Baloo: Measuring and Modeling the Performance Configurations of Distributed DBMS

Johannes Grohmann
University of Würzburg
Würzburg, Germany
johannes.grohmann@uni-wuerzburg.de

Daniel Seybold
Ulm University
Ulm, Germany
daniel.seybold@uni-ulm.de

Simon Eismann
University of Würzburg
Würzburg, Germany
simon.eismann@uni-wuerzburg.de

Mark Leznik
Ulm University
Ulm, Germany
mark.leznik@uni-ulm.de

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

Jörg Domaschka
Ulm University
Ulm, Germany
joerg.domaschka@uni-ulm.de

***Abstract*—Correctly configuring a distributed database management system (DBMS) deployed in a cloud environment for maximizing performance poses many challenges to operators. Even if the entire configuration spectrum could be measured directly, which is often infeasible due to the multitude of parameters, single measurements are subject to random variations and need to be repeated multiple times.**

In this work, we propose Baloo, a framework for systematically measuring and modeling different performance-relevant configurations of distributed DBMS in cloud environments. Baloo dynamically estimates the required number of measurement configurations, as well as the number of required measurement repetitions per configuration based on a desired target accuracy. We evaluate Baloo based on a data set consisting of 900 DBMS configuration measurements conducted in our private cloud setup. Our evaluation shows that the highly configurable framework is able to achieve a prediction error of up to 12%, while saving over 80% of the measurement effort. We also publish all code and the acquired data set to foster future research.

***Index Terms*—Performance Modeling, Distributed Database Management Systems, Parameter Optimization, Machine Learning, Cloud Computing**

I. Introduction

Many software systems offer parameters to configure internal workings. This allows performance engineers to influence non-functional properties and with them application performance. Yet, these configurations are not always easy to interpret and interdependencies between different configuration options are hard to oversee. In consequence, engineers traditionally rely on rules of thumb and testing. Performance prediction of configurable systems [1]–[4] aims at automating their modeling and optimization and providing engineers with a scientifically approved toolbox for taking configuration decisions.

Database Management Systems (DBMS) are particularly challenging configurable software systems: Relational DBMS offer a wide range of configuration options that expose a multitude of interdependent knobs to tweak the DBMS performance for a specific workload. To address the complexity of such DBMS optimization problems, many different approaches for finding the best possible DBMS configuration exist [5]–[7].

Beyond relational DBMS, NoSQL and NewSQL DBMS exploit distributed architectures that provide more non-functional properties including horizontal scalability, elasticity, and availability [8], [9], which are determined by additional configuration options such as cluster size and replication factor.

For operating distributed DBMS cloud resources have become the preferred infrastructure, as they provide scalability and elasticity on resource level [10]. Alas, this operational model further increases the configuration space adding cloud-related dimensions such as resource type, storage backend, and others. In addition to understanding the performance impact of each individual configuration, it is necessary to also understand the interdependencies between parameters in the overall configuration space. This is extremely challenging even for experts in the three domains and therefore demands for supportive methods covering the entire configuration space. Due to its size one cannot simply evaluate each and every configuration option, but instead need to improve decision-making by predicting the performance of all configurations using a subset of measurements and hence, decreasing overhead.

Related work on performance prediction of cloud-hosted DBMS focuses on single-node DBMS ignoring distribution [11], [12]; targets only specific DBMS technologies ignoring cloud resource characteristics [13]; or considers only cloud resources without DBMS characteristics [14].

A core challenge when dealing with performance models of distributed DBMS is the time-intensive and expensive generation of the underlying data set: (1) Measurements of single configuration points are costly, as it requires a cluster of cloud resources that need to be reserved during the entire measurement period. (2) Performance measurements of distributed DBMS have a high variability [15] and therefore need to be repeated multiple times to achieve statistical significance. (3) The thorough evaluation of a performance prediction approach for any configurable system requires measurements of every possible configuration; an exponentially growing space.

Our contributions in this paper are as follows:

1) We present a novel framework for the measurement and

modeling of arbitrarily complex configuration spaces of configurable software systems. The design of this approach is specifically targeted at distributed DBMS: Our approach (1) selects a suitable robust statistical measure for the given scenario, (2) determines the minimal required number of measurement repetitions for a given measurement point, (3) chooses the next required measurement point, and (4) constructs a model of the configured parameter space using Machine Learning.

2) We evaluate our approach using a data set with measurements of a distributed DBMS with various configuration options. We publish this reference data set[1] comprising a total of around 450 measurement hours and roughly 9,450 computing hours in a private cloud environment.

By modeling the whole configuration space, our approach can quickly extrapolate expected performance results for a given configuration without actually measuring it. This is a strong benefit over a naive black-box optimization search. Hence, the resulting DBMS performance configuration model provides the foundation for selecting a performance-optimal operations and deployment configuration of a configurable system. Besides finding the most performant configuration, it gives a better understanding of the entire configuration space, providing valuable insights for operators and architects when trading performance against other non-functional aspects such as security, reliability, and costs.

The generic nature of the proposed framework enables researchers and practitioners to configure, adapt, and modify our approach as well as to transfer it to other domains. Our publicly available data set supports other researchers analyzing the performance behavior of the investigated DBMS in detail and evaluate further approaches for performance prediction.

The rest of this paper is structured as follows. First, Section II clarifies requirements before we introduce the Baloo approach in Section III. Section IV presents an evaluation of Baloo including a description of our data set. Finally, we list the related work in Section V. Section VI concludes the paper.

II. Performance Engineering Requirements

For operating a distributed DBMS in volatile, cloud-like environments, operators can chose between multiple DBMS, cloud providers, cloud resources, and several configuration options of the DBMS including cluster size, replication factor, and client consistency. A performance model takes as input a possible configuration from that configuration space and outputs the expected performance of this configuration. As such, a performance model helps operators to answer various questions on performance impact of changes to the configurations (adding/removing resources, changing the consistency model, changing the size of the resources), but also changes in use as caused by, e.g., a changed read/write ratio. For DBMS, performance is either measured in latency or throughput.

In an ideal world, the operator knew the performance of all the different possible configurations, but in practice, this is not the case. In consequence, an operator can estimate the performance of a single configuration by benchmarking that configuration with a workload as close as possible to the expected workload. In the following, we refer to a single configuration as a *configuration point* and to an execution of a benchmark as a *measurement run*.

Requirement 1 (Comparability): In order to judge which configuration points are better, it is necessary to compare the outcomes of the measurement runs for different configuration points. Yet, this comparison is difficult, as the raw results of a measurement run is a time series of throughput and latency values. Instead, a higher-level metric is required that captures the quality of a measurement run for a configuration point.

Requirement 2 (Repeatability): Cloud-like environments are characterized by volatility in performance so that multiple independent measurement runs are required to conclude on the actual performance of a configuration point.

Requirement 3 (Prediction): Being able to predict the performance of unmeasured configuration points helps reducing required time and costs for creating the performance model.

Requirement 4 (Trade-off between quality and cost): Obviously, the more measurement runs per configuration point and the more configuration points are measured, the more precise the performance model will be. Yet, the more measurements, the higher the time and financial impact. Hence, an approach like Baloo needs to offer the capability to flexibly decide on the required quality and hence, costs.

Requirement 5 (Validation): In order to validate Baloo an extensive data set with DBMS performance measurements is required that needs to contain multiple measurements for different configuration points. Due to the fact that such data sets are hard to obtain from real-world applications such a data set needs to be based on synthetic data, but still make use of realistic configurations.

III. Approach

Figure 1 shows a graphical overview of our approach, Baloo. The overall goal of Baloo is to run as few measurements as possible for each configuration point and to use as few configuration points as possible to train an accurate performance model. This performance model then enables the performance prediction of arbitrary configuration points without explicitly measuring them.

The framework is initialized with the *Robust Metric Selection* (offline phase) that provides the means to summarize the results of individual measurement runs. The goal of this phase is to determines the robust metric that—based on a large data set of existing performance measurements—best reduces noise in the measurement data. Once the robust metric has been found, performance engineers can request the creation of a performance model from Baloo. For creating the performance model, Baloo generates a training data set in a workflow with three steps: (i) Baloo selects crucial configuration points from the configuration space; (ii) Baloo decides how often each of the configuration points needs to be measured; (iii) Baloo triggers the measurements using the Mowgli measurement

[1] https://doi.org/10.5281/zenodo.3854996

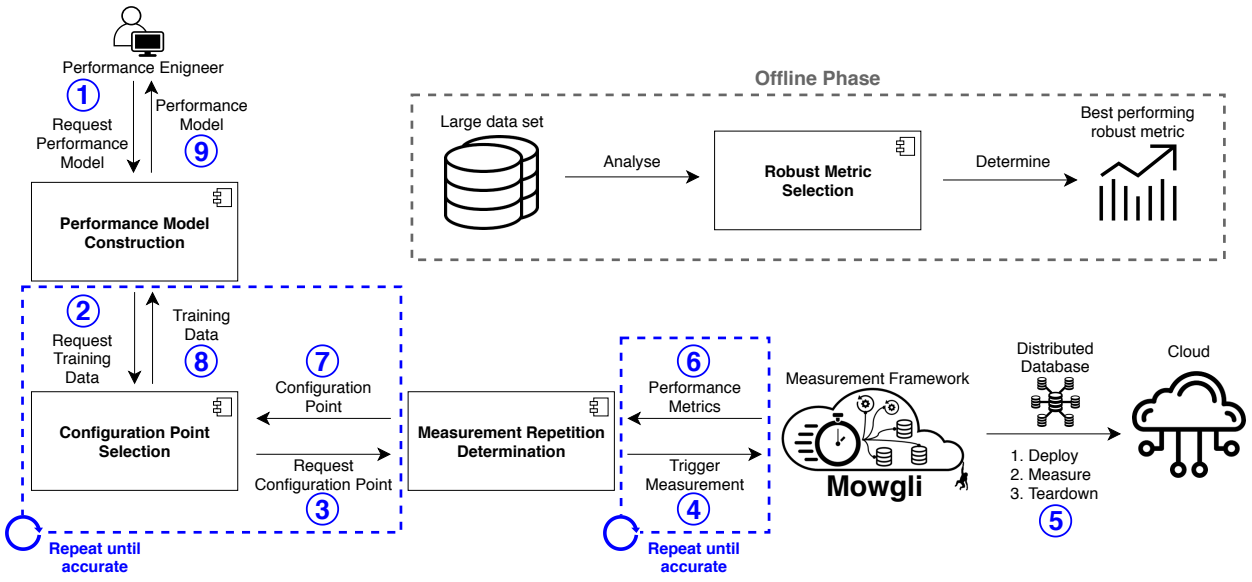

Fig. 1. Overview of Baloo. The offline phase is executed once, the blue steps are executed whenever a performance model is requested.

framework [16]. For all of these steps, Baloo applies statistical heuristics to decide how many measurements to conduct per configuration point and further to decide which and how many configuration points to chose.

In the following, we describe *Robust Metric Selection*, *Distributed DBMS Performance Measurement*, *Measurement Repetition Determination*, *Measurement Point Selection*, and *Performance Model Construction* in more detail.

A. Robust Metric Selection

This step addresses the Comparability Requirement (see Requirement 1 in Setion II) by selecting a suitable metric for summarizing the performance time series of one performance measurement. This is required, since distributed DBMS operated in volatile environments are subject to performance variations and fluctuations in latency and throughput measurements during one experiment run. We therefore need metrics to summarize each run. Furthermore, those metrics need to enable a robust comparability between different configuration points. The most important characteristic of a metric in our scenario is its robustness to the given fluctuations. Hence, the name *robust metric*. The "ideal" robust metric reports the same values for two repetitions of the same configuration point.

For finding the best robust metric, we compare different robust metrics candidates by analyzing their coefficient of variation (CV), a measure of the standard deviation in relation to the sample mean, over all measurement runs available. Based on the resulting list of CVs for each individual measurement, we rate each metric using the mean and the standard deviation of their respective CV-scores. The metric observing the lowest mean CV is then defined to be the most suitable robust metric for the given scenario.

For potential robust metrics, we investigate common robust measures of central tendency from literature [17]: mean, median, different percentiles, trimmed mean, winsorized mean, trimean, and the hodges-lehman estimator. A full analysis of all robust metrics is presented in Section IV-B.

B. Distributed DBMS Performance Measurement

This step lays the basis for the following steps as it generates the data for individual measurement runs (step 5 in Figure 1). For doing so, Baloo builds on the open source and extensible DBMS evaluation framework Mowgli[2] [16] that supports the design and execution of DBMS evaluations. Mowgli allows to define relevant domain-specific properties and allow their specification based on the supported technologies such as the DBMS itself, cluster size, replication factor, cloud provider, resource capacity, and workload type; Furthermore, it fully automates allocation of cloud resources, deployment and configuration, workload generation, calculation of performance data, and processing of results. The resulting evaluation data (step 6) sets contain the benchmark metrics, monitoring data, resource metadata, and execution logs of the respective evaluation tasks.

C. Measurement Repetition Determination

This step addresses Requirement 2 and Requirement 4 of Section II as it decides on the number of required measurements for a specific configuration point P. Here, the overall number of measurements is dependent on the desired confidence t_c and stability of the measurements, which is dependent on the volatility of the environment.

Baloo requires at least two measurements in order to judge on the stability of the results, and uses a configurable upper limit n_{max} in order to control execution time and costs. Details are presented in Algorithm 1.

First, the set M of performance measurements is filled by two consecutive calls of `triggerMeasurement(P)` (step 4)

[2]https://omi-gitlab.e-technik.uni-ulm.de/mowgli

Algorithm 1: Measurement Repetition Determination.

Input: Desired configuration point P,
target confidence threshold t_c,
maximum number of measurements n_{max}.
Output: Obtained measurement value m

1 M = triggerMeasurement(P)
2 **do**
3 $\quad M = M \cup$ triggerMeasurement(P)
4 $\quad M' =$ removeOutliers(M)
5 $\quad m =$ aggregate(M')
6 $\quad c =$ confidence(M')
7 **while** $c > t_c$ **and** $|M| < n_{max}$
8 **return** m

Algorithm 2: Performance model construction

Input: Configuration space definition S,
target score threshold t_s,
maximum configurations ratio r_{max}.
Output: Performance model p

1 $C =$ getInitMeasurements(S, i)
2 $p =$ constructPerformanceModel(C)
3 $s =$ scorePerformanceModel(C, p)
4 **while** $s < t_s$ **and** $|C| < r_{max} \cdot |S|$ **do**
5 $\quad C = C \cup$ addMeasurements(S, C, r)
6 $\quad p =$ constructPerformanceModel(C)
7 $\quad s =$ scorePerformanceModel(C, p)
8 **return** p

that wraps invocations to Mowgli as described in Section III-B. In lines 4–7, the obtained measurement values are analyzed and aggregated. If the calculated confidence c deceeds t_c or if n_{max} has been reached, the calculated aggregation m is returned (step 7).

Baloo allows different implementations of outlier detection, aggregation, and confidence estimations (cf. lines 4–7). Our implementation used in Section IV applies outlier detection based on isolation forests [18] through the Python version of scikit-learn [19] with two isolation trees. We use the mean as aggregation function and quantify confidence through the coefficient of variation (CV). The confidence threshold t_c is set to 0.02. Hence, the loop stops if $CV(M')$ <0.02. Note that both median and CV are not calculated on M, but rather on M' that does not contain outliers.

D. Configuration Point Selection

This step selects the next configuration point (step 3) that needs to be added to the training set (cf. Requirement 4). This process is commonly referred to as *sampling* [20], while the underlying selection strategy is called *sampling method*.

While our framework supports any strategy for the selection of the next configuration point, the implementation used in Section IV is based on uniformly distributed random sampling that was found leading to the most accurate performance models in general [21], but not always [22]. Our random implementation relies on enumerating the entire configuration space, which does not pose a problem in our scenario. Otherwise, more sophisticated solutions are required, e.g., based on binary decision diagrams [23] or satisfiability solvers [24].

E. Performance Model Construction

This step is concerned with the construction of the actual performance model for a configuration space S and, therefore, addresses Requirements 3 and 4 of Section II. Its iterative approach determines the minimal required number of measurements for achieving a configurable target accuracy t_S. A configurable maximum ratio r_{max} of S is used as configurable upper limit in order to control execution time and costs.

Here, S is the explorable part of the overall configuration space, i.e. the Cartesian product of all available feature values, while t_s is the targeted score.

Algorithm 2 shows that first a set of initial measurements is conducted (line 1) based on the given configuration space (step 2). All measurements (step 8) constitute the set of available measurements C, from which a performance model p is built and scored using an internal scoring function. The i parameter to `getInitMeasurements` determines ratio of S used.

The algorithm keeps adding measurement points in line 5 until $s \geq t_s$ or $\frac{|C|}{|S|} \geq r_{max}$. In each iteration, it recomputes p (line 6) and s (line 7). The r parameter to `addMeasurements` determines the ratio of S added in each iteration. After termination it returns the last trained performance model (step 9).

The algorithm is highly parameterizable by adapting t_s and the scoring function. For model construction, any regression or other performance modeling technique can be applied. The implementation used in Section IV applies a threefold cross-validation score on C to determine model accuracy and uses variance as score function. Furthermore, we compare and evaluate different machine learning algorithms for the model construction in Section IV-D.

IV. Evaluation

In this section, we evaluate our approach following a four-step method: in the first step, we use an external, third-party data set to determine the robust metric (cf. Section IV-B); in the second step, we use a newly generated data set to validate the chosen metric (cf. Section IV-B). The very same data set is then used to evaluate the quality of the measurement repetition determination in Section IV-C and the performance model construction in Section IV-D. The data sets are described in Section IV-A. The Baloo framework as well as all evaluation scripts are available for repetition as a CodeOcean capsule[3].

A. Validation Data Sets

For validating Baloo we make use of two different data sets. The configuration space they cover is depicted in Table I and comprises eleven dimensions. The first data set is an existing openly available data set [25], comprising 102 configuration points for Apache Cassandra and Couchbase. The Apache

[3]https://doi.org/10.24433/CO.6929232.v2

TABLE I
EVALUATION CONFIGURATION SPACE

Parameter	Seybold et al. [25]	Baloo data set
Infrastructure	public Amazon EC2, private OpenStack	private OpenStack
VM Type	small - t2.medium	tiny - small - large
DBMS	Apache Cassandra	Apache Cassandra
Cluster size	3 - 5 - 7 - 9	3 - 5 - 7 - 9 - 11
Client consistency	any - one - two	one - two - three
Replication factor	3	1 - 2 - 3
Benchmark	YCSB	YCSB
Workload	write-heavy	write-heavy
Records	4,000,000	4,000,000
Record size	5KB	5 KB
Storage backend	SSD, HDD, remote	SSD

TABLE II
COMPARING THE AVERAGE CV FOR EACH ROBUSTNESS METRICS FOR THE EXTERNAL DATA SET (EXT) WITH OUR OWN DATA SET (OWN).

Metric	Throughput CV		Latency CV	
	Ext	Own	Ext	Own
Mean	0.039	0.062	0.164	0.201
Median	0.044	0.063	0.036	**0.051**
95th percentile	**0.028**	**0.039**	0.138	0.174
90th percentile	0.029	0.042	0.079	0.107
80th percentile	0.032	0.047	0.052	0.076
Trimmed(5%) mean	0.039	0.062	0.046	0.076
Trimmed(10%) mean	0.039	0.062	0.038	0.056
Trimmed(30%) mean	0.042	0.062	**0.035**	**0.051**
Winsorized(5%) mean	0.039	0.062	0.057	0.090
Winsorized(10%) mean	0.039	0.062	0.043	0.059
Winsorized(30%) mean	0.041	0.063	**0.035**	0.052
Trimean	0.042	0.062	0.036	0.053
Hodges-Lehmann	0.039	0.063	0.036	0.054

Cassandra configuration points serve as a reference data set for determining the robust metric.

The second data set has been created for this work and is published as OpenData[1]. For this data set, seven configuration dimensions are static for the sake of this evaluation: We select a private OpenStack-based cloud infrastructure as this gives us control over OpenStack-specific configurations such as the overcommitting factor and Virtual Machine placement. We select Apache Cassandra as representative cloud-hosted DBMS due to its widespread adoption[4]. Finally, we use write-heavy workload issued from the Yahoo Cloud Serving Benchmark (YCSB) [26] and four million records of a record size of 5 KB for comparability to other work [16]. We use SSD as storage as it is recommended for most DBMS.

For the remaining four dimensions we use three (VM type, client consistency, replication factor) and five (cluster size) different configuration options. Due to the fact that client consistency and replication factor cannot be chosen fully independently, this yields a total of 90 different configuration points. The data set contains 10 measurements for each of those and for each measurement a time series of performance metrics, system metrics, and additional meta data.

B. Robust Metric Selection

Here, we evaluate the different statistical measures that are robust metric candidates. We start with a publicly available data set [25] and then evaluate the transferability by comparing the results with our data set. Table II compares the average CV score of the different robust metrics. The lower the score, the less variation and hence the better the metric fits our needs.

Although the two data sets are different in regard to the used cloud infrastructure, the VM-sizes, and the number of measurement repetitions, we observe that the performance of the different metrics is very comparable for both throughput and latency. For throughput, the 95th percentile achieves the best score across both data sets. When analyzing latency, we observe that both data sets have one minimum at for the trimmed mean of 30%. The first data set has an additional minimum using the 30%-winsorized metric, while the second data set performs slightly better using the median. However, both the 30%-winsorized and the median are on the third place for the respective other data set.

Summarizing, we can say that the results from the publicly available data set transfer very well to our own data set. Hence, we conclude that the 95%-percentile for throughput, as well as the trimmed mean or the winsorized mean for latency, are viable robust metrics that can be applied for comparing DBMS cloud performance. As DBMS are usually optimized for throughput, we concentrate on throughput for the remainder of this work. Based on our insights, we use the 95th percentile as robust metric.

C. Measurement Repetition Determination

In this section, we evaluate the measurement repetition determination discussed in Section III-C. We do so by comparing estimations obtained by Baloo with the median overall ten measurement repetitions from the evaluation data set.

As the true mean of the underlying distribution is unknown, we make the assumption that the median of all 10 measurements approximates the true mean of the underlying distribution and serves as a gold standard in this evaluation. We compare this gold standard against randomly selected 1, 2, 3, and 5 points as well as against the result obtained from Baloo.

As Baloo uses probabilistic elements and the selection of the next measurement point is non-deterministic as well and strongly influences the obtained results, we repeat the evaluation 100 times. Table III reports the average results.

Table III shows the mean absolute percentage error (MAPE) and the root mean squared error (RMSE) of Baloo and the five baselines together with the average measurement repetitions required. We observe that the error generally decreases as we increase the number of measurement points. This is expected as more measurement points reduce the impact of random measurement noise. Baloo outperforms 1-point, 2-point, and even 3-point in average MAPE and RMSE, while requiring only 2.59 measurement points on average. This shows that

[4]https://db-engines.com/en/ranking

TABLE III
ACCURACY AND MEASUREMENT REPETITIONS FOR DIFFERENT APPROACHES.

Approach	MAPE (%)	RMSE	Average # points
Baloo	1.42	251.8	2.59
1-point	2.75	538.4	1.00
2-point	2.19	403.3	2.00
3-point	1.54	315.4	3.00
5-point	0.76	148.1	5.00
10-point	0.00	0.0	10.00

in most of the cases, two measurements are sufficient to accurately describe a performance measurement.

In some cases, more measurements are required, though: This insight can be supported by comparing the reported MAPEs with the RMSEs. For MAPE our approach is only slightly better than 3-point ($\sim 8\%$ decrease); the difference is larger when analyzing RMSE ($\sim 20\%$ decrease). Since the RMSE puts a stronger focus on outliers, we can conclude that our approach is able to correctly determine "critical" measurement points, while keeping the requested measurement to the minimum amount of two measurements, when not required. The two baselines 5-point and 10-point consistently achieve lower errors than our proposed approach; this is expected, as they also conduct significantly more measurements. If a higher accuracy is required, our approach could be tuned accordingly. Summarizing, our approach successfully handles the trade-off between required measurement repetitions and target accuracy. It is worth noting that in this experiment, the average number of required measurement points is comparatively low, as all experiments were executed on a private cloud with relatively low load (see Section IV-A). Therefore, the number of required measurement repetitions as well as the gain achieved by our approach might be even higher for other environments [27].

D. Performance Model Construction

In this section, we evaluate the performance model construction techniques and the corresponding performance models. As our approach works with basically any regression technique, our analysis compares the performance of different machine learning algorithms. For this, we analyze the required amount of measurement configuration points together with the achieved target accuracy on the remaining validation set for the following algorithms: linear regression (LinReg), ridge regression (Ridge), elastic net regression (ElasticNet), bayesian ridge regression (BayesianRidge), and huber regression (HuberRegressor) as linear models, a gradient tree boosting regressor (GBDT), a random forest regressor (RandomForest), and a support vector machine (SVR). Additionally, to get a better comparison, we add a baseline regressor (ZeroR) which always predicts the mean of all seen samples.

In this experiment we vary the target accuracy threshold t_s of the internal MAPE score s based on a threefold cross-validation between $0.1, 0.15, 0.2$,0.25, and 0.3. The maximum ratio r_{max} was set to 0.9, resulting in a cutoff at 81 (0.9·90 total) configuration points. As initial configuration measurement set, we set a ratio of 0.05, resulting in at least 5 ($\lceil 0.05 \cdot 90 \rceil$) configurations for each approach. We only add one configuration per increment (i.e., r =0.01), as for our comparatively small example set, the training time was significantly lower than the measurement time and could therefore be neglected. Furthermore, this enables a better analysis of the required configuration points. All algorithms implementations are based on scikit-learn [19] and use the defined default parameterization.

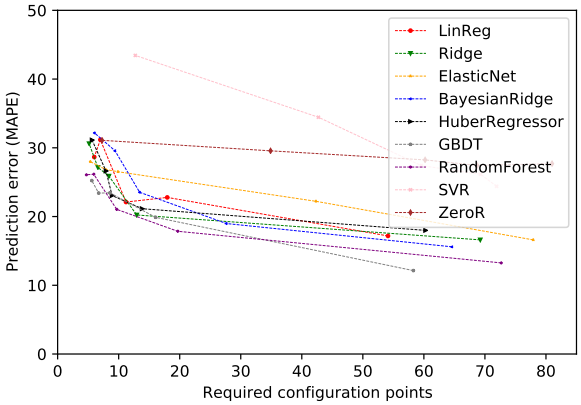

Fig. 2. Achieved accuracy versus required configuration points for varying target accuracies.

The comparison is shown in Figure 2. The x-axis depicts the number of configuration points that were measured, before the specific workflow terminated, i.e., t_s was achieved. The y-axis shows the MAPE on those configurations that were not added to the training set. We repeat all experiments 25 times and report the average value as these processes are highly influenced by the random seed.

From Figure 2, we observe that multiple approaches perform comparably in terms of required configuration points and the achieved prediction error. However, SVR and ZeroR (baseline) are not able to capture the performance structure very well and, therefore, perform poorly in terms of prediction error and required configuration points. The baseline even runs out of measurement options and therefore retrieves the maximum number of measurements for t_s =0.1.

All other approaches perform similarly well, but we can still identify small differences between them. Increasing the target accuracy has the expected effect of generally reducing the prediction error while increasing the measured configuration points. While GBDT achieves the overall lowest prediction error (11.63%), random forest is able to achieve slightly worse results using considerably fewer configuration points for t_s =0.15 and t_s =0.2. For bigger values of t_s, GBDT performs slightly better again.

Table IV shows more details on the performance of the individual algorithms for t_s =0.15. In addition to the average achieved error (MAPE) and the required configurations (#

TABLE IV
DETAILED PERFORMANCE OF ALL ALGORITHMS FOR FIXED TARGET ACCURACY t_s OF 0.15.

Approach	MAPE	# Meas.	# Conf.	Time (s)
LinReg	21.87	44.72	17.08	0.44
Ridge	23.02	38.52	14.80	0.38
ElasticNet	23.28	88.48	34.84	0.93
BayesianRidge	20.08	65.12	25.84	0.77
HuberRegressor	19.50	51.16	20.56	0.96
GBDT	19.93	47.28	17.44	1.02
RandomForest	17.13	65.04	25.60	4.95
SVR	27.03	175.52	68.84	1.90
ZeroR	29.24	200.36	77.92	2.03

Conf.), we see the average number of total measurement runs conducted (# Meas.) and the average time of the workflow execution excluding measurements (Time).

We conclude that even un-optimized machine learning approaches are able to achieve prediction errors of around 20% on a data set consisting of 90 configuration points, made up of 900 individual measurement series, while measuring just one fifth (<20) configurations and conducting even less than 5% (<45) of individual measurement runs.

Adding more measurement points increases the accuracy and reduces the error to up to 12%. However, the accuracy gain per added configuration decreases over time, which is consistent with our expectations. The additional computation effort introduced by our framework is negligible (run-times of less than 1s per training process), if we consider that one measurement run takes minutes or even hours to complete. Therefore, the achieved time savings are more than 95%, for an accuracy cost of just 20%. Even by reducing the accuracy cost to 12%, we can still achieve measurement time reductions of over 80%. We believe this result to be very impressive, considering that average performance in public clouds fluctuates by 5% [28].

V. RELATED WORK

Related approaches can be divided into the following three research areas.

A. Performance measurement of distributed DBMS

DBMS benchmarking is a common process to determine the optimal operational model for DBMS in general and as well for cloud-hosted DBMS. This process is supported by a multitude of benchmarks that support diverse workload models and evaluation objectives [29], [30]. In consequence, there are numerous supportive performance studies of cloud-hosted DBMS available [26], [27], [31]. Yet, each of these studies covers only a small part of the entire cloud resource and distributed DBMS scope.

B. Performance optimization of DBMS

DBMS performance optimization approaches such as ITuned [5], DBSherlock [6] and OtterTune [7] target single instance relational DBMS that are operated on dedicated resources. These approaches have a special focus on the workload by considering trace-based workloads [5], [6] or unknown workload types [7] Rafiki [11] targets the performance optimization of single instance NoSQL DBMS for different workload types by automatically determining DBMS runtime parameters and deriving their optimal configuration. Performance models for distributed DBMS are presented by Farias et al. [32] and Dipietro et al. [33], considering the performance prediction impact of different cluster sizes [32] and DBMS-specific runtime parameters [33] The URSA framework [12] targets the automated capacity planning of a single node DBMS operated on cloud resources. Thus, the focus of URSA lies on the resources, while the aspects of distributed DBMS are not considered. In summary, existing approaches provide comprehensive performance prediction mechanisms for single node DBMS on dedicated resources [5]–[7], [11], focus on distribution aspects [32], [33] without considering cloud resources or consider only cloud resources without considering DBMS distribution aspects [12].

C. Performance prediction of configurable systems

Zhang et al. propose the application of Fourier learning to predict the performance of configurable systems with theoretical accuracy guarantees. [1]. Siegmund et al. combine machine learning and sampling heuristics to build performance-influence models for highly configurable systems. [2]. Sarkar et al. compare different sampling techniques for CART-based performance models and introduce a novel heuristic for the selection of the initial samples [34]. Guo et al. improve CART-based performance model by resampling the training data to determine the accuracy of the resulting model and automated hyper-parameter tuning [35]. Ha et al. propose a deep sparse neural network architecture and hyper-parameter optimization approach for the performance prediction of configurable systems [3]. Westermann et al. [36] compare the accuracy of MARS, CART, Genetic programming, and Kriging for the construction of software performance models. Similarly, Noorshams et al. [37] evaluate the accuracy of linear regression, MARS, CART, M5 Trees and Cubist Forests for the performance modeling of storage systems. To the best of our knowledge, no existing performance prediction approach targets distributed DBMS or addresses measurement variability in the sample generation.

VI. CONCLUSION

In this work, we presented Baloo, a framework for measuring, modeling, and predicting the performance of distributed database management systems (DBMS) in cloud environments for different configurations. Our approach builds upon the Mowgli framework and works by (1) measuring a performance configuration, (2) determining the number of measurement repetitions, (3) determining the next configuration point to be measured, and (4) building a performance model using all available measurement points to predict the remaining unavailable measurement points of the configuration space.

To evaluate our framework, we measured the distributed DBMS Apache Cassandra in our private cloud using 90 dif-

ferent configurations and ten repetitions each, resulting in 900 measurement runs comprising of roughly 450 measurement hours and 9,450 compute hours. We also made this data set publicly available to foster future research towards this area. The evaluation shows that our highly configurable approach is able to save between 80% and 95% of measurement time for a respective accuracy cost of 12% to 20%.

In future work, we plan to include other DBMS and cloud providers (including measurements from public clouds) in our configuration model itself. This would enable to not only to guide with the desired configuration parameters, but also the best cloud environment or distributed DBMS system to select. Furthermore, this could include the actual discovery of which available configuration options are performance-relevant.

REFERENCES

[1] Y. Zhang, J. Guo, E. Blais, and K. Czarnecki, "Performance prediction of configurable software systems by fourier learning (t)," in *2015 30th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, 2015, pp. 365–373.

[2] N. Siegmund, A. Grebhahn, S. Apel, and C. Kästner, "Performance-influence models for highly configurable systems," in *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering*, 2015, pp. 284–294.

[3] H. Ha and H. Zhang, "Deepperf: performance prediction for configurable software with deep sparse neural network," in *IEEE/ACM 41st International Conference on Software Engineering*, 2019, pp. 1095–1106.

[4] 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*, 2012, p. 190–199.

[5] S. Duan, V. Thummala, and S. Babu, "Tuning database configuration parameters with ituned," *Proc. VLDB Endow.*, p. 1246–1257, Aug. 2009.

[6] D. Y. Yoon, N. Niu, and B. Mozafari, "Dbsherlock: A performance diagnostic tool for transactional databases," in *Proceedings of the 2016 International Conference on Management of Data*, 2016, p. 1599–1614.

[7] D. Van Aken, A. Pavlo, G. J. Gordon, and B. Zhang, "Automatic database management system tuning through large-scale machine learning," in *Proceedings of the 2017 ACM International Conference on Management of Data*, 2017, p. 1009–1024.

[8] A. Davoudian, L. Chen, and M. Liu, "A survey on nosql stores," *ACM Comput. Surv.*, vol. 51, no. 2, Apr. 2018.

[9] S. Mazumdar, D. Seybold, K. Kritikos, and Y. Verginadis, "A survey on data storage and placement methodologies for cloud-big data ecosystem," *Journal of Big Data*, vol. 6, no. 1, p. 15, Feb 2019.

[10] D. Abadi, A. Ailamaki, D. Andersen, P. Bailis, M. Balazinska, P. Bernstein, P. Boncz, S. Chaudhuri, and et al., "The seattle report on database research," *SIGMOD Rec.*, vol. 48, no. 4, p. 44–53, Feb. 2020.

[11] A. Mahgoub, P. Wood, S. Ganesh, S. Mitra, W. Gerlach, T. Harrison, F. Meyer, A. Grama, S. Bagchi, and S. Chaterji, "Rafiki: a middleware for parameter tuning of nosql datastores for dynamic metagenomics workloads," in *Proceedings of the 18th ACM/IFIP/USENIX Middleware Conference*, 2017, pp. 28–40.

[12] N. Zheng, Q. Chen, Y. Yang, J. Li, W. Zheng, and M. Guo, "Poster: Precise capacity planning for database public clouds," in *2019 28th International Conference on Parallel Architectures and Compilation Techniques (PACT)*, 2019, pp. 457–458.

[13] W. Xiong, K. Yang, and H. Dai, "Improving nosql's performance metrics via machine learning," in *2019 Seventh International Conference on Advanced Cloud and Big Data (CBD)*, 2019, pp. 90–95.

[14] Y. Zhu and J. Liu, "Classytune: A performance auto-tuner for systems in the cloud," *IEEE Transactions on Cloud Computing*, 2019.

[15] A. Iosup, N. Yigitbasi, and D. Epema, "On the performance variability of production cloud services," in *2011 11th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing*, 2011, pp. 104–113.

[16] D. Seybold, M. Keppler, D. Gründler, and J. Domaschka, "Mowgli: Finding your way in the dbms jungle," in *Proceedings of the 2019 ACM/SPEC International Conference on Performance Engineering*, 2019, pp. 321–332.

[17] Y. S. Dodonov and Y. A. Dodonova, "Robust measures of central tendency: weighting as a possible alternative to trimming in response-time data analysis," *Psikhologicheskie Issledovaniya*, vol. 5, no. 19, pp. 1–11, 2011.

[18] F. T. Liu, K. M. Ting, and Z. Zhou, "Isolation forest," in *2008 Eighth IEEE International Conference on Data Mining*, 2008, pp. 413–422.

[19] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[20] J. A. Pereira, H. Martin, M. Acher, J.-M. Jézéquel, G. Botterweck, and A. Ventresque, "Learning software configuration spaces: A systematic literature review," 2019.

[21] J. Alves Pereira, M. Acher, H. Martin, and J.-M. Jézéquel, "Sampling Effect on Performance Prediction of Configurable Systems: A Case Study," in *International Conference on Performance Engineering*, 2020.

[22] C. Kaltenecker, A. Grebhahn, N. Siegmund, and S. Apel, "The interplay of sampling and machine learning for software performance prediction," *IEEE Software*, 2020.

[23] J. Oh, D. Batory, M. Myers, and N. Siegmund, "Finding near-optimal configurations in product lines by random sampling," in *Proc. of the 11th Meeting on Foundations of Software Engineering*, 2017, p. 61–71.

[24] S. Chakraborty, K. S. Meel, and M. Y. Vardi, "A scalable and nearly uniform generator of sat witnesses," in *Computer Aided Verification*. Springer, 2013, pp. 608–623.

[25] D. Seybold and J. Domaschka, "Mowgli: DBMS Performance & Scalability Evaluation Data Sets," Oct. 2019. [Online]. Available: https://doi.org/10.5281/zenodo.3518786

[26] B. F. Cooper, A. Silberstein, E. Tam, R. Ramakrishnan, and R. Sears, "Benchmarking cloud serving systems with ycsb," in *Proceedings of the 1st ACM Symposium on Cloud Computing*, 2010, p. 143–154.

[27] J. Kuhlenkamp, M. Klems, and O. Röss, "Benchmarking scalability and elasticity of distributed database systems," *Proc. VLDB Endow.*, vol. 7, no. 12, p. 1219–1230, Aug. 2014.

[28] J. Scheuner and P. Leitner, "Estimating cloud application performance based on micro-benchmark profiling," in *2018 IEEE 11th International Conference on Cloud Computing (CLOUD)*. IEEE, 2018, pp. 90–97.

[29] D. Seybold and J. Domaschka, "Is distributed database evaluation cloud-ready?" in *European Conference on Advances in Databases and Information Systems (ADBIS)*, 2017, pp. 100–108.

[30] V. Reniers, D. Van Landuyt, A. Rafique, and W. Joosen, "On the state of nosql benchmarks," in *Proceedings of the 8th ACM/SPEC on International Conference on Performance Engineering Companion*, 2017, p. 107–112.

[31] A. Hendawi, J. Gupta, L. Jiayi, A. Teredesai, R. Naveen, S. Mohak, and M. Ali, "Distributed data stores: Performance analysis and a case study," in *IEEE International Conference on Big Data*, 2018, pp. 1937–1944.

[32] V. A. Farias, F. R. Sousa, J. G. R. Maia, J. P. P. Gomes, and J. C. Machado, "Regression based performance modeling and provisioning for nosql cloud databases," *Future Generation Computer Systems*, vol. 79, pp. 72 – 81, 2018.

[33] S. Dipietro, G. Casale, and G. Serazzi, "A queueing network model for performance prediction of apache cassandra," in *Proceedings of the 10th EAI International Conference on Performance Evaluation Methodologies and Tools on 10th EAI International Conference on Performance Evaluation Methodologies and Tools*, 2017, p. 186–193.

[34] A. Sarkar, J. Guo, N. Siegmund, S. Apel, and K. Czarnecki, "Cost-efficient sampling for performance prediction of configurable systems (t)," in *2015 30th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, 2015, pp. 342–352.

[35] J. Guo, D. Yang, N. Siegmund, S. Apel, A. Sarkar, P. Valov, K. Czarnecki, A. Wasowski, and H. Yu, "Data-efficient performance learning for configurable systems," *Empirical Software Engineering*, vol. 23, no. 3, pp. 1826–1867, 2018.

[36] 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*, 2012, pp. 190–199.

[37] 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 International Conference on Performance Engineering*, 2013, pp. 283–294.