Julius-Maximilians-
UNIVERSITÄT
WÜRZBURG

Descartes
research

# TeaStore

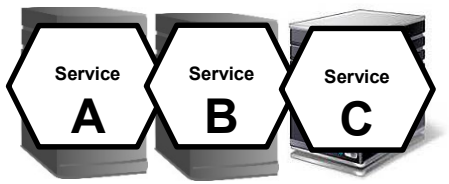
## A Micro-Service Application for Benchmarking, Modeling and Resource Management Research

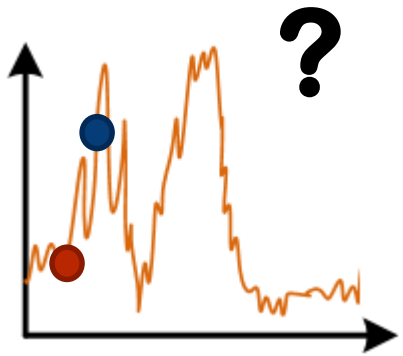
Jóakim von Kistowski, Simon Eismann, André Bauer, Norbert Schmitt,
Johannes Grohmann, Samuel Kounev

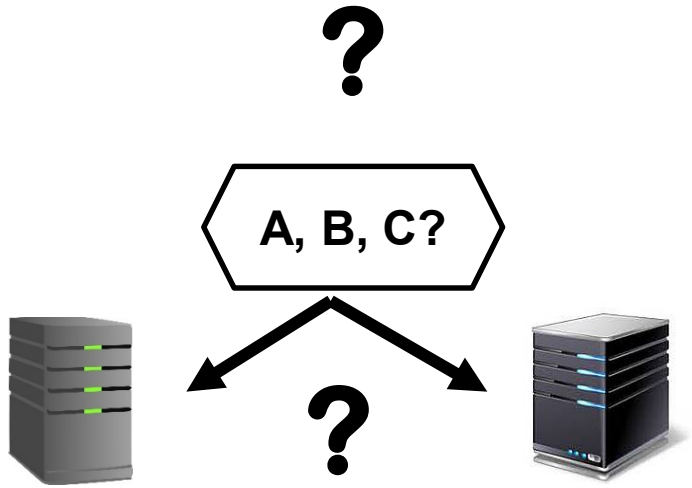
February 21, 2019

***https://github.com/DescartesResearch/TeaStore***

# Example Research Scenario




Many solutions for these questions have been proposed, however…

# Challenge

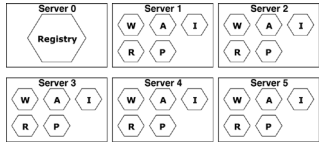
## How to evaluate

- Placement algorithms
- Auto-scalers
- Modeling approaches
- Model extractors

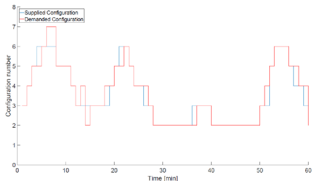
➢ Require realistic **reference and test applications**

## 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

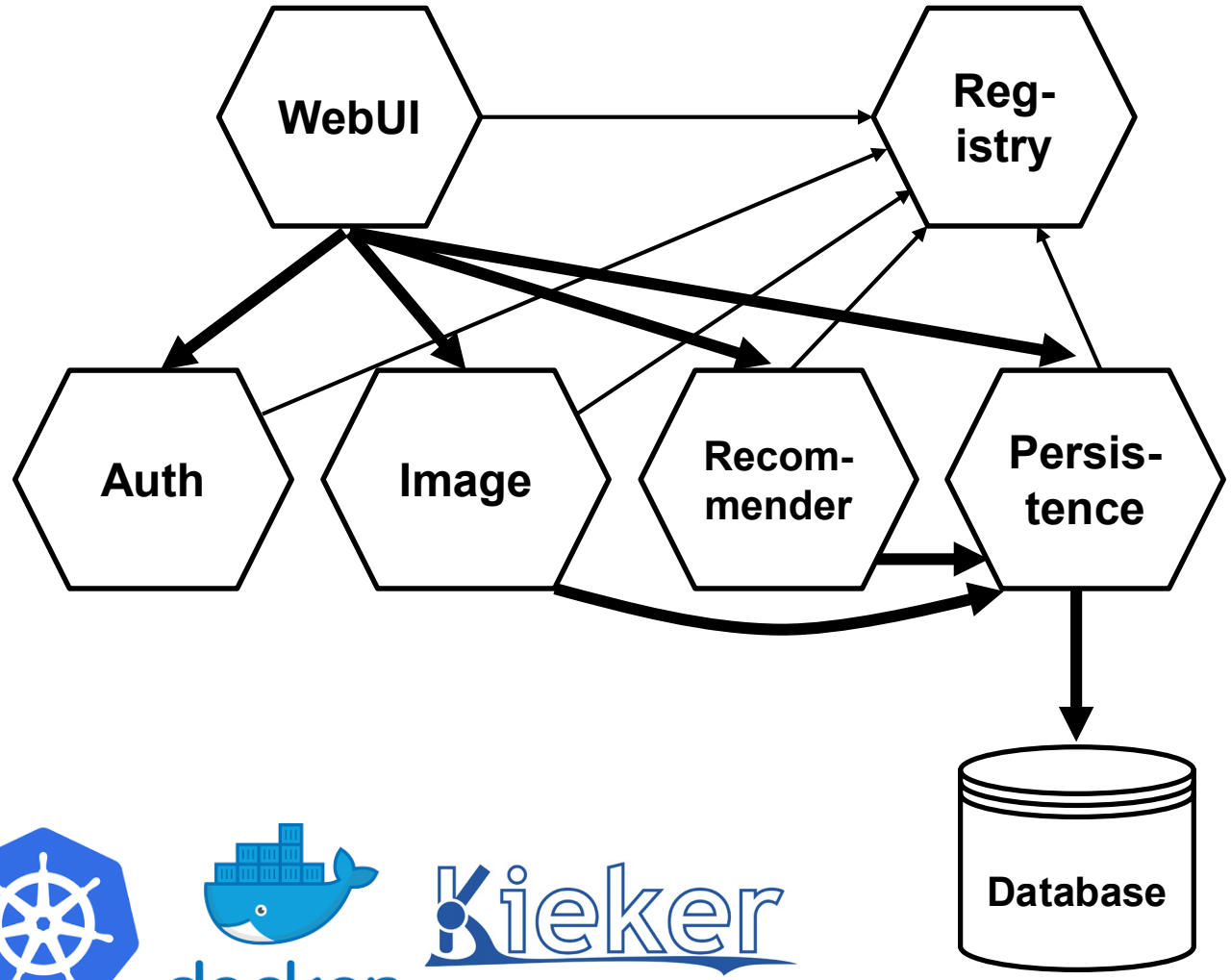
# Existing Test Applications

- 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



# Services I

## Registry

- Simplified Netflix Eureka
- Service location repository
- Heartbeat

## RegistryClient

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

## WebUI

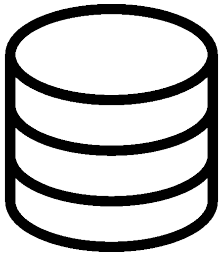
- Servlets/Bootstrap
- Integrates other services into UI
- ➢ **CPU + Memory + Network I/O**

## Authentication

- Session + PW validation
- SHA512 + BCrypt
- ➢ **CPU**

# Services II

## PersistenceProvider



- Encapsulates DB
- Caching + cache coherence

➢ **Memory**

## ImageProvider

- Loads images from HDD
- 6 cache implementations

➢ **Memory + Disk I/O**

## Recommender

- Recommends products based on history
- 4 different algorithms

➢ **Memory or CPU**

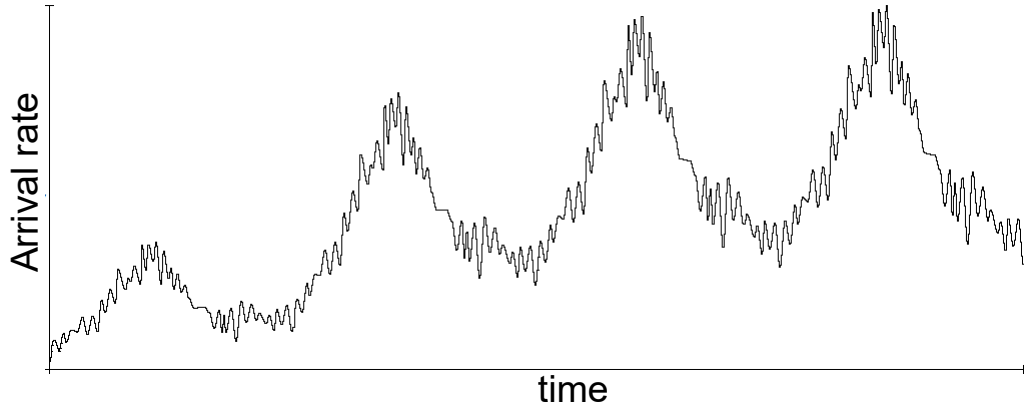
## TraceRepository

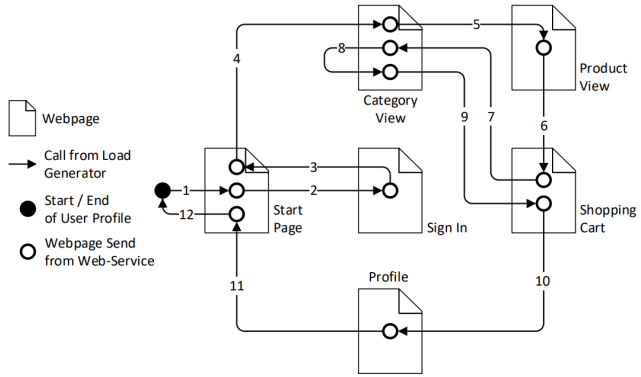

- AMQP Server
- Collects traces from all services

# Load and Usage Profiles (1/2)

## 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



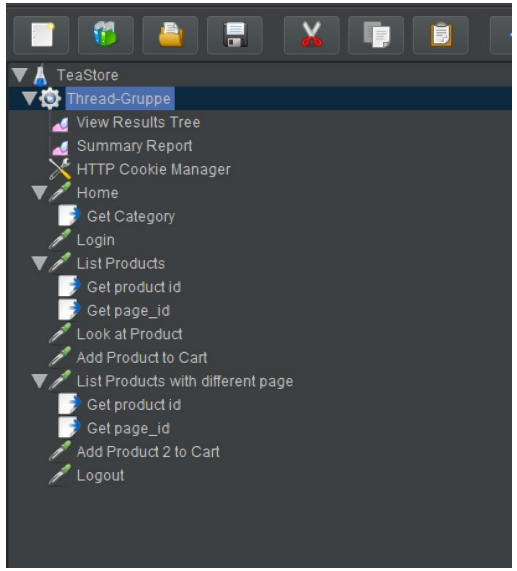



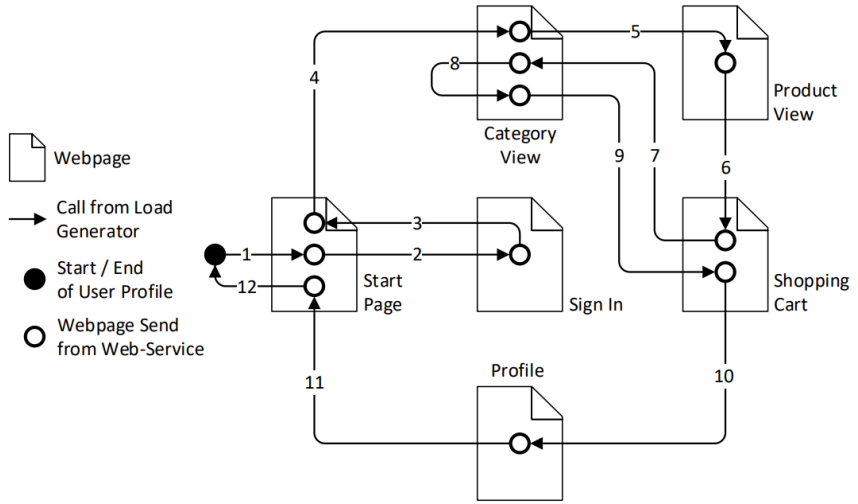

https://github.com/joakimkistowski/HTTP-Load-Generator

# Load and Usage Profiles (2/2)

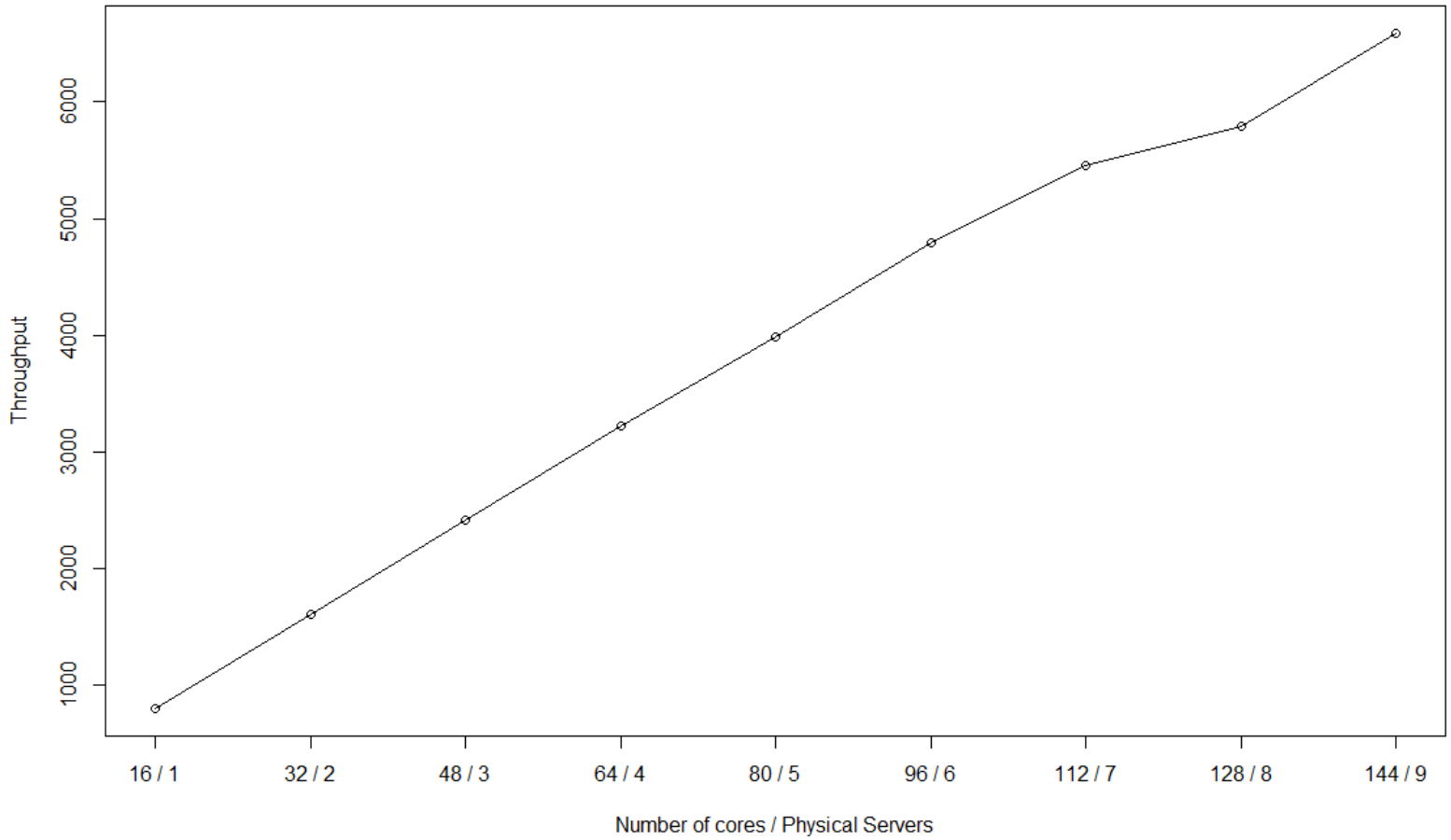
## JMeter

- Commonly used load generator
- Browse profile for JMeter
- Identical to HTTP Load Generator profile



# Evaluation Teaser: Does it scale?



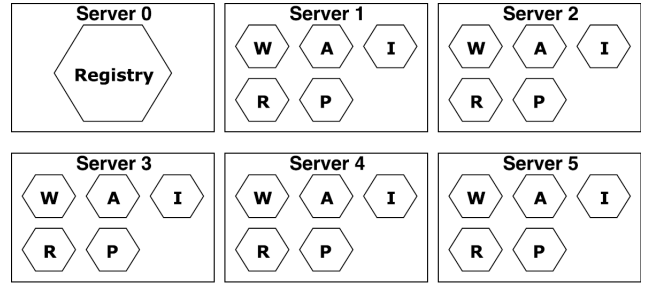

- Scales linearly
- Stresses 144 cores on 9 physical hosts
- HTTP Load Generator handles > 6000 requests per second

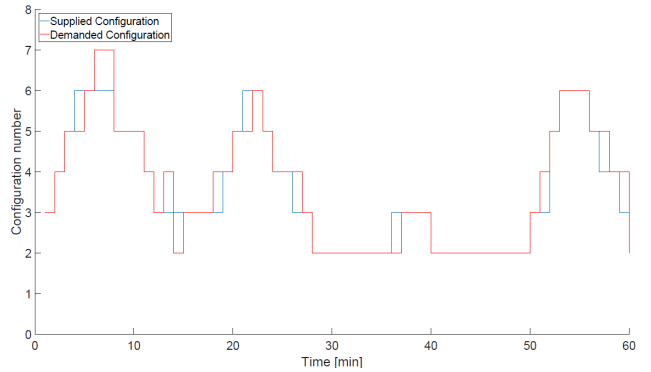
# Evaluation

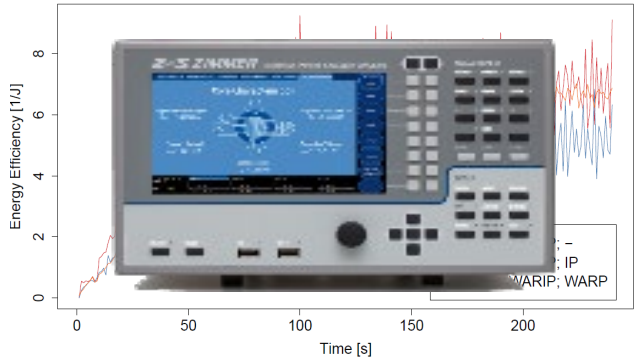
## Three Use-Cases

- Auto-Scaling
- Performance modeling
- Energy-efficiency of placements

**Goal:** Demonstrate TeaStore's use in these contexts



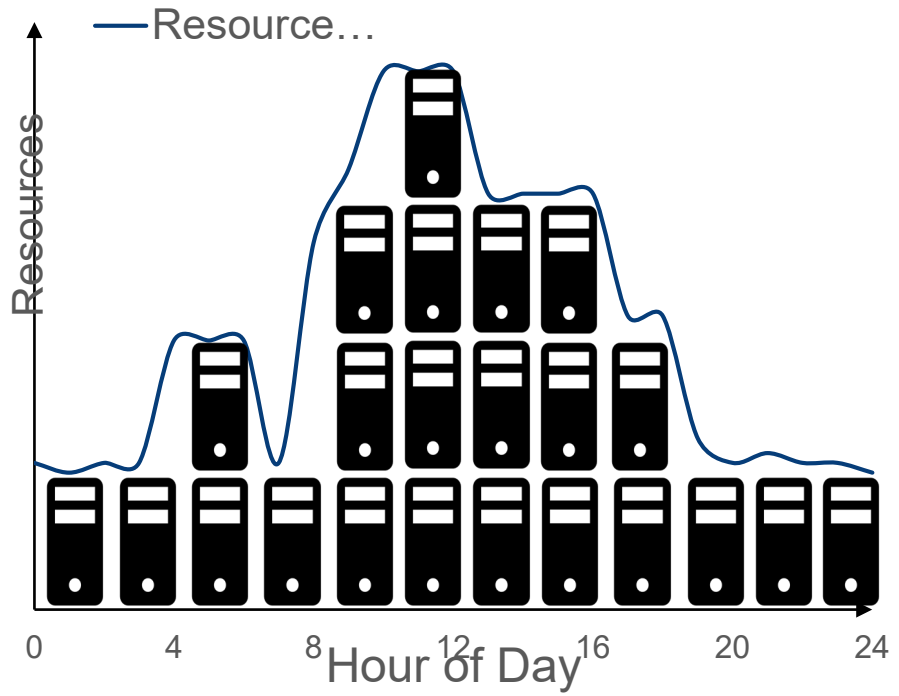



# Auto-Scaling - Scenario

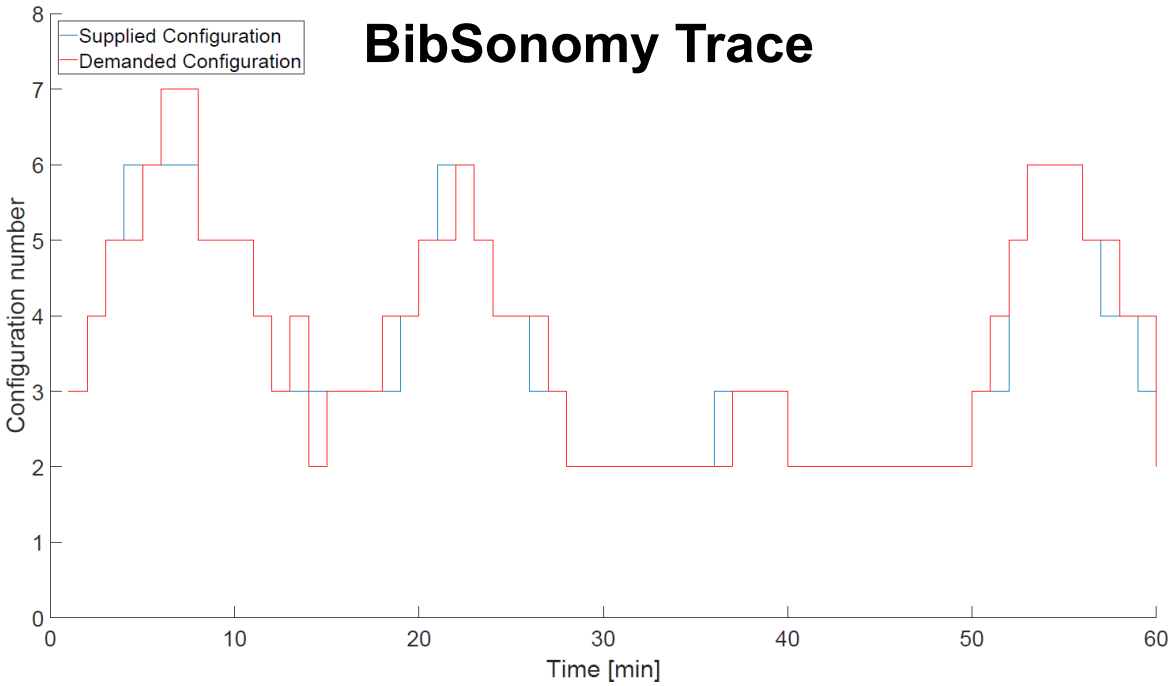
## Reactive Auto-Scaling Scenario

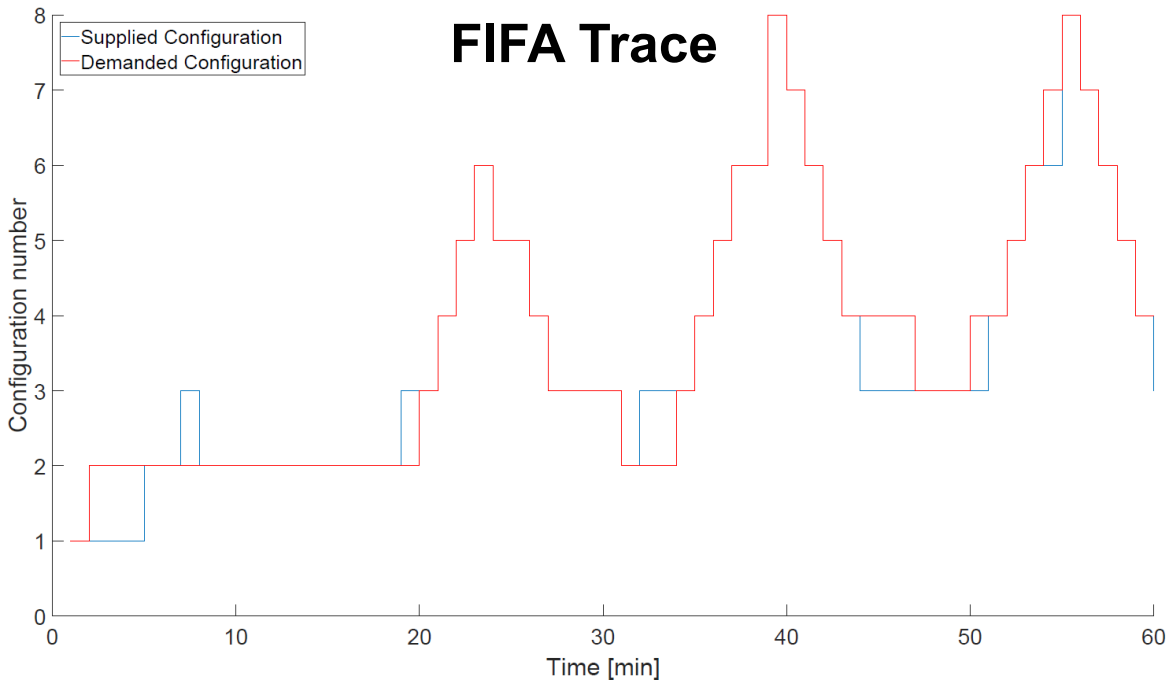
- Challenge: Scale in an elastic manner so that # *services* matches demand
- Additional Challenge: Which service to scale?
- Approach:
  - Create heterogeneous configuration order
  - Put TeaStore under varying load
  - Decide scale-up / scale-down using research auto-scaler REACT [7]




| Service | Configuration #1 | #2 | #3 | #4 | #5 | #6 | #7 | #8 | #9 |
|---|---|---|---|---|---|---|---|---|---|
| WebUI | 1 | 1 | 2 | 3 | 3 | 4 | 5 | 5 | 6 |
| Image Provider | 1 | 1 | 2 | 3 | 3 | 4 | 5 | 5 | 6 |
| Authentication | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
| Recommender | 1 | 1 | 1 | 2 | 2 | 2 | 3 | 3 | 3 |
| Persistence | 1 | 2 | 2 | 3 | 4 | 4 | 5 | 6 | 6 |

# Auto-Scaling - Results

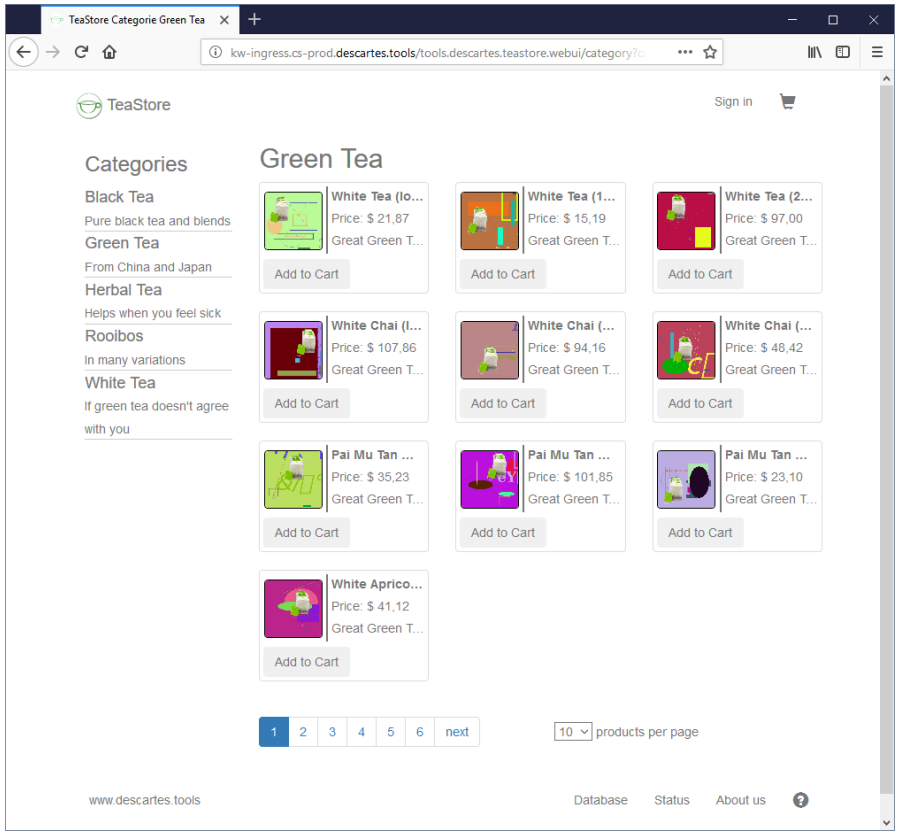



- Under- and overprovisioning-timeshare <= 15%
- TeaStore can be used for auto-scaler evaluation
- Open challenge: Which service to scale next?

# Performance Model - Scenario

- Question: How does utilization change with the default *# products per page* ?
- Approach:
  - 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

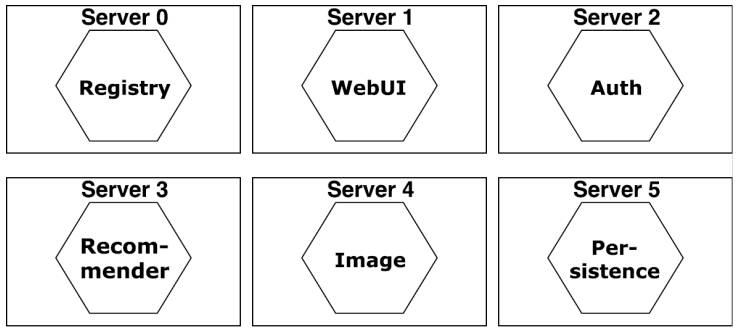
$$P_5(x) = \begin{cases} 0.9 & \text{if } x = 5 \\ 0.09 & \text{if } x = 10 \\ 0.01 & \text{if } x = 20 \\ 0 & \text{else.} \end{cases}$$

### To Predict

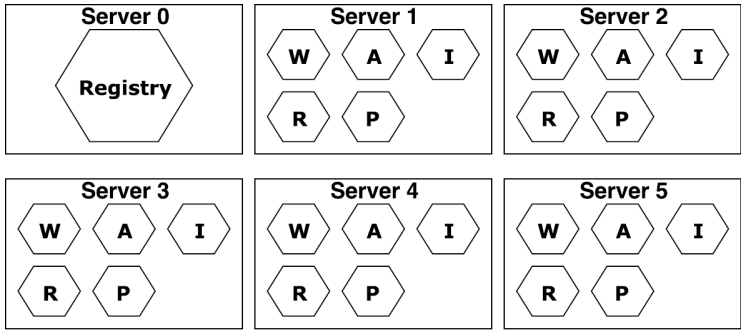
$$P_{10}(x) = \begin{cases} 0 & \text{if } x = 5 \\ 0.99 & \text{if } x = 10 \\ 0.01 & \text{if } x = 20 \\ 0 & \text{else.} \end{cases}$$

## Deployment

### Calibration

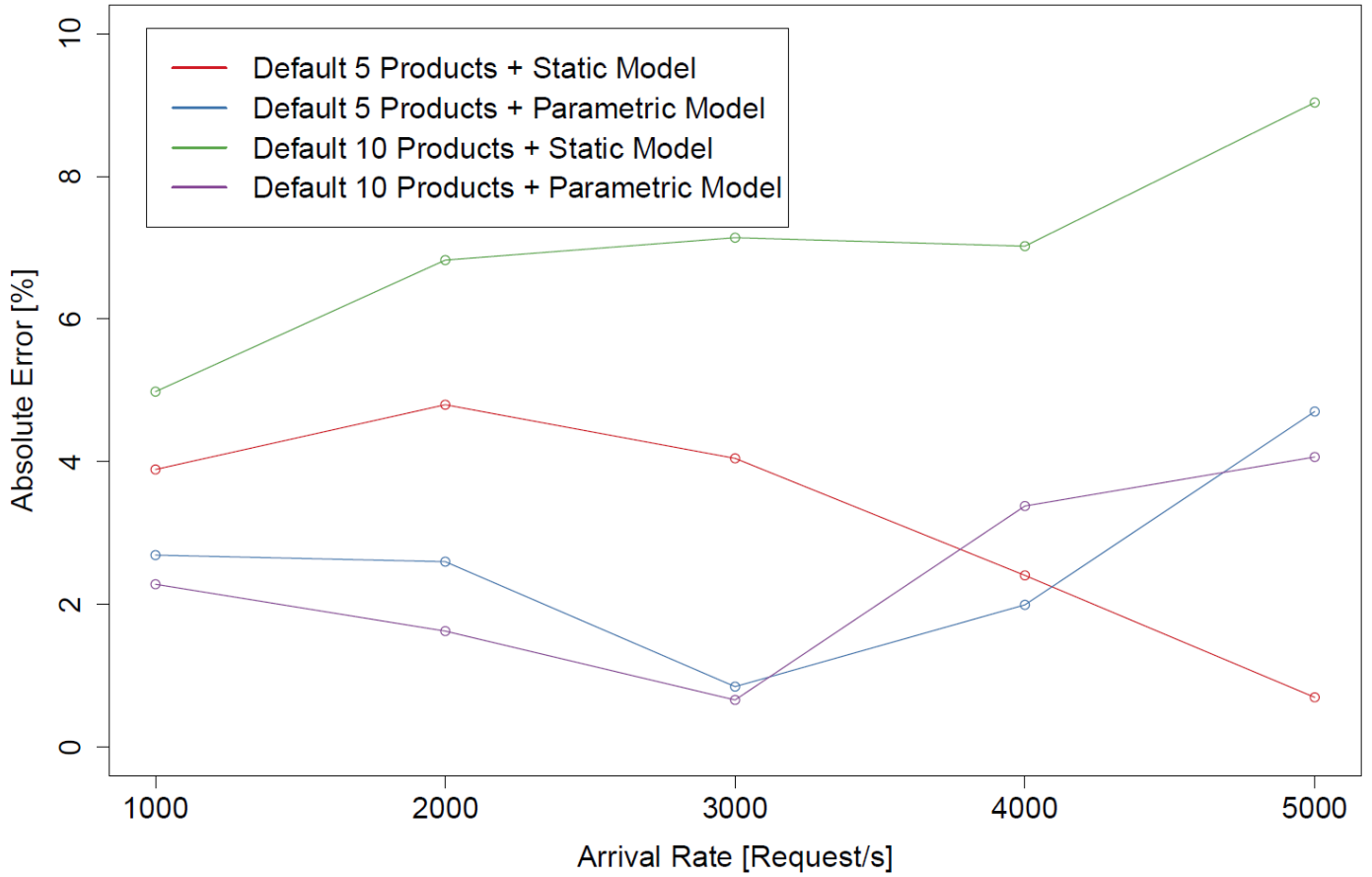



### To Predict



# Performance Model - Results

Results with and without considering the parametric dependency using Service Demand Law-based model



# Energy Efficiency - Scenario

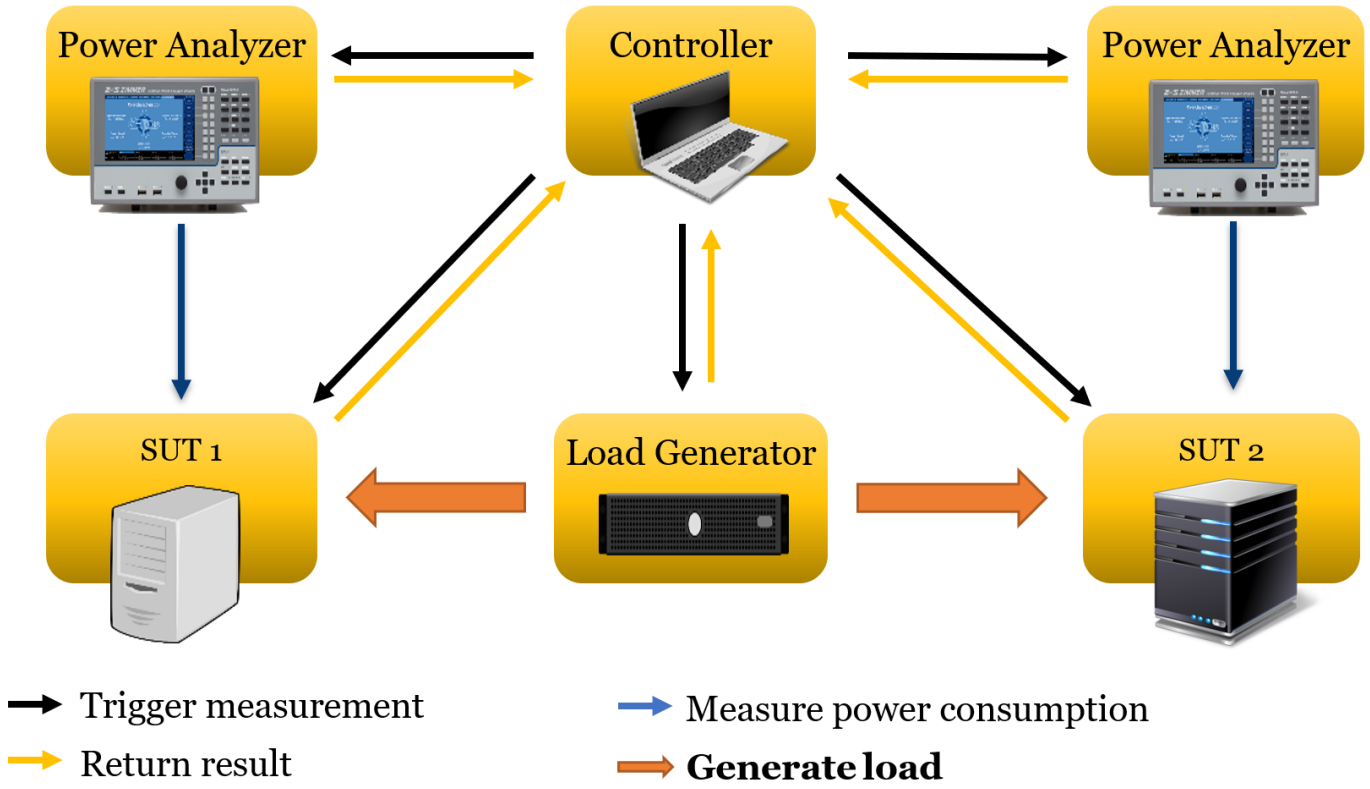
Energy efficiency of placements

- Goal: Show that power consumption, energy efficiency, and performance scale differently
  - Different optima for service placements
- Approach:
  - Distribute TeaStore on heterogeneous servers
  - Put TeaStore under stress-test load intensity
  - Measure TeaStore performance and server wall power

# Energy Efficiency - Measurement

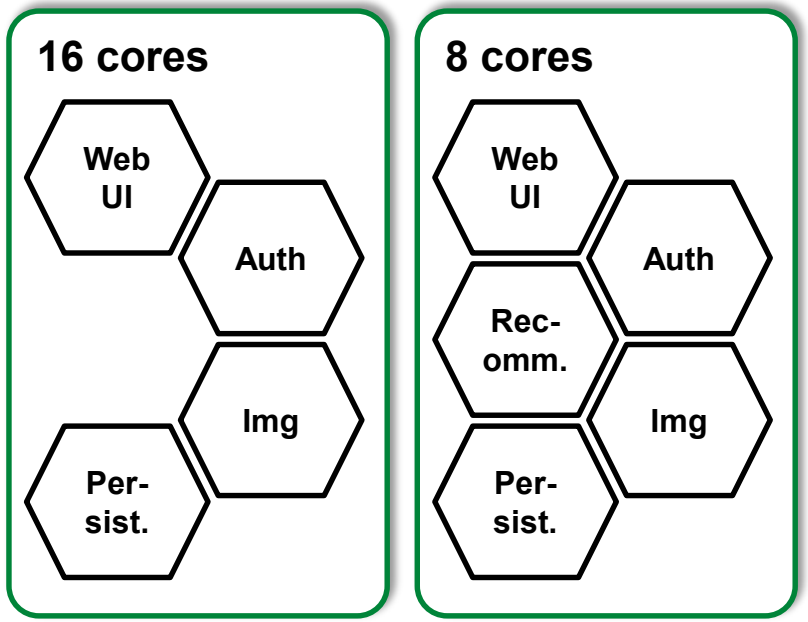
Measurements in heterogeneous setting

- SUT 1:
  - 16 core Haswell
  - 32 GB RAM
- SUT 2 (Heterogeneous):
  - 8 core Skylake
  - 16 GB RAM
- Metrics:
  - Throughput
  - Power
  - Energy Efficiency
    - Throughput / Power



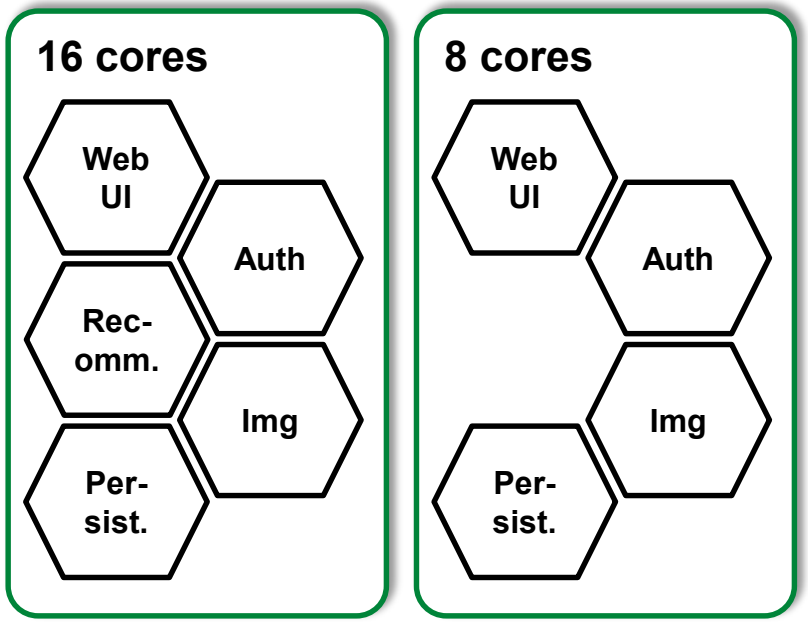
# Energy Efficiency – Optima for Heterogeneous Placement

## Placement Candidate 1




| | |
|---|---|
| Max | 1011.9 Tr/s |
| Max | 179.6 W |
| Geo | 4.4 Tr/J |

## Placement Candidate 2




| | |
|---|---|
| Max | 1067.7 Tr/s |
| Max | 187.0 W |
| Geo | 4.3 Tr/J |

# 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
  - 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


[1] RUBiS User's Manual, May 2008.

[2] Standard Performance Evaluation Corporation (SPEC). SPEC jEnterprise 2010 Design Document. https://www.spec.org/jEnterprise2010/docs/DesignDocumentation.html, May 2010, Accessed: 16.10.2017.

[3] Weaveworks Inc. Sock Shop: A Microservice Demo Application. https://github.com/microservices-demo/microservices-demo, 2017, Accessed: 19.10.2017.

[4] A. van Hoorn, J. Waller, and W. Hasselbring. Kieker: A framework for application performance monitoring and dynamic software analysis. In *Proceedings of the 3rd joint ACM/SPEC International Conference on Performance Engineering (ICPE 2012)*, April 2012.

[5] J. von Kistowski, M. Deffner, and S. Kounev. Run-time Prediction of Power Consumption for Component Deployments. In *Proceedings of the 15th IEEE International Conference on Autonomic Computing (ICAC 2018)*, Trento, Italy, September 2018.

[6] J. von Kistowski, N. Herbst, S. Kounev, H. Groenda, C. Stier, and S. Lehrig. Modeling and Extracting Load Intensity Profiles. *ACM Transactions on Autonomous and Adaptive Systems (TAAS)*, 11(4):23:1 - 23:28, January 2017.

[7] T. C. Chieu, A. Mohindra, A. A. Karve, and A. Segal. Dynamic scaling of web applications in a virtualized cloud computing environment. In *International Conference on E-Business Engineering*, 2009.

[8] N. R. Herbst, S. Kounev, and R. Reussner. Elasticity in Cloud Computing: What it is, and What it is Not. In *Proceedings of the 10th International Conference on Autonomic Computing (ICAC 2013)*, San Jose, CA, June 2013.