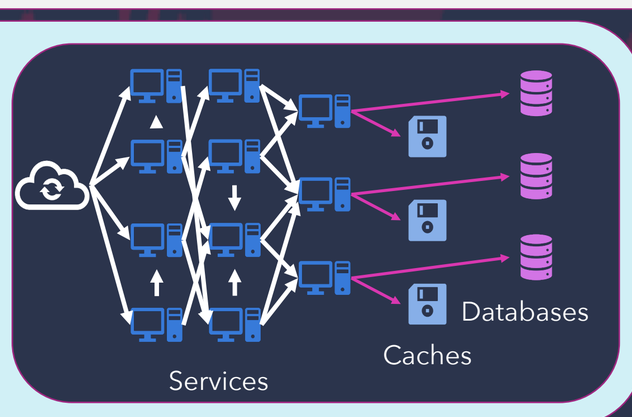
Millenial: Modular Microservice Macrobenchmarks

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

MAX PLANCK INSTITUTE FOR SOFTWARE SYSTEMS

Background

- Microservices increasingly popular for cloud apps.
- Good research requires variety of systems.
- Hard to get access to systems and change systems



GOAL: Generate configurable implementations of microservice systems on-demand based on user requirements in an extensible, flexible manner.

Researching Phenomena

- Performance or correctness issues
- Only show up at scale
- Impossible to diagnose from individual parts.
- Example: SLO violations due to low-level resource contention
- Investigating phenomena requires access to systems
- Access to systems is prohibited
- Open-source are Inflexible point solutions

Challenges

- **Flexibility:** Should be easy to reuse and generate multiple implementations
- **Extensibility:** The generation process should be extensible such that it is easy to add new features

Key Insights

- **Abstract Application**: The core 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 logic of various services
- Program against generic interfaces of common components

2 Wiring Spec
- Implementation choices for services
- Specify how services connect to each other
- Apply add-on features like tracing, replication, etc

Compiler

- Combines the application and wiring specs to generate the system IR.
- The IR encodes the source code information and the deployment information for each service
- Performs a series of passes on the system IR to generate the desired code.

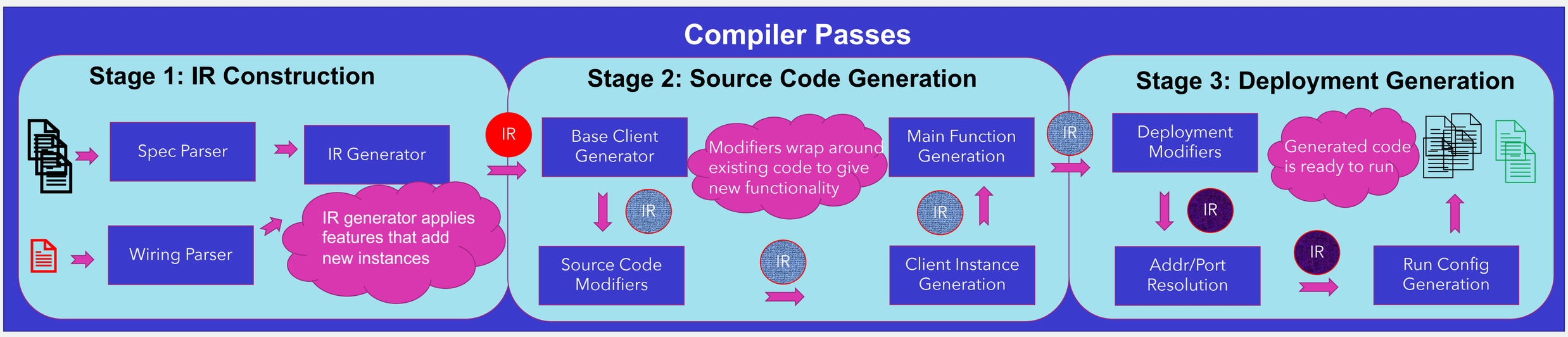
Output

1 Source Code

2 Deployment Files

- All the generated code is bundled together in go packages – one for each instance
- Each individual deployable unit is packaged independently from others
- A Workload generator is provided to test the deployment

Compiler Passes



Implementation

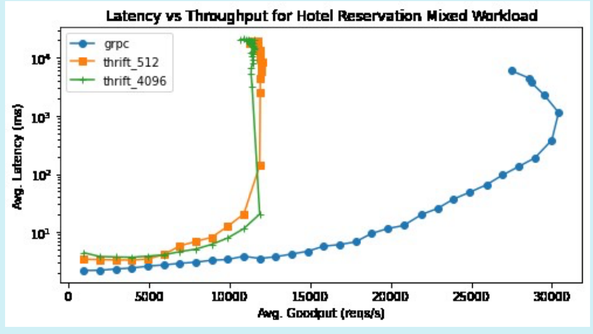
- Prototype implemented in 9774 lines of Go.
- Custom DSL for wiring is implemented in 781 lines of Python

Systems as Millenial Applications (LoC)

System	Original	Millenial Spec	Millenial Wiring	Millenial Generated
DSB-SN	8209	1478	57	8341
DSB-MM	7794	1401	42	9334
DSB-HR	5160	679	63	3675
TrainTicket	54466	9639	166	76912
SockShop	13987	2261	40	9084
Alibaba	N/A	1685939	2892	31383715

- In addition to being **highly reconfigurable**, Millenial application offers a **significant reduction** in the lines of code that a user needs to write.
- Alibaba system refers to the system we generated using Alibaba's public trace dataset

Generated System Performance



- Millenial systems are performant
- Millenial allows exploration of the design space along multiple dimensions

Extensibility

- Adding new **choices** for components requires <200 lines of code to be added to the Millenial compiler. Eg: Memcached required 161 lines of code.
- Adding new **features** to Millenial, i.e. modifiers, requires a few hundred lines of code. Eg: Adding xtrace-tracing required 421 lines of code.
- Adding new **features to an application** requires minimal changes to the wiring file. Adding xtrace to DSB-SN required 2 line changes in the wiring file.