resource competition **degrades** end-to-end performance
  - independent optimization, e.g., RAN and TN, fails in guaranteed performance
Network Slicing

- Enable customized end-to-end slice for each application
  - performance and functional isolation, SLA guarantee
  - customization in performance, function, security, etc.
Service Configuration

- Configure individual slice settings to maintain SLA
  - for example, cross-domain resources, attributes*
  - High-dim contextual states, e.g., traffic, users
  - long configuration interval, e.g., hours (non-markov)

*Generic Network Slice Template, V 7.0, GSM Association, June 2022
State-of-the-Art

- Offline approaches
  - design the policy in offline environments, e.g., simulator or dataset
  - offline approaches [1, 2] suffer simulation-to-reality discrepancy
  - the discrepancy between offline simulators and real-world networks

Online approaches

- learn the policy via interacting with real-world networks
- online ML methods [3] suffer safety and sample-efficiency issue
- sample-efficiency: long configuration interval in real-world networks
- safety: unpredictable configuration actions from DNN-parameterized policies

The first integrated offline-online network slicing

- Atlas automates service configuration of individual slices
- Atlas achieves safe and sample-efficient learn-to-configure in three integrated stages
  - **stage 1: learning-based simulator**, for reducing sim-to-real discrepancy
  - **stage 2: offline training**, for training an offline policy
  - **stage 3: online learning**, for learning the online policy
Learning-based Simulator
- **objective**: automatically reduce the simulation-to-reality discrepancy
- **action**: adjust the simulation parameters, e.g., base pathloss
- **rationale**: these parameters might not accurate enough
Stage 1

- Learning-based Simulator
  - Problem: minimize KL divergence between simulation and system measurement
  - Challenge: unknown correlation between KL divergence and high-dim simulation parameters
  - Solution: new Bayesian learning method
    - scalable Bayesian neural network
    - parallel Thompson sampling
Stage 2

- **Offline Training**
  - **objective**: offline train a policy in the augmented simulator
  - **problem**: minimize resource usage under requirement of percentile QoE
  - **challenge**: unknown correlation between slice QoE and configuration parameters
  - **solution**: constraint-aware method and Bayesian learning method
Online Learning

- **objective:** online learn the policy in real-world networks
- **rationale:** resolve the sim-to-real discrepancy eventually
- **problem:** minimize resource usage under requirement of percentile QoE
Online Learning

- **challenge**: assure safety (SLA violation) under limited online transitions
- **solution**:
  - sample-efficient GP model to learn sim-to-real gap only
  - conservative acquisition function with regret bound
  - hybrid multiplier update with both offline and online transitions
System Implementation

- **Testbed**
  - **User**: OnePlus 9 5G
  - **Agent**: PyTorch 1.5 (128x64x32)
  - **RAN**: OpenAirInterface w/ USRP (LTE B7)
  - **TN**: OpenDayLight w/ SDN switch
  - **CN**: OpenAir-CN w/ CUPS
  - **Edge**: Dockers collocated with SPGW-U

- **Virtualization**
  - **RAN**: FlexRAN (exclusive PRB assignment) + customized MCS offset
  - **TN**: OpenFlow with configurable bandwidth via “meter”
  - **CN**: isolated SPGW-U container per slice
  - **EN**: docker container via “docker update”

- **Applications**
  - Video analytics at the edge
  - send 540p image to edge server
  - the server run ORB to extract features
  - requirement: 300ms round-trip latency
Stage 1 Performance

- Atlas reduces sim-to-real discrepancy
  - obtains 81.2% discrepancy reduction under 0.12 parameter distance
  - more than 24.5% reduction than existing Bayesian optimization method (GP)

<table>
<thead>
<tr>
<th>Methods</th>
<th>Sim-to-Real Discrepancy</th>
<th>Parameter distance</th>
<th>Best simulation parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>Original Simulator</td>
<td>1.38</td>
<td>0</td>
<td>[38.57, 5.0, 9.0, 0.0, 0.0, 0.0, 0.0]</td>
</tr>
<tr>
<td>Aug. Simulator, GP</td>
<td>0.31</td>
<td>0.16</td>
<td>[38.57, 1.44, 7.48, 5.07, 9.23, 6.02, 6.47]</td>
</tr>
<tr>
<td>Aug. Simulator, Ours</td>
<td>0.26</td>
<td>0.12</td>
<td>[38.76, 0.68, 8.93, 5.03, 8.93, 2.16, 3.10]</td>
</tr>
</tbody>
</table>

Table 4: Details of offline learning-based simulator
Stage 2 Performance

- Atlas trains the policy with reduced resource usage
  - obtains up to 47.5% usage reduction than existing solutions
  - better Pareto boundary performance

Stage 3 Performance

- Atlas reduces usage and QoE regret
  - obtains up to 63.9% reduction on the regret of resource usage
  - obtains up to 85.7% reduction on the regret of slice QoE
  - results show the necessity of integrating three stages

End-to-end slicing is necessitated to assure diversified performance of slices.

We proposed Atlas, the first integrated offline-online network slicing system that automates the service configuration of individual slices.

Atlas addressed practical challenges of online machine learning, i.e., safety and sample-efficiency, by designing three interrelated stages.

We prototype Atlas in end-to-end slicing testbed with extensive performance evaluation.

GitHub: [https://github.com/int-unl/Atlas.git](https://github.com/int-unl/Atlas.git)
Simulation and Configuration Space

- **Simulation Space**
  - selected according to its impact on the sim-to-real discrepancy

- **Configuration Space**
  - selected according to its impact on the performance of slice users

- Atlas can handle more simulation and configuration space

<table>
<thead>
<tr>
<th>Configuration</th>
<th>Meaning</th>
<th>Range</th>
</tr>
</thead>
<tbody>
<tr>
<td>bandwidth_ul</td>
<td>maximum uplink PRBs</td>
<td>[0, 50]</td>
</tr>
<tr>
<td>bandwidth_dl</td>
<td>maximum downlink PRBs</td>
<td>[0, 50]</td>
</tr>
<tr>
<td>mcs_offset_ul</td>
<td>uplink MCS offset [24]</td>
<td>[0, 10]</td>
</tr>
<tr>
<td>mcs_offset_dl</td>
<td>downlink MCS offset [24]</td>
<td>[0, 10]</td>
</tr>
<tr>
<td>backhaul_bw</td>
<td>transport bandwidth (Mbps)</td>
<td>[0, 100]</td>
</tr>
<tr>
<td>cpu_ratio</td>
<td>CPU ratio of docker</td>
<td>[0, 1.0]</td>
</tr>
</tbody>
</table>

Table 2: Network configuration space

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>baseline_loss</td>
<td>base loss in pathloss model (dBm)</td>
</tr>
<tr>
<td>enb_noise_figure</td>
<td>noise by non-ideal transceivers (dBm)</td>
</tr>
<tr>
<td>ue_noise_figure</td>
<td>noise by non-ideal transceivers (dBm)</td>
</tr>
<tr>
<td>backhaul_bw</td>
<td>additional transport bandwidth (Mbps)</td>
</tr>
<tr>
<td>backhaul_delay</td>
<td>additional transport delay (ms)</td>
</tr>
<tr>
<td>compute_time</td>
<td>additional server compute time (ms)</td>
</tr>
<tr>
<td>loading_time</td>
<td>additional loading time in UE (ms)</td>
</tr>
</tbody>
</table>

Table 3: Simulation parameter space