Julius-Maximilians-
UNIVERSITÄT
WÜRZBURG

Descartes
research

TeaStore

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

Presentation

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

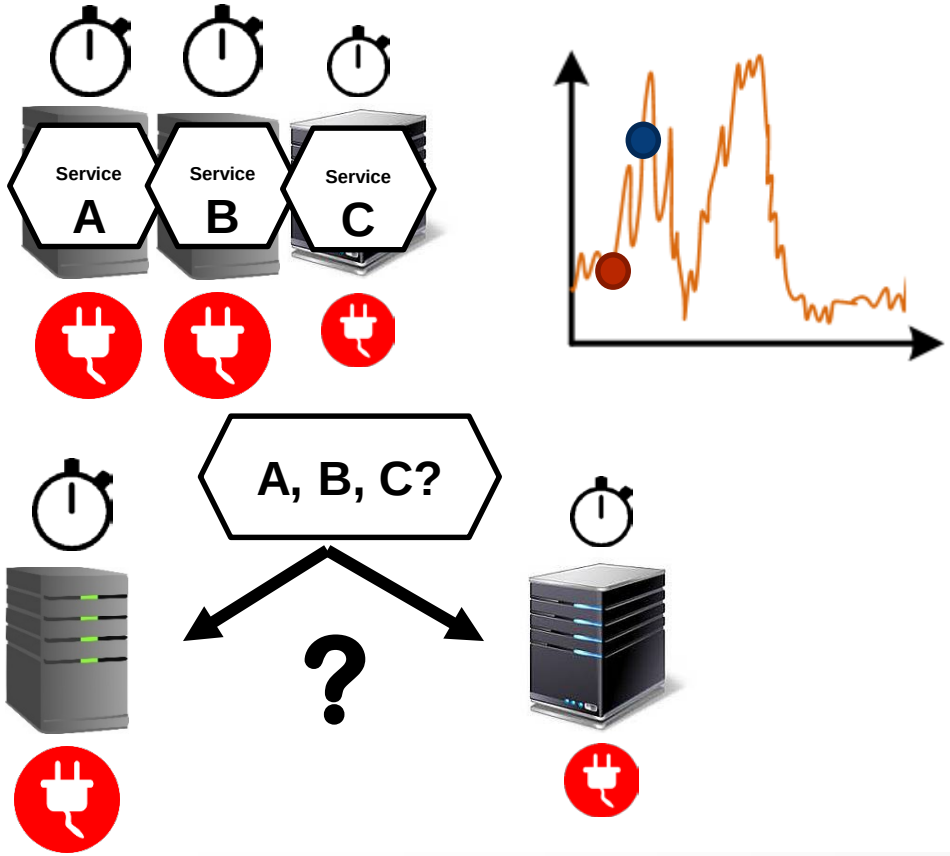
April 9, 2018

https://github.com/DescartesResearch/TeaStore

Example Research Scenario

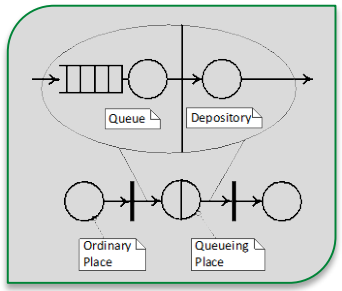
Auto-Scaling and Placement

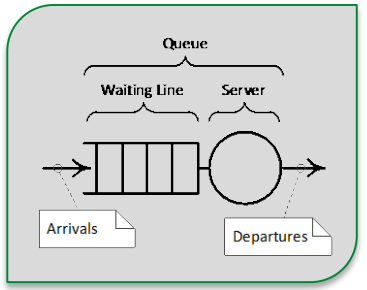
- Placement at run-time



Performance Modeling

- An approach for the auto-scaling + placement problem
- Build or extract model





- Use Model for placement decision

Challenge: Evaluation of …

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

➢ Require **reference and test applications**

Reference applications help to

- Measure placement power consumption and performance
- Measure auto-scaler elasticity
- Evaluate model (extractor) accuracy

Requirements for a Test Application

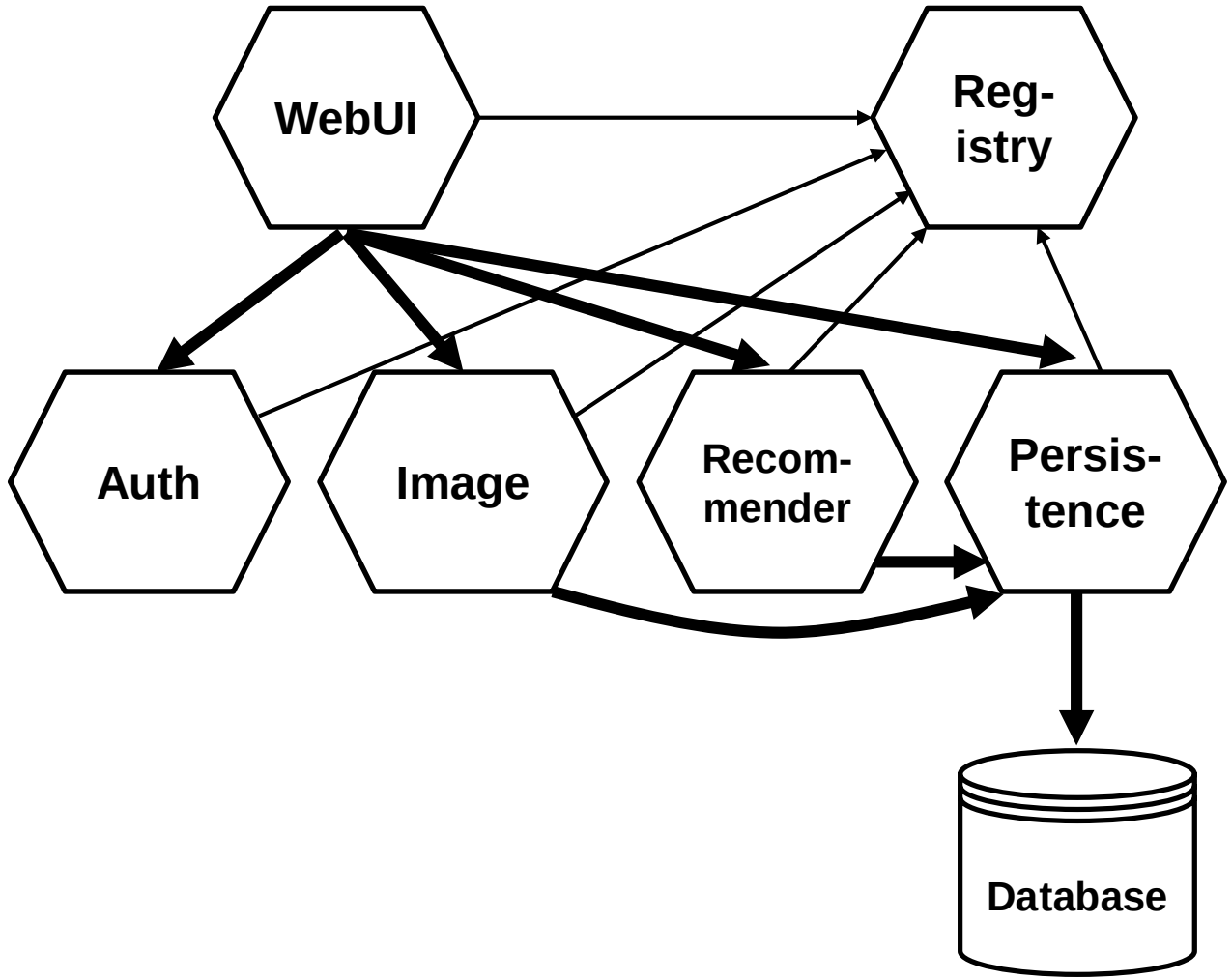
- Scalable
- Allows for changes at run-time
- Reproducible performance results
- Diverse performance behavior
- Online monitoring
- Load Profiles

- Simple setup
- Modern technology stack

The TeaStore

Micro-Service test application

- Five Services + Registry
- Uses Netflix “Ribbon” client-side load balancer
  - Swarm/Kubernetes supported, not required
- Pre-instrumented variant with Kieker
- Has Docker Images
  - Alternatively: documentation for manual deployment



Services I

Registry

- Simplified Eureka
- Service location repository
- Heartbeat

RegistryClient

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

WebUI

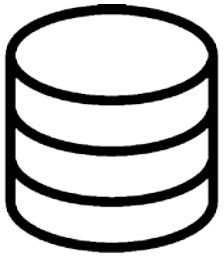
- Servlets/Bootstrap
- Integrates other services into UI
- ➢ **CPU + Memory + Network I/O**

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 + Storage**

Recommender

- Recommends prod-ucts based on history
- 4 different algorithms

➢ **Memory or CPU**

TraceRepository

- AMQP Server
- Collects traces from all services

Additional Performance Properties

Two types of caches

- Black-box persistence cache
- White-box image provider cache

Different load types

- CPU
- I/O
- Network

Internal state

- Database size influences resource demands

Load independent tasks

- Periodic recommender retraining (optional)

Startup behavior

- Auth and WebUI start “instantly”
- Recommender needs training on startup
- Image Provider creates images on startup

Configuration options

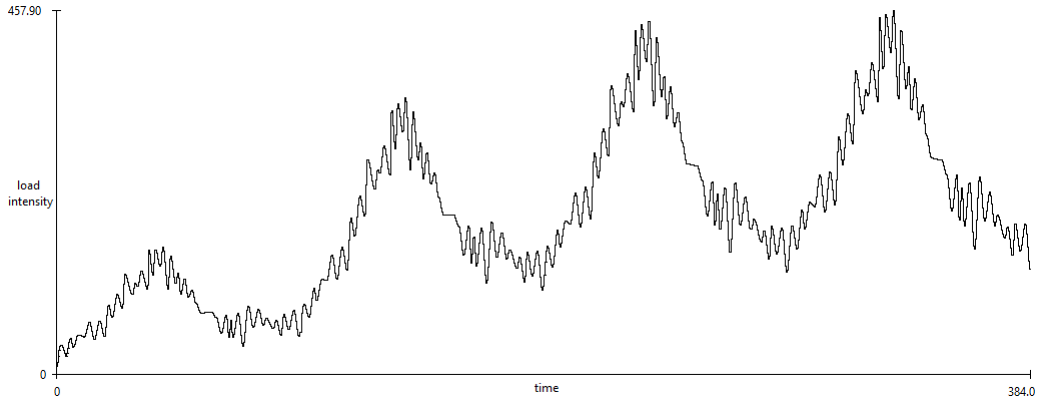
- Recommender algorithms
- Recommender retraining interval
- Image Provider cache implementations
- Database size

Load and Usage Profile

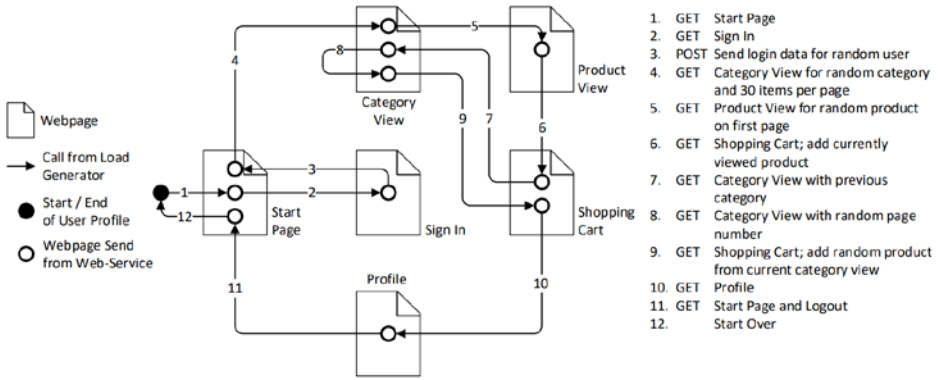
HTTP load generator

- Supports varying load intensity profiles
  - Can be created manually
  - Or using LIMBO
- Scriptable user behavior
  - Uses LUA scripting language
  - “Browse” Profile on Github

Example load intensity profile:

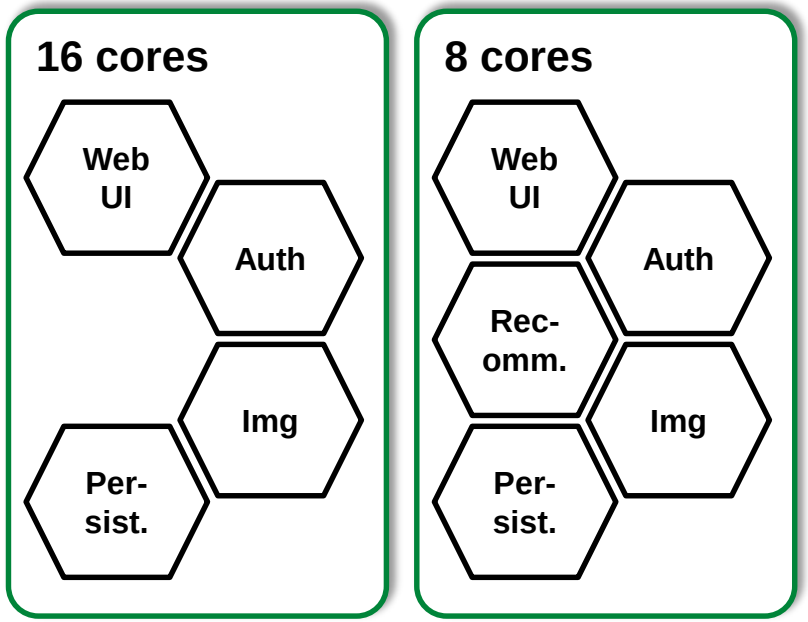


“Browse” user profile:



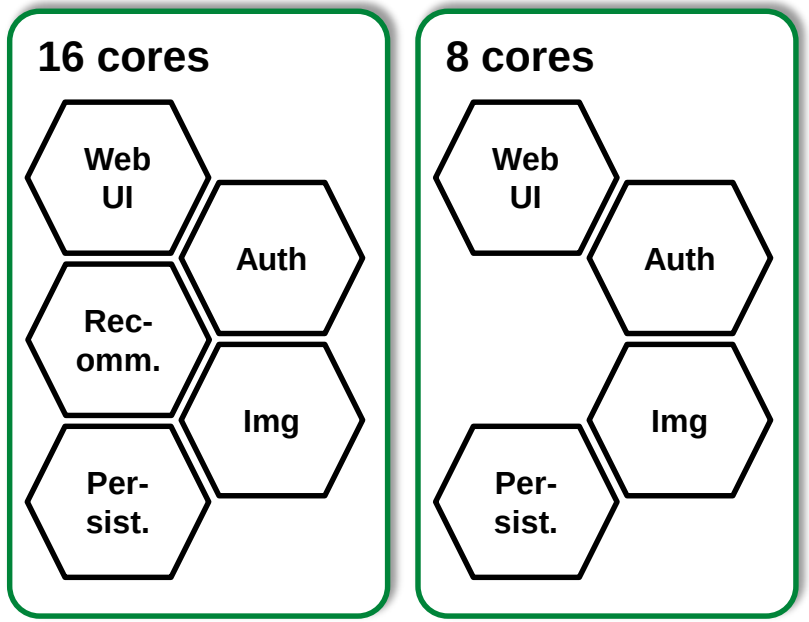
Example: Energy Efficiency of Placements

Placement Candidate 1



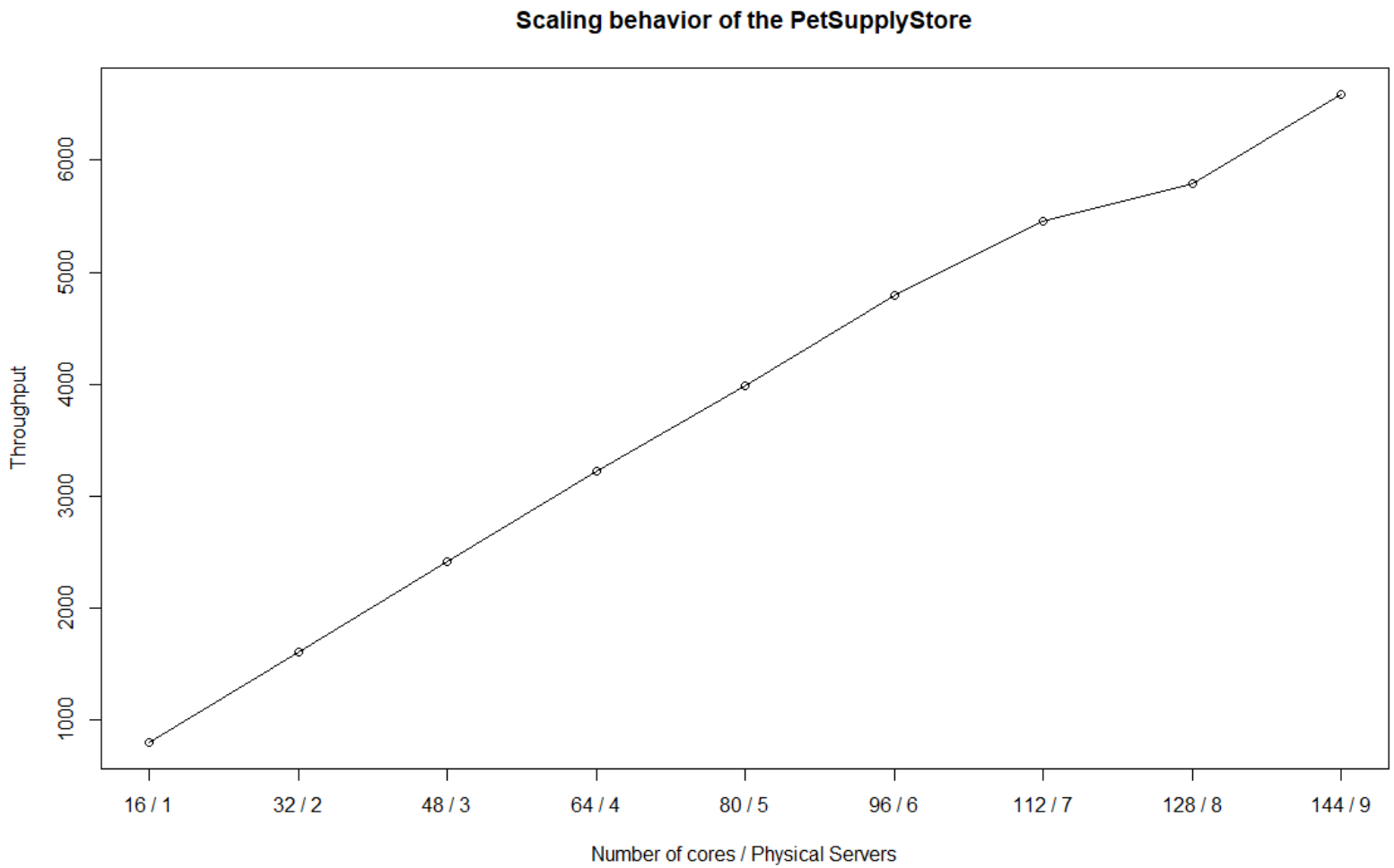
Max	1011.9 Tr/s
Max	**179.6** W
Geo	**4.4** Tr/J

Placement Candidate 2



Max	**1067.7** Tr/s
Max	187.0 W
Geo	4.3 Tr/J

Does it scale?



Thank You!

https://github.com/DescartesResearch/TeaStore