TinyEdge
Authors

Wenzhao Zhang
Yuxuan Zhang
Hongchang Fan
Yi Gao
Wei Dong
Jinfeng Wang

Zhejiang University China. Alibaba-Zhejiang University Joint Institute of Frontier Technologies Technologies
Enabling Rapid Edge System Customization for IoT Applications: Quick Overview

1) Using a top-down approach for designing the software and estimating the performance of a customized edge system under different hardware specifications.

1) Developers select and configure modules to specify the critical logic of their interactions, without dealing with the specific hardware or software.

1) TinyEdge automatically generates the deployment package and estimates the performance after sufficient profiling.

1) TinyEdge provides a unified customization framework for modules to specify their dependencies, functionalities, interactions, and configurations.
Usage

Fig. 1. Workflow overview of TinyEdge.
Design
Customization Service
System Level Customization

User first selects modules and provides necessary configurations for them.

TinyEdge dependency checker makes sure the dependencies are satisfied.
System Level Customization: Key Techniques

In order to reduce module configuration time, TinyEdge proposes two configuration reduction methods:

1) Configuration classification
2) Global Configuration Sharing

In order to resolve fine-grained module dependency dynamically, TinyEdge proposes a dependency checking mechanism which contains checking policies and a checking algorithm.
Application Level Customization

User writes application code using TinyEdge Domain Specific Language (DSL)

TinyEdge runtime parses the code into different parts and generates message queue topics, then distribute each part to designated execution modules or engines.
Fig. 8. TinyEdge topic generation mechanism, where M represents Module, Tp represents Topic, App represents Application.
Performance Estimation Service
Module Profile

Contains key information source for the performance models

- Module category
- Customization information (functionality and configuration)
- Performance models for specific module functions

Then TinyEdge splits modules

- System modules - make up essential part of edge system like device mgmt and logging
- Processing modules - take charge of the computation part of an edge system like data filter and video analyzing
- Connecting modules - responsible for accessing IoT devices or transmitting data to the cloud
Performance models

TinyEdge chooses to model latency and workload because

1) Both general performance metrics
2) Can describe two essential edge computing features

Does not measure accuracy
Implementation
Module Selection

Conducted small investigation of 20+ edge computing related papers and 6 mainstream industrial edge computing platforms

Findings:

The main functionalities of edge systems for IoT applications include:

1) Connector for both IoT devices and cloud services -&gt; HTTP, Modbus, Bluetooth, integrate EMQ
2) Data processing (e.g. stream analytics and ML) -&gt; simple data filter and object recognition
3) Traditional database -&gt; MySQL
4) Security -&gt; device authentication module
Module Compaction, OS tools, and Runtime

1) Smaller base images

1) Optimized Dockerfile

TinyEdge runtime -> enables customized system to update

- Message queue engine and serverless engine (Apache Kafka)
- Connector (OpenFaas)
Evaluation
Evaluation Cases

1) Data connection and visualization (IoT)

1) Intelligent data processing (EI)

1) Hybrid-analysis system (GloTTO)
<table>
<thead>
<tr>
<th>Case</th>
<th>Module</th>
<th>Original</th>
<th>Optimized</th>
<th>Reduction</th>
</tr>
</thead>
<tbody>
<tr>
<td>IoT</td>
<td>(1) Virtual Device</td>
<td>930 MB</td>
<td>105 MB</td>
<td>54.10%</td>
</tr>
<tr>
<td></td>
<td>(2) MQTT</td>
<td>85 MB</td>
<td>85 MB</td>
<td>0%</td>
</tr>
<tr>
<td></td>
<td>(3) InfluxDB</td>
<td>260 MB</td>
<td>260 MB</td>
<td>0%</td>
</tr>
<tr>
<td></td>
<td>(4) Grafana</td>
<td>250 MB</td>
<td>250 MB</td>
<td>0%</td>
</tr>
<tr>
<td>EI</td>
<td>(1) Virtual Device</td>
<td>Same as above</td>
<td>Same as above</td>
<td>56.87%</td>
</tr>
<tr>
<td></td>
<td>(5) Obj Recognition</td>
<td>1980 MB</td>
<td>1150 MB</td>
<td>56.87%</td>
</tr>
<tr>
<td></td>
<td>(1) (2) (3) (4)</td>
<td>Same as above</td>
<td>Same as above</td>
<td>56.87%</td>
</tr>
<tr>
<td>GloTTO</td>
<td>(6) Device Management</td>
<td>935 MB</td>
<td>120 MB</td>
<td>87.70%</td>
</tr>
<tr>
<td></td>
<td>(7) Authentication</td>
<td>935 MB</td>
<td>120 MB</td>
<td>87.70%</td>
</tr>
<tr>
<td></td>
<td>(8) Virtual Sensor</td>
<td>1210 MB</td>
<td>390 MB</td>
<td>68.87%</td>
</tr>
<tr>
<td></td>
<td>(9) MySQL</td>
<td>380 MB</td>
<td>380 MB</td>
<td>0%</td>
</tr>
<tr>
<td>Platform</td>
<td>TinyEdge</td>
<td>Azure IoT Edge</td>
<td>EdgeX</td>
<td></td>
</tr>
<tr>
<td>---------------------</td>
<td>----------</td>
<td>----------------</td>
<td>-------</td>
<td></td>
</tr>
<tr>
<td>Component/Use Case</td>
<td>IoT</td>
<td>EI</td>
<td>GloTTO</td>
<td>IoT</td>
</tr>
<tr>
<td>Time Series Database</td>
<td>✓</td>
<td>-</td>
<td>✓</td>
<td>-</td>
</tr>
<tr>
<td>Traditional Database</td>
<td>-</td>
<td>✓</td>
<td>-</td>
<td>✓</td>
</tr>
<tr>
<td>Device Connector</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Data Visualization</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>-</td>
</tr>
<tr>
<td>Intelligent Analysis</td>
<td>-</td>
<td>✓</td>
<td>✓</td>
<td>-</td>
</tr>
<tr>
<td>Device Management</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>-</td>
</tr>
<tr>
<td>Virtual Device</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>

**Notations**
- ✓: Have the out-of-the-box component
- X: Do not have the component
- -: No need to use in the case
- *: Have similar component but needs extra effort to use

Customization related results
System level customization results

Fig. 10. System-level customization time comparison among TinyEdge, EdgeX, and Azure IoT Edge.
Dependency checking efficiency comparison results
Application-level customization results
Performance estimation related results

Fig. 13. Performance Modeling Comparison Between TinyEdge and Conventional ML Methods.
Final Results

Final results show that TinyEdge reduces customization time of edge systems, reducing 44.15% of customization time and 67.79% lines of on average while giving accurate performance estimation in various settings.
Discussion

1) Effectiveness of evaluation cases and components. Are they representative?
2) Dependency checking efficiency?
3) Irrelevant data? (# of topics vs performance and resources)
4) Effectiveness of configuration reduction methods

Most importantly:

1) What did we learn?
   ● Customization time of set up, configuration, and deployment can be reduced
   ● The number of lines of code to customize can be reduced
     ○ Follow up: Is this a valuable metric?
   ● Can more accurately estimate performance for the user

1) Does what we learned provide value for the IoT Industry?