Chameleon: A Hybrid, Proactive Auto-Scaling Mechanism on a Level-Playing Field

André Bauer

Workshop on Hot Topics in Cloud Computing Performance
Umeå, June 16, 2019
Auto-Scaling (AS) of Cloud Infrastructures

- Cloud infrastructure providers have to face changing requirements

- To guarantee a reliable service, most application run with a fixed amount of resources
  - High energy consumption, if the system is not fully utilized
  - Bad performance, if unexpected peaks appear

- High quality auto-scalers are required, which reconfigure the system regarding its load
Related Work on Auto-Scaling Methods

Auto-scalers can be classified into 5 groups [Lorido-Botran14]

Prominent examples are:

- Threshold-based Rules
  - [Maurer11]

- Queueing Theory
  - [Urgaonkar08] (Hist)

- Reinforcement Learning
  - [Rao09]

- Control Theory
  - [Adhikari12] (Adapt)

- Time Series Analysis
  - [Iqbal11] (Reg)

→ Predictive models from different disciplines are applied mostly in isolation.
→ Smart integration of multiple predictive/proactive with reactive mechanisms is missing.
Challenges of Auto-Scaling

Based on related work, we identify following challenges:

- Knowledge: models, history
- Awareness of own and system’s performance and its boundaries
  - Descriptive performance model
- Guide to detect need/demand
  - Resource demand estimation
- Proactive planning of actions
  - Time series forecasting
- Reliable fallback options
  - Reactive cycle as fallback

A resource demand is the time a unit of work (e.g., request) spends obtaining service from a resource (e.g., CPU or hard disk) in a system (excluding waiting time). [Spinner15]
Chameleon Auto-Scaler: Decision Logic

- Simplification: Each service modelled as M/M/1/∞ queue
- Input: observed (reactive) and forecast (proactive) arrival rate
- Resource demand estimations based on monitored utilization, throughput and response time, e.g., service demand law
- Target utilization & response time → # resources add/remove
- Check “trustworthiness” of proactive scaling decisions
- Resolve conflicts in between proactive and reactive
- Optimize proactive scaling decisions pairwise
Chameleon: Example

$\text{f}_1$

$\text{p}_{3,0}$
Assumptions and Limitations

- Forecasting
  - 2 days of historical data is required

- Monitoring
  - Requests per second, response time and utilization are gathered by a monitoring infrastructure

- SLO
  - Response time of the application

- Use case
  - CPU intensive, request-based applications due to resource demand estimation

- Descriptive model
  - Can be transformed into a queuing network
Evaluation Setup

- Scaling a Java web application
  - Re-implementation of LU worklet from Rating Tool SERT™2
  - LU decomposition of nxn matrix, where n is GET parameter

- 3 different Environments
  - Private CloudStack
  - AWS EC2 IaaS cloud
  - Distributed ASCI Supercomputer 4 (DAS-4)

- 5 real-world traces
  - FIFA, BibSonomy, IBM, Wikipedia, and Retailrocket
  - 3 days each 3.2 hours → 9.6 hours experiment

- More than 400 hours of experiments
Benchmarking

- Evaluation with Bungee experiment controller [Herbst15]
  - Perform each scenario with Chameleon
  - Perform each scenario with standard reactive auto-scaler
  - Perform each scenario with sota auto-scalers
    - Hist [Urgaonkar08]
    - Reg [Iqbal11]
    - Adapt [Adhikari12]
    - ConPaaS [Pierre12]
  - Compare the results with benchmarking metrics
    - Individual elasticity metrics
    - Aggregate elasticity metrics
    - User metrics
Elasticity Metrics

- Accuracy
- AS deviation
- Timeshare
- Elasticity speed-up
- Instability
- Pairwise competition

\[ \kappa_a(\%) := \frac{1}{(n-1) \cdot |x|} \cdot \sum_{i=1; i \neq j=1}^{n} |x| \cdot \omega(i, j) \]

where

\[ \omega(i, j) := \begin{cases} 
0, & x_a(j) > x_i(j) \\
0.5, & x_a(j) = x_i(j) \\
1, & x_a(j) < x_i(j) 
\end{cases} \]
Experiment Example: Chameleon on CS, Retailrocket

- **Introduction**
- **Evaluation**
- **Summary**
- **Approach**
Experimental Evaluation: CS, Retailrocket

- **Setup**
- **Introduction**
- **Evaluation**
- **Summary**

### Approach

**Chameleon (Retailrocket)**

**Adapt (Retailrocket)**

**Hist (Retailrocket)**

**ConPaaS (Retailrocket)**

**Reactive (Retailrocket)**

**Reg (Retailrocket)**

- **Demand VMs**
- **Supplied VMs**

Time in minutes: 0 30 60 90 120 150 180 210 240 270 300 330 360

Amount of VMs: 0 6 12 18
Experimental Evaluation: private CS vs. AWS EC2

Introduction

Evaluation

Summary

Approach
## Summary of all Experiments: Average Metrics

<table>
<thead>
<tr>
<th>Metric</th>
<th>Chameleon</th>
<th>Adapt</th>
<th>Hist</th>
<th>ConPaaS</th>
<th>Reg</th>
<th>Reactive</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\bar{\theta}_U$ (avg. accuracy$_U$)</td>
<td>3.63%</td>
<td>6.45%</td>
<td>4.70%</td>
<td>15.55%</td>
<td>15.69%</td>
<td>6.98%</td>
</tr>
<tr>
<td>$\bar{\theta}_O$ (avg. accuracy$_O$)</td>
<td>17.88%</td>
<td>19.94%</td>
<td>52.64%</td>
<td>25.98%</td>
<td>10.51%</td>
<td>34.47%</td>
</tr>
<tr>
<td>$\bar{\tau}_U$ (avg. time share$_U$)</td>
<td>13.32%</td>
<td>30.43%</td>
<td>22.75%</td>
<td>42.04%</td>
<td>43.71%</td>
<td>25.41%</td>
</tr>
<tr>
<td>$\bar{\tau}_O$ (avg. time share$_O$)</td>
<td>65.06%</td>
<td>51.41%</td>
<td>62.35%</td>
<td>41.69%</td>
<td>33.42%</td>
<td>62.08%</td>
</tr>
<tr>
<td>$\bar{v}$ (avg. instability)</td>
<td>13.91%</td>
<td>16.60%</td>
<td>11.95%</td>
<td>17.42%</td>
<td>17.02%</td>
<td>12.99%</td>
</tr>
<tr>
<td>$\bar{\psi}$ (avg. SLO violations)</td>
<td>10.29%</td>
<td>32.76%</td>
<td>15.59%</td>
<td>44.11%</td>
<td>60.16%</td>
<td>21.96%</td>
</tr>
<tr>
<td>$\bar{\sigma}$ (avg. as deviation)</td>
<td>39.63%</td>
<td>46.90%</td>
<td>46.43%</td>
<td>54.03%</td>
<td>63.46%</td>
<td>48.14%</td>
</tr>
<tr>
<td>$\bar{\kappa}$ (avg. pairwise comp.)</td>
<td>69.44%</td>
<td>50.00%</td>
<td>58.33%</td>
<td>36.51%</td>
<td>42.46%</td>
<td>55.56%</td>
</tr>
<tr>
<td>$\bar{\epsilon}$ (avg. elastic speedup)</td>
<td>2.02</td>
<td>1.48</td>
<td>1.38</td>
<td>1.10</td>
<td>1.41</td>
<td>1.49</td>
</tr>
</tbody>
</table>
Auto-Scaler Benchmark Competition: Findings

- **Chameleon** outperforms in the evaluated scenarios
  - Reliable slight over-provisioning, lowest SLO violations
  - Coupling of proactive and reactive scaling decisions improves the elasticity
- **Adapt:** closely follows the demand, high number of adaptations
- **Hist and Reactive:** high over-provisioning accuracy
- **Reactive:** accurate, timely CPU utilization metrics required – not always reliable
- **ConPaaS and Reg:** unstable behavior – often not reliable
In a Nutshell

- Cloud Infrastructure providers have to face changing requirements

- High quality auto-scaler are required
  - Predictive models from different disciplines are applied mostly in isolation
  - Smart integration of multiple predictive/proactive with reactive mechanisms is missing

- Design of a hybrid auto-scaler Chameleon

- More than 400 hour evaluation in 3 different environments with 5 real-world traces

- Chameleon outperforms other auto-scalers
References


Thank you for your attention!
Experimental Evaluation: CS, Wiki

Chameleon (German Wikipedia)

Reactive (German Wikipedia)

Adapt (German Wikipedia)

Hist (German Wikipedia)

ConPaaS (German Wikipedia)

Reg (German Wikipedia)

Amount of VMs

Time in minutes

D - Demanded VMs  S - Supplied VMs
Experimental Evaluation: CS, IBM

Chameleon (IBM)

Reactive (IBM)

Adapt (IBM)

Hist (IBM)

ConPaaS (IBM)

Reg (IBM)

![Graphs showing the comparison of different methods over time.](image-url)
Chameleon Components

- Chameleon (Retailrocket)
- Chameleon Proactive (Retailrocket)
- Chameleon Reactive (Retailrocket)

Amount of VMs vs. Time in minutes (0-360 minutes)

Legend:
- Demanded VMs
- Supplied VMs
Evaluation – Scaling Behavior

![Graph showing scaling behavior for different cloud services.](image)