Design and Evaluation of a Proactive, Application-Aware Auto-Scaler

Tutorial Paper

André Bauer
University of Würzburg, Germany
andre.bauer@uni-wuerzburg.de

Nikolas Herbst
University of Würzburg, Germany
nikolas.herbst@uni-wuerzburg.de

Samuel Kounev
University of Würzburg, Germany
samuel.kounev@uni-wuerzburg.de

ABSTRACT

Simple, threshold-based auto-scaling mechanisms as mainly used in practice bring no features to overcome resource provisioning delays and non-linear scalability of a software service. In this tutorial paper, we guide the reader step-by-step through the design and evaluation of a proactive and application-aware auto-scaling mechanism.

First, we introduce the building blocks for such an auto-scaling mechanism: (i) an on-demand arrival rate forecasting method, (ii) resource demand estimates at run-time, (iii) a descriptive and continuously updated performance model of the deployed software and (iv) an intelligent adaptation planner that incorporates a threshold-based mechanism as fall-back.

Second, we cover auto-scaler evaluation steps: (i) the preparation steps are an application scenario and workload profile definition and (ii) an automated scalability analysis. In step (iii), we show how representative and repeatable auto-scaler experiments can be conducted and (iv) the results analyzed with the help of elasticity and end-user metrics for a detailed and fair comparison of alternative auto-scaler mechanisms and their respective configurations even across platforms.

For the individual steps of the construction of the auto-scaler building blocks and for their evaluation, we shortly introduce open-source tools available online[1].

CCS Concepts

•General and reference → Cross-computing tools and techniques; •Networks → Cloud computing; •Computer systems organization → Self-organizing autonomic computing; •Software and its engineering → Virtual machines;

Keywords

Elasticity, IaaS, Cloud, Auto-Scaler, Benchmark, Metric, Load Profile, Resource Demand, Forecasting, Descriptive Performance Model

1. INTRODUCTION

In practice, threshold-based controllers are commonly applied to dynamically add or remove computing resources of a service in an elastic cloud environment. Numerous proactive approaches based on control theory, queuing theory, machine learning, and time series analysis have been proposed in literature. In general, a proactive auto-scaler aims to overcome resource provisioning delays. We understand application-aware as features to learn non-linear scalability and inter-dependencies between different layers of a service.

In the first part, this tutorial paper presents an ensemble of open-source tools available online[1] that enable the composition of a proactive, and application-aware auto-scaler with the following components:

Forecasting of Arrival Rates:
Forecasting the next request arrival rates of incoming requests is an essential requirement for proactive auto-scaling. Hence, we use a self-adaptive forecasting technique with combinations of time series analysis methods and forecast accuracy feedback. We select as solution for this issue the WCF (Workload Classification and Forecasting Framework, see Section 2) [2].

Online Resource Demand Estimation:
Resource demand estimates per request class are crucial information for good auto-scaling decisions. In this part of the tutorial, we cover a library of methods that estimate the resource demand per request class based on basic performance metrics. For us, the library LibReDE (Library for Resource Demand Estimation, see Section 3) [5] offers the demanded features.

Descriptive Software Performance Model:
In the case, multiple layers of a service are elastically scalable, the next required information is at which layer and how many resources should be added or removed. Thus, we discuss about a descriptive and continuously updated performance model of the deployed application. We base on the Descartes Modeling Language (DML, see Section 4) [4]).

Proactive, Application-Aware Auto-Scaler:
After the required components are introduced, we cover

[1]Descartes Tools: http://descartes.tools/

ICPE'17, April 22 - 26, 2017, L'Aquila, Italy

ISBN 978-1-4503-4404-3/17/04…$15.00

DOI: http://dx.doi.org/10.1145/3030207.3053678

a possible integration of these tools and the associated logic behind the scaling decisions. We call our proactive, and application-aware elasticity mechanism Chameleon (see Section 5) [1].

In the second part, we cover the methodology for evaluating the elasticity achieved by the auto-scaler:

Modelling of Load Profiles:
In order to trigger and benchmark automated scaling events, a variable, representative, and reproducible load intensity profile is required. We show how to automatically a extract load intensity model from a given trace including how adapt such a load profile for their needs. We use for our workload scenario definition the load intensity modeling tool called LIMBO (see Section 6) [6].

Benchmarking of Elasticity:
The performance measurement and the comparison with other auto-scalers or different configurations is an important step in the development of such an auto-scaler. Hence, we discuss elasticity metrics to rate and rank the designed auto-scaler. For our elasticity measurements, we use the BUNGEE Cloud Elasticity Benchmarking Framework (see Section 7) [3].

2. WCF FORECASTING

As forecasting component of our proactive auto-scaler, we use the Workload Classification and Forecasting Tool [2]. It provides a set of automated forecasting methods based on time series analysis such as auto-regressive integrated moving averages with seasonality (sARIMA), extended exponential smoothing (ETS) or its extension tBATS for complex seasonal patterns, and a machine learning based-forecaster that leverages neural nets. For achieving a good forecast accuracy, WCF uses a decision tree for selecting dynamically a set of suitable methods. WCF allows the user to define objectives based on parameters such like forecast time horizon, confidence level and processing overhead limitation. Forecast results come with an accuracy estimate based the mean absolute scaled error (MASE) metric that helps to drop and repeat or trust a forecast result.

3. LIBREDE ESTIMATION

In order to estimate during run-time the amount of time that an individual request requires on a resource given the application and the current mix of request classes, the Library for Resource Demand Estimation [5] is used. Usually, for obtaining accurate resource demands, a fine-grained instrumentation of the software would be necessary. However, this instrumentation is likely to introduce high overheads and can distort their own measurement. Therefore, LibReDE supplies offline and online resource demand estimation techniques based on basic, aggregate monitoring data (e.g., utilization, number of request arrivals and average response times) with different state-of-the-art techniques based on the service demand law, linear regressions or Kalman filters. Also concepts from queuing theory are used.

4. DML MODEL

The Descartes Modeling Language (DML) [4] belongs to the architecture-level modeling languages for performance, quality-of-service, and resource management. It provides information about the performance behaviour and system properties. In addition, the configuration space and adaptation processes can be modelled. The structure and the four meshing models (resource landscape, application architecture, adaption points and adaption process) are depicted in Figure 1. Leveraging simulative what-if-analyses, DML allows to find an appropriate system configuration without changing the system. I.e., Chameleon uses a DML model instance of the controlled application continuously parameterized with the latest resource demand estimates to determine where and how to adapt the system resource configuration.

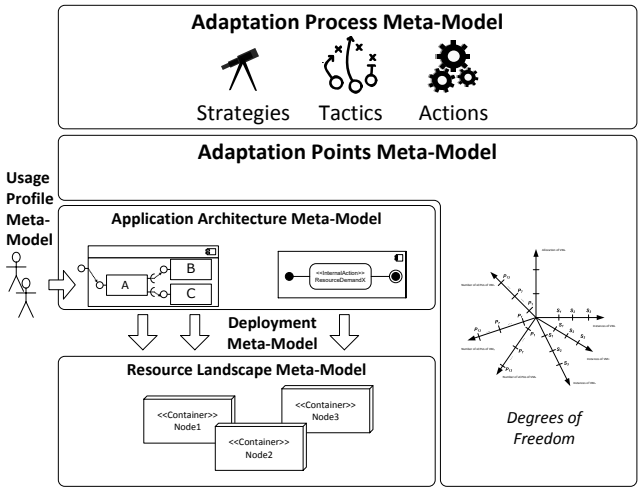

Figure 1: Structure of DML.

5. CHAMELEON AUTO-SCALER

Our proactive and application-aware auto-scaler consists of four components: (i) the controller, (ii) a performance data repository, (iii) the forecast component (here a modified version of WCF) and (iv) the resource demand estimation component (here LibReDE). The design and the flow of information is depicted in Figure 2. The central part of Chameleon is the controller. It communicates with the three remaining components and the managed cloud. The functionality of the controller is divided into two sub-tasks: monitoring and reconfiguration.

During the monitoring task, the controller communicates with the cloud and the performance data repository. This repository contains a time series storage and the DML descriptive performance model instance of the application. The controller periodically polls (e.g., every minute) information on the current state of the application from message queues. The collected information includes CPU utilization averages per node, average request residence times at the nodes, and the number of request arrivals. The new information is stored in the performance data repository for the recent time window. If the CPU utilization exceeds configurable thresholds, Chameleon scales the system reactively according to a standard threshold-based rule approach that is implemented as a fallback.

The reconfiguration of the system is planed proactively in longer intervals, e.g., 4 minutes, for a set of future scaling

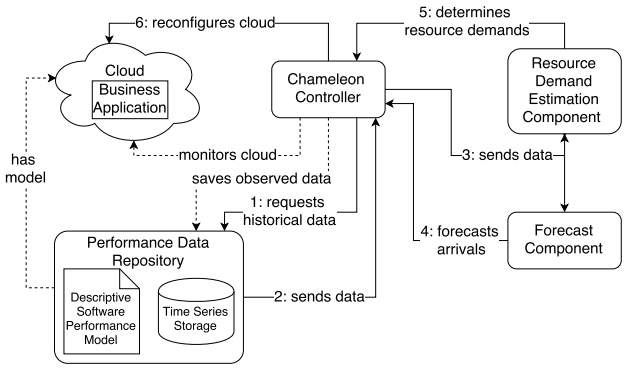

Figure 2: Design overview of Chameleon.

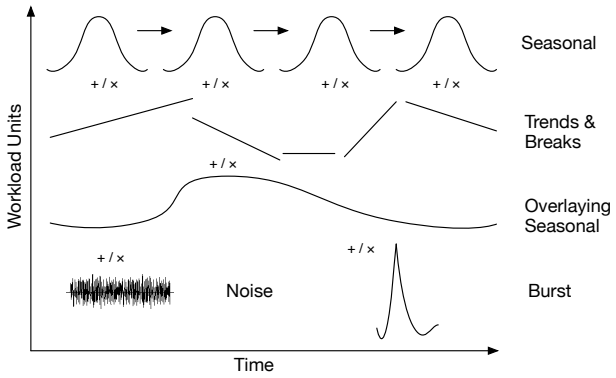

Figure 3: Elements of a load profile model.

intervals. It involves six tasks: (1) at first, the controller queries the performance data repository for the available historical data. (2) Then, the available time series data is send to the controller. (3) Afterwards, the time series of request arrival rates is forwarded to the forecast component and data about the CPU utilization and average request residence times per node is send to the resource estimation component. The forecast component predicts the arrival rates for the next reconfiguration intervals. If an earlier forecast result still contains predicted values for requested future arrival rates, no new time series forecast is computed. In case, a drift between the forecast and the recent monitoring data is detected, a new forecast is executed. (4) Then, the new forecast values are sent to the controller. (5) The resource demand estimation component estimates the time a single request needs to be served on the CPU of a node and sends the estimated value to the controller. (6) Finally, the controller scales the application deployment based on the estimated resource demands, the forecast arrivals and knowledge from the descriptive software performance model in case multiple tiers are scaled.

6. LIMBO LOAD MODELING

The LIMBO tool is capable to automatically extract from a given workload trace (that contains the amount of request arrivals per interval over a given period) a Descartes Load Intensity Model (DLIM) instance [6]. A DLIM model instance can be used to define any load intensity profile as it represents a tree of mathematical functions that are combined over the modeled time. Each instance captures the characteristics of the trace (e.g., trends, seasons, bursts, and noise, see Figure 3) that can be modified for creating a target dynamic work load scenario.

7. BUNGEE EXPERIMENT CONTROLLER

The BUNGEE Cloud Elasticity Benchmarking Framework measures the elasticity independently of the given platform performance. Therefore, it observes the given application under stress via a given load profile that is provided by LIMBO or as plain arrival rate data. The resulting workflow of BUNGEE is depicted in Figure 4 and is split into four phases:

1. System Analysis: BUNGEE constructs a mapping function for the relationship between a given load intensity and the associated demand for resources. The demand for resources is the minimal amount of resources that is required for handling the intensity without violating a predefined response-time SLO. This is done via a binary search per possible resource configuration.
2. Benchmark Calibration: BUNGEE creates based on the given load profile and the mapping function an identical set of demand changes on every platform under comparison.
3. Measurement: BUNGEE starts with a warm-up period and then runs the measurement with the predefined workload intensity profile, in which the maximum intensity and the experiment time can be adjusted. During the run, BUNGEE coordinates, times and monitors each request and gathers additional information on resource supply from the platform. At the end, a measurement sanity check is conducted.
4. Elasticity Evaluation: In this last step, BUNGEE calculates elasticity metrics and a score based on the observed supply changes and the generated demand changes:
   - Under-provisioning accuracy
     This metric describes the relative amount of resources that is under-provisioned during the measurement interval. I.e., acc_U is the number of resources that the system has too less in relation to the current demand divided by the experiment time.
   - Over-provisioning accuracy
     Analogous to the under-provisioning accuracy, this metric represents the relative amount of resources that are over-provisioned during the measurement interval. In other words, acc_O is the number of resources that the system has in excess in relation to the current demand divided by the experiment time.
   - Under-provisioning time share
     This metric captures the time in percentage, in which the system is under-provisioned during the experiment interval. I.e., ts_U is the time relative to the measurement duration, in which the system has to less resources.
   - Over-provisioning time share
     Analogous to the under-provisioning time share,

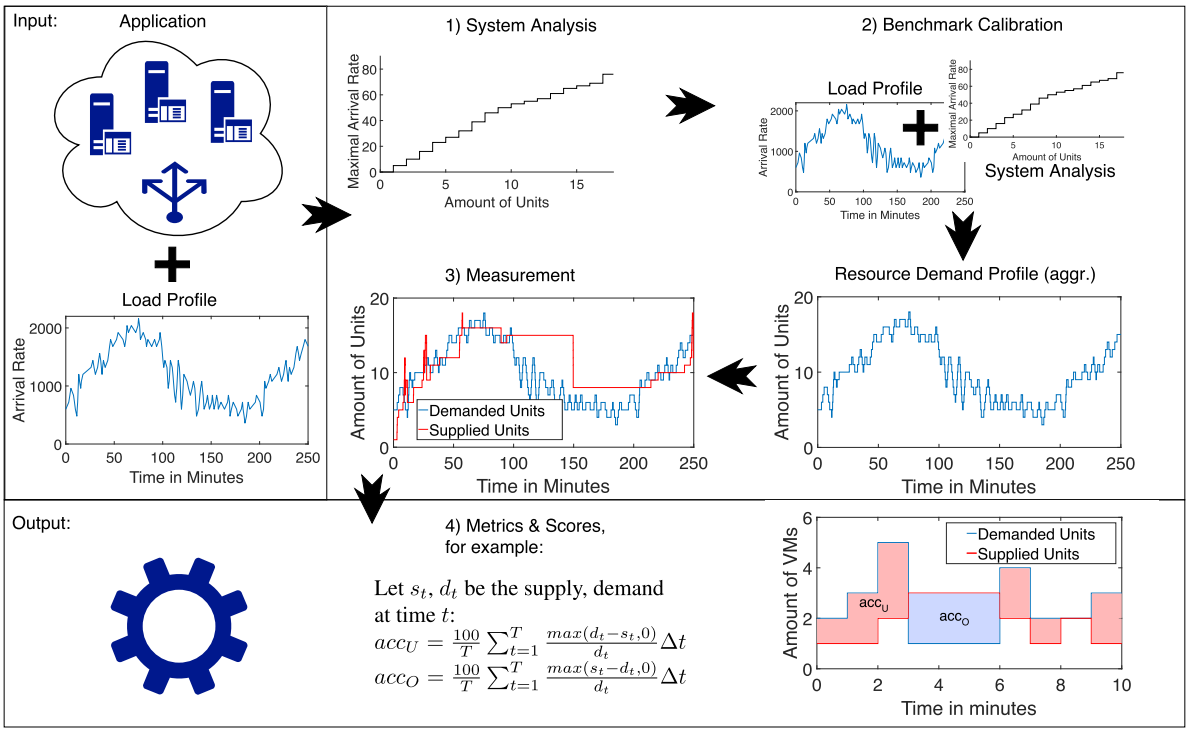

Figure 4: Operation principle of the BUNGEE elasticity benchmarking methodology.

this metric describes the time in percentage, in that the system is over-provisioned during the measurement interval. In other words, ts_O is the time relative to the measurement duration, in that the system has more resources than required.

- INSTABILITY
  The instability metric describes the percentage of time in which the supply and the demand curves are not changing in the same direction. I.e., *in* measures the fraction of time in which the demand change and the resource change have different signs.

The elasticity score of a cloud platform with a deployed elasticity mechanism is then computed via a weighted geometric mean of the aforementioned metrics normalized by a baseline for the given scenario. Further metrics, like the average/median response time, SLO violation percentage, average resources and the amount of adaptions are reported.

8. CONCLUSION

The tutorial addresses topics of central interest to the ICPE community such as run-time performance management using software performance models and estimation techniques. We demonstrate how theory in these disciplines can be brought into practice. The system properties of elasticity and scalability play a central role when designing and evaluating an auto-scaling mechanism. We touch new aspects of a benchmarking methodology to enable the rating of elastic properties.

Acknowledgments

This research is supported by the Research Group[2] of the Standard Performance Evaluation Corporation (SPEC)[3]. This work was co-funded by the German Research Foundation (DFG) under grant No. KO 3445/11-1.

9. REFERENCES

[1] A. Bauer. Design and Evaluation of a Proactive, Application-Aware Elasticity Mechanism. Master Thesis, University of Würzburg, September 2016. [Online] http://descartes.tools/chameleon.

[2] N. Herbst, N. Huber, S. Kounev, and E. Amrehn. Self-Adaptive Workload Classification and Forecasting for Proactive Resource Provisioning. *Wiley CCPE*, 26(12):2053–2078, March 2014.

[3] N. Herbst, S. Kounev, A. Weber, and H. Groenda. BUNGEE: An Elasticity Benchmark for Self-adaptive IaaS Cloud Environments. In *SEAMS 2015*, pages 46–56, Piscataway, NJ, USA, 2015. IEEE Press.

[4] S. Kounev, N. Huber, F. Brosig, and X. Zhu. A Model-Based Approach to Designing Self-Aware IT Systems and Infrastructures. *IEEE Computer*, 49(7):53–61, July 2016.

[5] S. Spinner, G. Casale, X. Zhu, and S. Kounev. LibReDE: A Library for Resource Demand Estimation. In *ACM/SPEC ICPE 2014*, pages 227–228, New York, NY, USA, March 2014. ACM Press.

[6] J. von Kistowski, N. Herbst, S. Kounev, and more. Modeling and Extracting Load Intensity Profiles. *ACM TAAS*, 11(4):23:1–23:28, January 2017.

[2]SPEC Research: http://research.spec.org
[3]SPEC: http://www.spec.org