Millenial: Modular Microservice Macrobenchmarks
Generating highly reconfigurable microservice benchmarks for systems research!

Vaastav Anand (vaastav@mpi-sws.org)
Gerd Alliu (galliu@mpi-sws.org)
Antoine Kaufmann (antoinek@mpi-sws.org)
Deepak Garg (dg@mpi-sws.org)
Jonathan Mace (jcmace@mpi-sws.org)

Background
❖ Microservices increasingly popular for cloud apps.
❖ Present a gold mine of research problems.
❖ Good research requires variety of systems.

What do researchers want?
❖ Multiple diverse systems for robust evaluation!
❖ Existing systems make choice of features (tracing, replication, etc) fixed with no flexibility.
❖ Most papers end up using limited number of systems because of high amount of effort required to test ideas on even 1 system.

GOAL: Generate implementations of microservice systems on-demand based on user requirements while providing the flexibility to enable/disable features and making it easy to integrate new components.

Challenges
❖ Flexibility: Should be easy to reuse and generate multiple implementations of the same application
❖ Extensibility: The generation process should be extensible with new features.
❖ Systematic: Generation can’t be ad-hoc.

Key Insights
❖ Abstract Application: The business logic of the app is independent of the features and impl choices.
❖ Reusable Features/Components: Features are implemented once and used many times.

Millenial Overview

Input
1. App. Spec: Core business logic of various services
2. Wiring Spec
   ♦ Implementation choices for services
   ♦ Apply add-on features like tracing, replication

Compiler
SpecParser
Wiring Parser
Type Checking
Feature Application
Parser extracts the system AST from spec
AST is the input and output for each compiler pass
Extensible as a new compiler pass has a strict interface it follows

Output
1. Source Code
2. Deployment Files

Systems as Millenial Applications (LoC)

<table>
<thead>
<tr>
<th>System</th>
<th>Original</th>
<th>Millenial Spec</th>
<th>Millenial Wiring</th>
<th>Millenial Generated</th>
</tr>
</thead>
<tbody>
<tr>
<td>DSB-SN</td>
<td>8209</td>
<td>1601</td>
<td>59</td>
<td>6012</td>
</tr>
<tr>
<td>DSB-MM</td>
<td>7794</td>
<td>1146</td>
<td>42</td>
<td>6308</td>
</tr>
<tr>
<td>DSB-HR</td>
<td>5160</td>
<td>977</td>
<td>63</td>
<td>6081</td>
</tr>
<tr>
<td>TrainTicket</td>
<td>54466</td>
<td>10264</td>
<td>166</td>
<td>45151</td>
</tr>
<tr>
<td>SockShop</td>
<td>13987</td>
<td>2015</td>
<td>40</td>
<td>7413</td>
</tr>
</tbody>
</table>

❖ Lines of Code numbers shown from an early prototype.
❖ In addition to being highly reconfigurable, Millenial application offers a significant reduction in the lines of code that a user needs to write.
❖ The large fraction of the code generated by Millenial is the “glue code” to bind the services with features such as tracing, replication, etc and concrete choices of caches, databases, and queues.

Implementation
❖ Early prototype implemented in 6K lines of Python
❖ Custom DSL for wiring.
❖ Input Spec and Output will be in Go for 2 reasons
❖ Good performance!
❖ Easy to write specs in Go!

On the Road to Evaluation
❖ Can Millenial generate equivalent replicas of existing microservice systems?
❖ Do the systems generated by Millenial have realistic performance?
❖ How easy is it to reconfigure applications with Millenial?