Systematic Search for Optimal Resource Configurations of Distributed Applications

André Bauer, Simon Eismann, Johannes Grohmann, Nikolas Herbst, and Samuel Kounev
University of Würzburg, Germany
Email: andre.bauer@uni-wuerzburg.de, {firstname}.{lastname}@uni-wuerzburg.de

***Abstract*—With the advent of the micro-service paradigm, applications are divided into small, distributed parts. Knowledge of optimal resource configurations of such applications is required both for autonomic resource management as well as its assessment. Due to the high-dimensional search space of all possible configurations, the systematic measuring of the optimal configurations is challenging. To this end, we introduce a search algorithm based on hill-climbing for finding all optimal configurations in a feasible time and integrate it in an existing measuring framework. This approach enables the assessment, comparison and optimization of autonomic resource management approaches for micro-service applications. The evaluation shows that our approach is able to find all optimal configurations in the considered scenarios, while state-of-the-art multi-objective search algorithms do not.**

***Index Terms*—Elasticity, Benchmarking, Multi-objective search algorithms, Pareto optimal**

I. Introduction

In recent years, the micro-service paradigm has become very popular, along with the spread of DevOps practices and container technologies. This shift is also evident in the software industry, where micro-services are used more often than other software architecture models [1]. That is, prior applications are split now into small, individual pieces that are deployed separately. However, this paradigm introduces complexity for resource management due to a variety of services and their inherent possible resource configurations. By resource configuration, we mean the number of instances provided to each service. To counteract this complexity, autonomic management systems can be used to reconfigure the resource allocation of the application due to occurring load [2]. On the one hand, the customers pay for the resources; on the other hand, if an application runs on fewer resources than needed, the performance sharply drops below usability thresholds. Consequently, it is essential that the provided amount of resources is neither too high, nor too low. To this end, the adaption of the autonomic management systems is subjected to high-quality standards.

A popular measure for the quality of those adaptions is elasticity [3]. The core idea is to compare the provided and required resources. While the amount of provided resources can be easily monitored, the resources necessary for a given load can either be estimated or measured. For the measuring approach, the BUNGEE cloud elasticity benchmark [4] can be used. BUNGEE automatically stresses the application and determines a mapping between the load intensity, i.e., concurrent arrival rates, and the required resources. BUNGEE is limited to applications that consist of one service. As a consequence, we pose ourselves two research questions: (RQ1) *How can we adapt BUNGEE for the analysis of applications consisting of multiple services?* (RQ2) *How can the overhead of the search for optimal resource configurations be reduced?*

Towards addressing the questions above, our contribution is three-fold:

1) We formulate the search problem for optimal resource configurations as a multi-objective problem and use Pareto optimisation (Section IV-A).
2) We adapt BUNGEE and create a search algorithm based on hill-climbing to find all optimal resource configurations (Section IV-B).
3) We compare our algorithm with three state-of-the-art approaches in three different scenarios. In contrast to the state-of-the-art, our approach was able to find all optimal resource configurations.

II. Foundations

This section starts with highlighting the term elasticity. In Section II-B, the BUNGEE cloud elasticity benchmark is explained. Finally, the limitations of this tool are discussed.

A. Elasticity in Cloud Computing

In cloud computing, elasticity is commonly considered as a central characteristic of the cloud paradigm [3] and is defined as *"the degree to which a system is able to adapt to workload changes by provisioning and de-provisioning resources in an autonomic manner, such that at each point in time the available resources match the current demand as closely as possible"* [5].

Figure 1 shows the underlying idea of elasticity. The green dashed curve is the load intensity (i.e., the concurrent request rate), the solid black curve is the minimal resource demand to handle the load, and the dotted blue curve shows the supplied resources. The red areas labelled with a U represent the under-provisioning, i.e., the demand is higher than the supply. Analogously, the yellow areas marked with O represent over-provisioning.

B. BUNGEE

To evaluate the autonomous resource management for different cloud systems, user- and system-oriented metrics can be considered. In literature, there are many approaches on

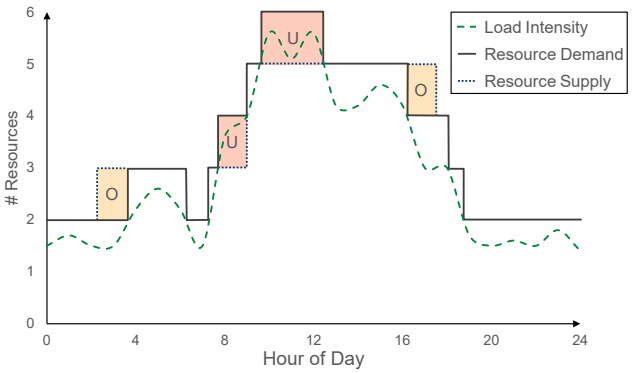

Fig. 1: The core idea of elasticity: **O**ver-provisioning (yellow boxes) and **u**nder-provisioning (red boxes) .

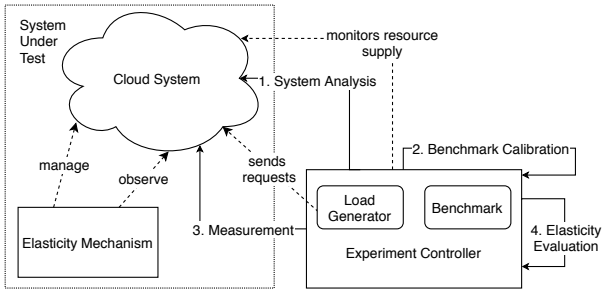

Fig. 2: BUNGEE Cloud Elasticity Benchmark.

how to measure the autonomous management quality at the system level. As we want to have comparable and precise measurements among different systems, we suggest to use the BUNGEE Cloud Elasticity Benchmark [4] as it considers general and cloud-specific benchmark requirements as stated in the work of Huppler et al. [6], [7] and Folkerts et al. [8].

1) Cloud Elasticity Benchmark: The working principle is depicted in Figure 2. On the left side, the system under test (SUT) is depicted. It contains the cloud that hosts the applications and the scaling controller. On the right side, the experiment controller (BUNGEE) with its four phases is illustrated:

1) **System Analysis:** The benchmark analyses the system under test (SUT) concerning the performance of its underlying resources and its scaling behaviour. That is, a discrete mapping function is generated, which determined for each load intensity the associated minimum amount of resources required to meet the service level objectives (SLOs).
2) **Benchmark Calibration:** The results of the analysis are used to adjust the load intensity profile injected on the SUT in a way that it induces the same resource demand on all platforms.
3) **Measurement:** The load generator exposes the SUT to a varying workload according to the adjusted load profile. The benchmark extracts the actual induced resource demand and monitors resource supply changes on the SUT.
4) **Elasticity Evaluation:** Elasticity metrics are computed and used to compare the resource demand and resource supply curves concerning different elasticity aspects.

2) System Analysis: In general, the system analysis works by exposing the system to a specific workload (load intensity) and checking whether all SLO are fulfilled. That is, BUNGEE obtains the resource demand for a specific request rate. More precisely, the system analysis maps the load intensity and the resource demand by executing a binary intensity search for all resource amounts starting with one resource unit and increasing the number of resources, until the maximum load does no longer increase or until the maximum number of resources is reached. The binary search is done by evaluating whether the SLO is violated for a starting load intensity and by then iteratively doubling or halving the load intensity until upper and lower bounds for the load intensity are found that are narrow enough. During each load intensity test, the application is stressed for 3 minutes with a constant request rate. Based on this information, the demand and supply curves can be evaluated. For the load generation, Apache JMeter[1] is used, with a plugin[2] that allows sending requests at specific timestamps. The timestamps are automatically generated from load-profiles that are modelled using the Descartes Load Intensity Model (DLIM) [9].

C. Limitation and Assumptions

The benchmark is restricted to horizontal scaling, and it is assumed that all services are deployed on homogeneous physical resources. Further, the original version of BUNGEE only supports applications which consists of one service, and the assumption is that all resources belong to the same service.

For applications with one service, there is only a limited number of possible resource configurations where the binary intensity search has to be applied. In contrast, an application with multiple services has at maximum $\prod n_i$ possible resource configurations (n_i is the maximum allowed number of resources for service i). So, it would take a long time to apply the binary intensity search for each configuration.

III. Related Work

Existing work related to this paper can be divided into elasticity metrics, approaches to measure the elasticity of a system, and configuration analysis approaches.

Over the years, a number of elasticity metrics based on the observed system response time have been proposed [10]–[13]. These metrics analyse how well an elastic system can keep an application available. However, these metrics do not account for over-provisioning and therefore favour systems that use more resources than necessary. Cost-based elasticity metrics rate the elasticity of a system from an economical perspective [8], [14]. The cost is calculated as a combination of the cost of resources and penalties for service-level agreement

[1]https://jmeter.apache.org/
[2]https://github.com/andreaswe/JMeterTimestampTimer

violations. However, these approaches tie the elasticity of a system to a specific cost model. Herbst et al. proposed a set of elasticity metrics, which calculate how much autonomic resource management deviates from the behaviour of an omniscient, theoretically optimal behaviour [5]. Based on this deviation, the timing, stability and accuracy aspects of an elastic system are quantified. These metrics are widely used in academia and endorsed by the SPEC Research Group [5]. Therefore, we focus on this set of elasticity metrics in this work.

The quality of automatic resource management is measured with the elasticity. A common approach is to evaluate the elasticity of such a management system by simulating its behaviour (simulation-based approaches are surveyed in [15]). Every simulation has to make assumptions about the behaviour of the application and therefore favours elasticity approaches that use the same set of assumptions as the simulation. Another approach is to measure the elasticity metrics through an intensive search. In a previous publication, we introduced BUNGEE, a framework for benchmarking the elasticity of cloud platforms [4]. A different benchmark that measures elasticity is the SPEC Cloud^{TM}IaaS 2016[3]. To the best of our knowledge, there is currently no approach that allows measuring the widely used elasticity metrics [5] for a micro-services on a real system.

The performance of an application depends on the underlying hardware, its parameterisation and the current load level. Measuring the performance of every combination of these influencing factors is infeasible. Therefore, a number of approaches have been proposed that use statistical inference or machine learning models to infer the performance of unmeasured configurations from a set of exploratory measurements [16]–[19]. Research in this area focuses on measurement point selection techniques and the applicability of different statistical inference or machine learning models. These approaches aim to predict the performance of all possible configurations and therefore tolerate a degree of inaccuracy. To calculate elasticity metrics, accurate measurements of the performance of only the ideal configurations are required.

IV. Approach

In this section, we first formulate the problem of how to find optimal resource configurations. Afterwards, our approach for finding these configurations is explained.

A. Problem Formulation

The first phase of BUNGEE is the system analysis in which a mapping between load intensity and optimal resource configuration is determined. By resource configuration, we mean the number of instances provided to each service. For applications consisting of one service it is sufficient to find the maximum load intensity for all possible resource configurations due to the small number of configurations. For applications containing multiple services however, as already noted in Section II-C, the approach of measuring each possible configuration would lead to a infeasible long measurement time in which the application cannot be used.

In the original version, the intensity-demand-mapping is used to find the smallest resource configuration that can handle a load intensity (i.e., a number of requests per second). For the use case of distributed micro-services however, it is not obvious how to define if one resource configuration should be considered smaller than another one. As an example, imagine an application consisting of three services: While it is clear that $c_1 = (1, 1, 1)$ is smaller than $c_2 = (2, 2, 2)$, it is not clear if $c_3 = (3, 1, 1)$ could be seen as smaller than $c_4 = (1, 2, 2)$. The problem with the comparison of c_3 and c_4 is that the first services of c_3 has more resources than the first services of c_4, while the second and third service has fewer resources.

Our approach is not to compare configurations per service but to consider the total amount of running resources per configuration. This also makes sense from a practical perspective, under the assumption that resources of all services are approximately equally expensive. Then, only the total number of resources is essential and fewer resources are preferred both by the user of a cloud service and the provider.

With this notion of which resource configurations are smaller than other configurations, we can now decide which configurations are optimal and should be part of the intensity-demand-mapping. But when is a resource configuration considered optimal? *A resource configuration with maximum load l is optimal (I) if there is no smaller configuration that can handle l and (II) no equally sized configuration that can handle load intensity higher than l.* The second condition ensures that if r resources are needed for a load level, then a most performant configuration with r resources is chosen. With this definition, it can be the case that multiple configurations are optimal, if they have equal size, have the same load processing capabilities, and all other smaller or equal sized configurations have smaller maximum load intensity. To this end, the function $demand(intensity)$ can be written as a function that maps load intensity to sets of configurations, because there can be multiple configurations that are optimal for a load level:

$$demand : \mathbb{R}_+ \rightarrow \mathcal{P}(\text{set of possible configurations},)$$

with $\mathcal{P}$ being the powerset (i.e., the set of all subsets of a set).

B. System Analysis Search Concept

One possible way to perform the search for optimal resource configurations is to model the problem as a multi-objective search problem and to use appropriate algorithms. They can be used for search problems where there are multiple objective functions and the desired output is a so-called Pareto set. In this set, a solution is considered to be Pareto optimal if and only if there is no other solution, which is equally good or better in all objective functions and better in at least one objective function.

To this end, objective functions are designed so that the Pareto set is the set of all configurations that are part of the

[3]https://spec.org/cloud iaas2016/

intensity-demand mapping. This can be done with a first objective function that prefers smaller resource configurations and a second objective function that favours higher maximum load intensity. The algorithms typically try to either minimise or maximise all objective functions at once. Hence, the objective functions cannot be chosen in a way in which lower values are better for one function while higher ones are better for the other function. However, this problem can be overcome by using the inverse value $1/f(x)$ of an objective function f.

In result, the objective functions for the resource configuration $r = (r_1, \ldots, r_n)$ for n services with maximum load l could be set to the inverse size of the configuration $f_1(r, l) = 1/\sum_{i=1}^{n} r_i$ and the maximum load $f_2(r, l) = l$. This means that there is no configuration with an equally good objective f_1 respectively f_2 and better objective f_2 respectively f_1. In other words, finding the Pareto set is equivalent to finding the configurations for the intensity-demand-mapping.

Another way to find the optimal resource configuration is the use of local search algorithms. To this end, we developed a search algorithm based on hill-climbing that (i) stores multiple current states like a local-beam-search algorithm and (ii) can be configured to continue the search even if the direct neighbours are not better than the current nodes. The neighbours of a configuration are the set of configurations that can be created by adding one resource to one of the services.

The pseudo-code for our adapted hill-climbing algorithm called multi-hill-climbing is shown in Algorithm 1. The parameter *maxDepth* controls how often the algorithm will expand all neighbouring nodes, even if no node was found that is better than the best current node. The parameter *activeNodes* determines, how many of the nodes found during an iteration are added to the set of possible solutions and further expanded in the next iteration. A value of *activeNodes* $= 1$ means, the algorithm will only continue with the best node, with a value of n, the algorithm will continue with the best node and the $(n-1)$-best nodes. Both parameters are not changed during during the search.

Algorithm 1: Multi-hill-climbing

Input: a config. problem, *maxDepth*, *activeNodes*
Result: a set of resource configurations

```
1  solutions ← ∅
2  current ← makeSet(smallest resource config.)
3  while current ≠nil do
4  |   solutions ← solutions ∪ current
5  |   current ← findNextBestPoints(current)
6  solutions ← non-dominated configs. in solutions
7  return solutions
8
9  function findNextBestPoints(current)
10 |   bestScore ← best load intensity in current
11 |   neighbours ← expand(current)  // neighbours
   |    of current (incl. current)
12 |   for depth = 1 to maxDepth do
13 |   |   bestPoints ← activeNodes-many points from
   |   |    neighbours sorted by load intensity
14 |   |   if no node in bestPoints is better than
   |   |    bestScore then
15 |   |   |   neighbours ← expand(neighbours)
16 |   |   else
17 |   |   |   return bestPoints
18 |   return nil
```

V. Evaluation

In this section, we first introduce the measured scenarios and cloud environment. Afterwards, the algorithms in competition are highlighted. In Section V-D, we investigate how the search parameters influence our approach. Finally, we compare our approach against the state-of-the-art.

A. Cloud Environment and Scenarios

The experiment setup consists of multiple components: the benchmark application, a load balancer, and the load generator. Our experimental environment uses CloudStack[4] for managing virtualised KVM-server hosts in a cluster of identical servers (HP DL160 Gen9 with eight physical cores @2.4Ghz and 32GB). Three of them are reserved for the application and three additional servers to host the load balancer (Traefik[5]), the CloudStack management system, and BUNGEE. For the evaluation, we use up to 12 virtual machines (Debian 4.9.110 with 2 vCPUs and 8 GB) on which the application is deployed.

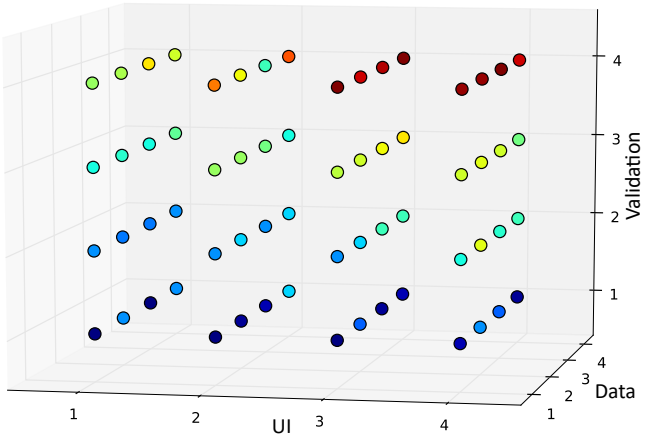

Fig. 3: Measured load-intensities of the application.

The stressed benchmark application represents a lightweight micro-service application with three different services: (i) UI service, (ii) validation service, and (iii) data service. The application is written in Java and is deployed on a Tomcat server (v8.5). The load generator requests data by sending HTTP requests to the UI service. The UI forwards each request to the validation service for checking its validity. After that, the request is redirected to the data service. This service provides the requested data and sends the response to the UI for rendering the content. The measured maximal

[4]CloudStack: https://cloudstack.apache.org/
[5]Traefik: https://traefik.io/

capable load intensity of the application is depicted in Figure 3 where blue colours mean lower load intensities and red colours mean higher load intensities. Each axis shows the amount of instances per service.

To test the system analysis in a big scale, we simulate an application with the following formulas: the maximum load intensity for the i-th service with r_i resources was modelled with an equation such as:

$$maxload_i(r_i) = (r_i \cdot x_i)^{p_i}$$

The overall performance of the application with configuration $(r_1, \ldots, r_n)$ is then modelled as the performance of the least performant service:

$$maxload(r_1, \ldots, r_n) = \min_{i=1,\ldots,n} maxload_i(r_i)$$

To be comparable with the application, we simulate three services and set the following parameters for scenario *Simulation I* and *Simulation II*: $x_I = (10, 15, 8)$, $p_I = (0.9, 1, 0.95)$, $x_{II} = (4, 2, 8)$ and $p_{II} = (0.9, 0.8, 0.5)$. These configurations are set to be a convincing model that could also realistically come from a real-world application: Some services act as bottlenecks that have to be scaled before other services can be scaled. Further, if the resource configurations are sorted by their maximum load, a conspicuous pattern emerges: each configuration is identical to its predecessor except for one service, where the resources have increased.

B. Algorithms in Competition

Different genetic algorithms for multi-objective problems are discussed in [20], [21], [22]. These algorithms are all specialised genetic algorithms using selection, crossover and mutation operations allowing the approximation of the population to the Pareto optimal set. In this evaluation, we focus on NSGAII [23], SPEA2 [24], and IBEA [25] as they were most often used in comparisons such as in [21].

As these algorithms depend on their hyper parameters, we use for each algorithm different configurations and report in the following only the best results. We change the parameters: crossover- and mutation-probability (cp, mp) and distribution-index (cdi, mdi), the maximum number of iterations (mi), and the population size (ps). In other words, each algorithm can be characterized by the vector $p = (cp, cdi, mp, mdi, mi, ps)$. That is, the vectors are $p_N = (0.0, 20.0, 1.0, 20.0, 2000, 20)$ for NSGAII, $p_S = (0.0, 20.0, 1.0, 20.0, 200, 40)$ for SPEA2, and $p_I = (0.1, 10.0, 1.0, 20.0, 1000, 30)$ for IBEA.

C. Investigation of the Parameters

As mentioned in Section IV-B, we developed a search algorithm called multi-hill-climbing and add two parameters *maxDepth* and *activeNodes*. Hence, we investigate how the parameters influence the search performance of the algorithm. To this end, we change both parameters while searching the optimal resource configuration for the application. Table I shows the percentage of found optimal configurations, the ratio between configurations that are claimed as optimal (false positive) and optimal configurations (true positive), and the number of visited resource configurations while changing both parameters. While *activeNodes* is set to one, the algorithm continues the search in each step from the best node. That is, the algorithm may not find all optimal configurations as the next optimal solution may not be a direct neighbour of the current optimal configuration. In Table I, this effect occurs and the algorithm only finds 25% of the optimal resource configurations. Further, it can be seen that with increasing *maxDepth*, the number of visited configurations increases. In a nutshell, the choice of the parameters for the multi-hill-climbing algorithm has a significant impact on its performance: if the values of *maxDepth* or *activeNodes* are chosen too low, then the algorithm does not find all optimal configurations. At the same time, the number of searches can increase if higher values for the parameters are used.

maxDepth	*activeNodes*	Found	Wrong/Found	Visited
1	1	0.25	0.0	13.0
2	1	0.25	0.714	25.0
1	2	1.0	0.0	25.0
2	2	1.0	0.0	28.0
3	2	1.0	0.0	29.0
3	3	1.0	0.0	32.0

TABLE I: Investigation of the influence of the parameters.

D. Experiment Results

We compare the search of our multi-hill-climbing algorithm *MHC* (*maxDepth* is set to 1 and *activeNodes* to 2) with the multi-objective algorithms IBEA, NSGAII and SPEA2. To this end, we conduct real-world experiments with the application (64 possible resource configurations) and simulate two further applications (each 1000 possible resource configurations). Each measurement is repeated 100 times, and the averages are reported in Table II. For each experiment, the percentage of found optimal configurations, the ratio between configurations that are claimed as optimal (false positive) and optimal configurations (true positive), and the number of visited resource configurations are listed. For example, the SPEA2 algorithm finds in the Simulation I 90.1% of the optimal configurations, reports no wrongly classified optimal configuration, and measured 216.12 out of 1000 configurations. In contrast, our approach is able to find all optimal configurations, has no false positives, and only visited 116 configurations. Across all scenarios, our approach finds all optimal configurations, while the other algorithms find at most 90.1%, 98.5% and 82.2%. Also, our approach has no false positives while the other algorithms have a low number of false positives and our approach also has fewer visits.

VI. Future Work

Although our approach is able to find the optimal resource configurations, there is still the challenge of how to rate a unknown, non-optimal configuration: As our approach and general multi-objective optimization approaches only measures a small subset of the vast configuration space, it is impossible to compare a non-optimal resource configuration

	Simulation I			Simulation II			Application		
Name	Found	Wrong/Found	Visited	Found	Wrong/Found	Visited	Found	Wrong/Found	Visited
MHC	1	0	116.0	1	0	60.0	1	0	25.0
IBEA	0.513	0.002	138.34	0.677	0.004	150.24	0.828	0.012	33.0
NSGAII	0.65	0.015	145.15	0.742	0.003	144.39	0.798	0.01	30.84
SPEA2	0.901	0	216.12	0.985	0	209.13	0.779	0.004	43.49

TABLE II: Comparison between the search algorithms.

that are not measured to an optimal resource configuration. As part of our future work, we are looking into estimating the performance of unseen configurations based on currently measured configurations. This estimation will enable the comparison across autonomic resource management algorithms that do not achieve optimal resource configurations.

VII. Conclusion

Nowadays, with the micro-service paradigm, applications are split into small pieces and are deployed distributed. In the context of automatic resource management, this paradigm poses the challenge of finding the optimal resource configurations due to the vast search space of all possible configurations. In this article, we tackle the problem of finding the optimal resource configurations in such an environment. To this end, we extend the system analysis of BUNGEE by introducing a search algorithm based on hill-climbing. In the evaluation, our approach is able to find all optimal configurations and outperforms state-of-the-art multi-objective search algorithms. The knowledge of these optimal configurations enables to quantify, compare and optimise autonomic resource management approaches for a micro-service application using established elasticity metrics.

Acknowledgments

The authors would like to thank Marcus Wilhelm for his contribution of the experimental evaluation.

References

[1] M. Viggiato and more, "Microservices in practice: A survey study," *arXiv preprint arXiv:1808.04836*, 2018.

[2] A. Bauer, N. Herbst, S. Spinner, A. Ali-Eldin, and S. Kounev, "Chameleon: A Hybrid, Proactive Auto-Scaling Mechanism on a Level-Playing Field," *IEEE Transactions on Parallel and Distributed Systems*, vol. 30, no. 4, pp. 800 – 813, April 2019.

[3] D. C. Plummer and more, "Study: Five Refining Attributes of Public and Private Cloud Computing," Gartner, Tech. Rep., 2009.

[4] N. R. Herbst, S. Kounev, A. Weber, and H. Groenda, "BUNGEE: An Elasticity Benchmark for Self-Adaptive IaaS Cloud Environments," in *Proceedings of the 10th International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS 2015)*, May 2015.

[5] N. Herbst and more, "Quantifying Cloud Performance and Dependability: Taxonomy, Metric Design, and Emerging Challenges," *ACM Transactions on Modeling and Performance Evaluation of Computing Systems (ToMPECS)*, vol. 3, no. 4, pp. 19:1–19:36, August 2018.

[6] K. Huppler, "Performance Evaluation and Benchmarking." Berlin, Heidelberg: Springer-Verlag, 2009, ch. The Art of Building a Good Benchmark, pp. 18–30.

[7] K. Huppler, "Benchmarking with your Head in the Cloud," in *Technology Conference on Performance Evaluation and Benchmarking*. Springer, 2011, pp. 97–110.

[8] E. Folkerts and more, "Benchmarking in the Cloud: What It Should, Can, and Cannot Be," in *Selected Topics in Performance Evaluation and Benchmarking*, ser. LNCS, 2012, vol. 7755.

[9] J. v. Kistowski, N. R. Herbst, and S. Kounev, "Modeling variations in load intensity over time," in *Proceedings of the Third International Workshop on Large Scale Testing*, ser. LT '14. New York, NY, USA: ACM, 2014, pp. 1–4.

[10] C. Binnig, D. Kossmann, T. Kraska, and S. Loesing, "How is the weather tomorrow?: Towards a benchmark for the cloud," in *Proceedings of the Second International Workshop on Testing Database Systems*, ser. DBTest '09. New York, NY, USA: ACM, 2009, pp. 9:1–9:6.

[11] B. F. Cooper and more, "Benchmarking cloud serving systems with ycsb," in *Proceedings of the 1st ACM Symposium on Cloud Computing*, ser. SoCC '10. New York, NY, USA: ACM, 2010, pp. 143–154.

[12] T. Dory, B. Mejías, P. Roy, and N.-L. Tran, "Measuring elasticity for cloud databases," in *Proceedings of the The Second International Conference on Cloud Computing, GRIDs, and Virtualization*. Citeseer, 2011, pp. 37–48.

[13] M. Becker, S. Lehrig, and S. Becker, "Systematically deriving quality metrics for cloud computing systems," in *Proceedings of the 6th ACM/SPEC International Conference on Performance Engineering*, ser. ICPE '15. New York, NY, USA: ACM, 2015, pp. 169–174.

[14] S. Islam, K. Lee, A. Fekete, and A. Liu, "How a consumer can measure elasticity for cloud platforms," in *Proceedings of the 3rd ACM/SPEC International Conference on Performance Engineering*, ser. ICPE '12. New York, NY, USA: ACM, 2012, pp. 85–96.

[15] A. V. Papadopoulos and more, "Peas: A performance evaluation framework for auto-scaling strategies in cloud applications," *ACM Trans. Model. Perform. Eval. Comput. Syst.*, vol. 1, no. 4, pp. 15:1–15:31, Aug. 2016.

[16] M. Faber and J. Happe, "Systematic adoption of genetic programming for deriving software performance curves," in *Proceedings of the 3rd ACM/SPEC International Conference on Performance Engineering*. ACM, 2012, pp. 33–44.

[17] Q. Noorshams, D. Bruhn, S. Kounev, and R. Reussner, "Predictive performance modeling of virtualized storage systems using optimized statistical regression techniques," in *Proceedings of the 4th ACM/SPEC International Conference on Performance Engineering*, ser. ICPE '13. New York, USA: ACM, 2013, pp. 283–294.

[18] D. Westermann, J. Happe, R. Krebs, and R. Farahbod, "Automated inference of goal-oriented performance prediction functions," in *Proceedings of the 27th IEEE/ACM International Conference on Automated Software Engineering*. IEEE, 2012, pp. 190–199.

[19] E. Thereska, B. Doebel, A. X. Zheng, and P. Nobel, "Practical performance models for complex, popular applications," *SIGMETRICS Performance Evaluation Review*, vol. 38, no. 1, pp. 1–12, 2010.

[20] R. Marler and J. Arora, "Survey of multi-objective optimization methods for engineering," *Structural and Multidisciplinary Optimization*, vol. 26, no. 6, pp. 369–395, 2004.

[21] C. von Lücken, B. Barán, and C. Brizuela, "A survey on multi-objective evolutionary algorithms for many-objective problems," *Computational Optimization and Applications*, vol. 58, no. 3, pp. 707–756, 2014.

[22] A. Zhou and more, "Multiobjective evolutionary algorithms: A survey of the state of the art," *Swarm and Evolutionary Computation*, vol. 1, no. 1, pp. 32–49, 2011.

[23] K. Deb, S. Agrawal, A. Pratap, and T. Meyarivan, "A fast elitist non-dominated sorting genetic algorithm for multi-objective optimization: Nsga-ii," in *International conference on parallel problem solving from nature*. Springer, 2000, pp. 849–858.

[24] E. Zitzler, M. Laumanns, and L. Thiele, "Spea2: Improving the strength pareto evolutionary algorithm," *TIK-report*, vol. 103, 2001.

[25] E. Zitzler and S. Künzli, "Indicator-based selection in multiobjective search," in *International Conference on Parallel Problem Solving from Nature*. Springer, 2004, pp. 832–842.