Chameleon: Design and Evaluation of a Proactive, Application-Aware Elasticity Mechanism

André Bauer, Simon Spinner, Nikolas Herbst

Chair of Software Engineering
University of Würzburg
http://se.informatik.uni-wuerzburg.de/

SPEC RG Cloud Work Group
Würzburg, 25/10/2016
1. Motivation
2. Related Work
3. Approach
4. Evaluation
5. Conclusion
Motivation

- Cloud computing is a paradigm which faces the increasing scale and complexity of modern internet services

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

Prominent examples of each group:

- Threshold-based Rules [Maurer11]
- Queueing Theory [Urgaonkar08] (Hist)
- Control Theory [Adhikari12] (Adapt)
- Reinforcement Learning [Rao09]
- Time Series Analysis [Iqbal11] (Reg)

Mechanism are either generic or application-specific

Delimitation: Chameleon is both generic and application-aware
Existing auto-scalers are either application-specific or generic. Bad performance can be translated into loss of money or customers.

Controller combines time series analysis and query theory enriched with application knowledge.

Expected: More accurate scaling decisions compared to other controllers or standard reactive.

Taking forecasting, resource demand estimation and application model into account to increase the performance of the reconfiguration.
Data Flow of Chameleon

1: requests historical data
2: sends data
3: sends data
4: forecasts load
5: determines resource demands
6: reconfigures cloud

Cloud
Business Application

Chameleon Controller

Resource Demand Estimation Component - LibReDE -

Forecast Component - WCF -

Performance Repository

DML Model

Time Series Storage
has DML model
monitors cloud
saves the observed data
Assumptions

- Availability of at least 4 days of historical data
- The SLO is the response time
- Chameleon is limited to scale CPUs
- A DML model is already provided

Chameleon is driven with two different scenarios
1. CPU intensive scenario
2. The business application SPECjEnterprise2010

Both scenarios are deployed in a private CloudStack environment
FIFA world cup 1998 is used for an authentic workload
- A sequence of 7 days were cropped out
- 4 days as historical data for WCF
- 3 days for the evaluation
Evaluation with Bungee experiment controller
- Perform each scenario with Chameleon
- Perform each scenario with other auto-scalers
  1. Hist [Urgaonkar08]
  2. Reg [Iqbal11]
  3. Adapt [Adhikari12]
  4. ConPaaS [Pierre12]
- Compare the results with benchmarking metrics

Benchmarking Metrics
- Elasticity metrics
- User metrics
  - Number of SLO violations
  - Average response time
  - 90th percentile of response time
CPU Intensive Scenario

- Workload is accelerated 8 times: 3 d → 9 h
  - Between 40 and 200 request/second
  - 3.5 million requests

- Setup contains up to 20 WildFly slaves, 1 Domain controller and 1 Citrix Netscaler (LB)
  - 2 cores @2.6GHz
  - 4 GB memory

- Each slave is a http application
  - Request parameter n, here 600
  - LU decomposition of a random generated nxn matrix

- SLO: 90% of responses < 5 seconds
Scaling Chameleon

- Amount of VMs
- Time in minutes
- Demanded VMs
- Supplied VMs

Response Time in milliseconds
- Waiting Time
- Service Time
- Time in minutes

Evaluation Part I

A. Bauer

Introduction
Approach
Conclusion
Evaluation Part II

No Auto Scaling

Scaling Chameleon

Scaling Hist

Scaling Reg
Evaluation Part III

Scaling Chameleon

Scaling Adapt

Scaling ConPaaS
<table>
<thead>
<tr>
<th>Metrics</th>
<th>No Scaler</th>
<th>Chameleon</th>
<th>Hist</th>
<th>Adapt</th>
<th>Reg</th>
<th>ConPaaS</th>
</tr>
</thead>
<tbody>
<tr>
<td>accuracy₀</td>
<td>0.282</td>
<td>0.350</td>
<td>0.387</td>
<td>0.079</td>
<td>0.289</td>
<td>0.387</td>
</tr>
<tr>
<td>accuracyᵤ</td>
<td>0.147</td>
<td>0.030</td>
<td>0.055</td>
<td>0.613</td>
<td>0.078</td>
<td>0.105</td>
</tr>
<tr>
<td>timeshare₀</td>
<td>0.453</td>
<td>0.755</td>
<td>0.608</td>
<td>0.181</td>
<td>0.52</td>
<td>0.535</td>
</tr>
<tr>
<td>timeshareᵤ</td>
<td>0.491</td>
<td>0.129</td>
<td>0.285</td>
<td>0.788</td>
<td>0.371</td>
<td>0.355</td>
</tr>
<tr>
<td>jitter</td>
<td>-0.924</td>
<td>-0.723</td>
<td>-0.654</td>
<td>-0.615</td>
<td>-0.719</td>
<td>-0.466</td>
</tr>
<tr>
<td>instability</td>
<td>0.478</td>
<td>0.215</td>
<td>0.22</td>
<td>0.418</td>
<td>0.196</td>
<td>0.235</td>
</tr>
<tr>
<td>#Adaptions</td>
<td>0</td>
<td>85</td>
<td>110</td>
<td>618</td>
<td>93</td>
<td>204</td>
</tr>
<tr>
<td>Avg. #VMs</td>
<td>10</td>
<td>12.5</td>
<td>13.4</td>
<td>8.1</td>
<td>11.1</td>
<td>13.2</td>
</tr>
<tr>
<td>score&lt;sub&gt;NoScaler&lt;/sub&gt;</td>
<td>1</td>
<td>1.330</td>
<td>1.151</td>
<td>0.896</td>
<td>1.177</td>
<td>1.143</td>
</tr>
<tr>
<td>Avg. response time</td>
<td>1251 ms</td>
<td>671 ms</td>
<td>1267 ms</td>
<td>4339 ms</td>
<td>2018 ms</td>
<td>2082 ms</td>
</tr>
<tr>
<td>P₉₀ response time</td>
<td>5002 ms</td>
<td>1051 ms</td>
<td>5002 ms</td>
<td>5002 ms</td>
<td>5002 ms</td>
<td>5002 ms</td>
</tr>
<tr>
<td>Violations</td>
<td>18.07%</td>
<td>7.50%</td>
<td>19.24%</td>
<td>82.75%</td>
<td>32.25%</td>
<td>35.98%</td>
</tr>
</tbody>
</table>

Chameleon achieves highest score and lowest SLO violations.
Future Work

- For Chameleon:
  - Add vertical scaling and migration
  - Improve precision by dynamic method selection
  - Integration of a DML extractor and solver
  - Taking billing of reconfigurations into account

- For evaluation:
  - Distributed JMeter usage
  - Calibration search vs. bottleneck shadowing
  - Retaking the experiments in a public cloud
  - Multitier scaling of SPECjEnterprise2010
  - Scale another multi-tier application (e.g., RuBiS)
  - Evaluate energy savings by auto-scaling
Conclusion

Problem

Existing auto-scalers are either application-specific or generic. Bad performance can be translated into loss of money or customers.

Idea

Chameleon combines time series analysis and query theory enriched with application knowledge.

Results

Chameleon achieves the best score in the CPU intensive scenario compared with 4 existing auto-scaler.

Ongoing Work

Release Chameleon as open source, …
Thank you for your attention

andre.bauer@stud-mail.uni-wuerzburg.de
http://se.informatik.uni-wuerzburg.de


Let $s_t, d_t$ be the supply, demand at time $t$, with $t \in [0, T]$.

**Elasticity Metrics**

$$\text{accuracy}_{U_k} = \frac{1}{T} \sum_{t=1}^{T} \frac{\max(d_t - s_t, 0)}{d_t} \Delta t$$

$$\text{accuracy}_{O_k} = \frac{1}{T} \sum_{t=1}^{T} \frac{\max(s_t - d_t, 0)}{d_t} \Delta t$$

$$\text{timeshare}_{U_k} = \frac{1}{T} \sum_{t=1}^{T} \max(\text{sign}(s_t - d_t), 0) \Delta t$$

$$\text{timeshare}_{O_k} = \frac{1}{T} \sum_{t=1}^{T} \max(\text{sign}(s_t - d_t), 0) \Delta t$$

$$\text{instability}_k = \frac{1}{T - 1} \sum_{t=2}^{T} \min(\max(\text{sign}(s_t) - \text{sign}(d_t), 0), 1) \Delta t$$

$$\text{jitter}_k = \frac{1}{T} \left( \sum_{t=2}^{T} |s_t - s_{t-1}| - \sum_{t=2}^{T} |d_t - d_{t-1}| \right)$$

$$\text{score}(\nu)_k = \left( \frac{\text{accuracy}_{b}}{\text{accuracy}_{y_k}} \right)^{w_{ac}} \times \left( \frac{\text{timeshare}_{b}}{\text{timeshare}_{k}} \right)^{w_{ts}} \times \left( \frac{\text{jitter}_{k}}{\text{jitter}_{y_k}} \right)^{w_{jn}} \times \left( \frac{\text{instability}_{k}}{\text{instability}_{y_k}} \right)^{w_{in}}$$
Reproducibility

- Reproducibility of the scaling behaviour
  - 3 runs with the same setting
  - Elasticity score has a standard variation of 0.017
- Reactive scaling events vs. proactive scaling events

- Chameleon plans reconfiguration steps in advance
Logic II

- Approach
  - Evaluation
    - $t_1 = \min(f_1, f_2)$
    - $t_2 = \max(f_2 - f_1, 0)$

- Conclusion
  - $t_1 = \text{floor}(d \cdot f_1)$
  - $t_2 = \text{ceil}(d \cdot (f_2 + f_1))$
  - $0 < d \leq 1$
  - $t_1 = \text{ceil}(d \cdot f_1)$
  - $t_2 = \text{floor}(d \cdot (f_2 + f_1))$
Resource demand for Chameleon is determined via LibReDE

Resource demand for the existing auto-scalers is determined via the server speed
Chapter 1: Introduction

Approach

Conclusion
Modified WCF forecast strategy:
- Forecasts on demand
- Forecast model contains the next 24 predictions
- If MAPE > tolerance then build new forecast model
Setup contains 8 different tiers (= WildFly slaves), 1 Domain controller and 1 Citrix Netscaler (LB)

Problems with benchmarking
- Very small response time (about 10 ms)
- High load generation burden
- Difficult calibration (reproducibility, hidden bottlenecks, …)

SLO: 90% responses < 100 milliseconds

Only user metrics can be determined
- Number of SLO violations
- Average & median response time
- 90th percentile of response time
## Metric Comparison

### Metrics

<table>
<thead>
<tr>
<th>Metrics</th>
<th>Reactive</th>
<th>Chameleon</th>
<th>Hist</th>
<th>Adapt</th>
<th>Reg</th>
</tr>
</thead>
<tbody>
<tr>
<td>Avg. response time</td>
<td>669 ms</td>
<td>476 ms</td>
<td>404 ms</td>
<td>592 ms</td>
<td>500 ms</td>
</tr>
<tr>
<td>P$_{50}$ response time</td>
<td>9 ms</td>
<td>9 ms</td>
<td>7 ms</td>
<td>12 ms</td>
<td>10 ms</td>
</tr>
<tr>
<td>P$_{90}$ response time</td>
<td>2005 ms</td>
<td>1876 ms</td>
<td>1754 ms</td>
<td>1899 ms</td>
<td>1915 ms</td>
</tr>
<tr>
<td>Violations</td>
<td>33,02 %</td>
<td>27,19 %</td>
<td>26,57%</td>
<td>27,32 %</td>
<td>30,16 %</td>
</tr>
</tbody>
</table>

**Chameleon achieves the second best user metrics**