TeaStore
A Micro-Service Application for Benchmarking, Modeling and Resource Management Research

Jóakim von Kistowski, Simon Eismann, André Bauer, Norbert Schmitt, Johannes Grohmann, Samuel Kounev

November 9, 2018
https://github.com/DescartesResearch/TeaStore
Many solutions for these questions have been proposed, however…
Challenge

How to evaluate

- Placement algorithms
- Auto-scalers
- New modeling formalisms
- Model extractors

➢ Require realistic reference and test applications

Reference applications help to

- Evaluate model (extractor) accuracy
- Measure auto-scaler elasticity
- Measure placement power consumption and performance
Requirements for a Test Application

- Scalable
- Allows for changes at run-time
- Reproducible performance results
- Diverse performance behavior
- Dependable and stable
- Online monitoring
- Load profiles
- Simple setup
- Modern, representative technology stack
Existing Test Applications

- **RUBiS [1]**
  - eBay-like bidding platform
  - Created 2002
  - Single service

- **SPECjEnterprise 2010 [2]**
  - SPEC Java Enterprise benchmark
  - Three tier architecture
  - No run-time scaling
  - Database is primary bottleneck

- **Sock Shop [3]**
  - Microservice network management demo application
  - Created 2016
  - Low load on non-network resources

- Dell DVDStore, ACME Air, Spring Cloud Demo, and more in our MASCOTS paper [4]
The TeaStore

Micro-service test application

- Five services + registry
- Netflix “Ribbon” client-side load balancer
- Kieker APM [5]
- Documented deployment options:
  - Manual
  - Docker images
  - Kubernetes
Registry
- Simplified Netflix Eureka
- Service location repository
- Heartbeat

RegistryClient
- Dependency for every service
- Netflix “Ribbon”
- Load balances for each client

WebUI
- Servlets/Bootstrap
- Integrates other services into UI
- CPU + Memory + Network I/O

Authentication
- Session + PW validation
- SHA512 + BCrypt
- CPU
Services II

**PersistenceProvider**
- Encapsulates DB
- Caching + cache coherence
  - Memory

**Recommender**
- Recommends products based on history
  - 4 different algorithms
  - Memory or CPU

**ImageProvider**
- Loads images from HDD
- 6 cache implementations
  - Memory + Disk I/O

**TraceRepository**
- AMQP Server
- Collects traces from all services
HTTP load generator [5]

- Supports varying load intensity profiles
  - Can be created manually
  - Or using LIMBO [6]
- Scriptable user behavior
  - Uses LUA scripting language
  - “Browse” and “Buy” profiles on GitHub

https://github.com/joakimkistowski/HTTP-Load-Generator
Load and Usage Profiles (2/2)

JMeter

- Commonly used load generator
- Browse profile for JMeter
- Identical to HTTP Load Generator profile
Evaluation Teaser: Does it scale?

- Scales linearly
- Stresses 144 cores on 9 physical hosts
- HTTP Load Generator handles > 6000 requests per second
Three Use-Cases

- Performance modeling
- Auto-scaling
- Measuring energy-efficiency of placements

**Goal:** Demonstrate TeaStore’s use in these contexts
Performance Model - Scenario

- Question: How does utilization change with the default \# products per page?
- Approach:
  - Create two workloads with different products per page distributions
  - Create and calibrate performance model with default distribution
  - Predict performance for
    - Different products per page distribution
    - Different service placement
Performance Model - Models

Products per Page Distribution

**Calibration**

\[ P_5(x) = \begin{cases} 
0.9 & \text{if } x = 5 \\
0.09 & \text{if } x = 10 \\
0.01 & \text{if } x = 20 \\
0 & \text{else.} 
\end{cases} \]

**To Predict**

\[ P_{10}(x) = \begin{cases} 
0 & \text{if } x = 5 \\
0.99 & \text{if } x = 10 \\
0.01 & \text{if } x = 20 \\
0 & \text{else.} 
\end{cases} \]

Deployment

**Calibration**

- Server 0: Registry
- Server 1: WebUI
- Server 2: Auth
- Server 3: Recommender
- Server 4: Image
- Server 5: Persistence

**To Predict**

- Server 0: Registry
- Server 1: W, A, I
- Server 2: W, A, I
- Server 3: W, A, I
- Server 4: W, A, I
- Server 5: W, A, I

Introduction  ➔  TeaStore  ➔  Use-Cases  ➔  Conclusion
Performance Model - Results

Results with and without considering the parametric dependency using Service Demand Law-based model
Reactive Auto-Scaling Scenario

- Challenge: Scale in an elastic manner so that \# services matches demand
- Additional Challenge: Which service to scale?
- Approach:
  - Create heterogeneous configuration order
  - Put TeaStore under varying load
  - Decide scale-up / scale-down using research auto-scaler REACT [7]
Auto-Scaling - Results

- Under- and overprovisioning-timeshare $\leq 15\%$
- TeaStore can be used for auto-scaler evaluation
- Open challenge: Which service to scale next?
Energy Efficiency - Scenario

Energy efficiency of placements

- **Goal**: Show that power consumption, energy efficiency, and performance scale differently
  - Different optima for service placements

- **Approach**:
  - Distribute TeaStore on homogeneous and heterogeneous servers
  - Put TeaStore under load using increasing stress-test load intensity
  - Measure TeaStore performance and server wall power
Energy Efficiency - Measurement

Measurements in homogeneous and heterogeneous setting

- **SUT 1:**
  - 16 core Haswell
  - 32 GB RAM

- **SUT 2 (Heterogeneous):**
  - 8 core Skylake
  - 16 GB RAM

- **Metrics:**
  - Throughput
  - Power
  - Energy Efficiency
    - Throughput / Power
Energy Efficiency – Optima for Heterogeneous Placement

**Placement Candidate 1**

- **16 cores**
  - Web UI
  - Auth
  - Img
  - Persist.

- **8 cores**
  - Web UI
  - Auth
  - Recomm.
  - Img
  - Persist.

<table>
<thead>
<tr>
<th>Metric</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Max Tr/s</td>
<td>1011.9</td>
</tr>
<tr>
<td>Max W</td>
<td>179.6</td>
</tr>
<tr>
<td>Geo Tr/J</td>
<td>4.4</td>
</tr>
</tbody>
</table>

**Placement Candidate 2**

- **16 cores**
  - Web UI
  - Auth
  - Img
  - Persist.

- **8 cores**
  - Web UI
  - Auth
  - Img
  - Persist.

<table>
<thead>
<tr>
<th>Metric</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Max Tr/s</td>
<td>1067.7</td>
</tr>
<tr>
<td>Max W</td>
<td>187.0</td>
</tr>
<tr>
<td>Geo Tr/J</td>
<td>4.3</td>
</tr>
</tbody>
</table>

Introduction  ➤ TeaStore  ➤ Use-Cases  ➤ Conclusion

TeaStore: Micro-Service Benchmarking Application
TeaStore - Conclusions

• Teastore can be used for
  ▪ Performance modeling evaluation
  ▪ Auto-Scaler evaluation
  ▪ Placement and energy-efficiency evaluation

▪ Micro-service reference application
  ▪ Five Services + Registry
  ▪ Different resource usage characteristics
  ▪ Kieker monitoring
  ▪ Load Generators and Load Profiles
  ▪ Kubernetes support

▪ Under Review by SPEC RG

https://github.com/DescartesResearch/TeaStore
Thank You!

From the TeaStore Dev Team

https://github.com/DescartesResearch/TeaStore

- HTTP Load Generator
  https://github.com/joakimkistowski/HTTP-Load-Generator

- LIMBO Load Intensity Modeling Tool
  http://descartes.tools/limbo

- Kieker Application Monitoring
  http://kieker-monitoring.net

All tools available at:

http://descartes.tools/
References


