2016 16th IEEE/ACM International Symposium on Cluster, Cloud, and Grid Computing

Machine Learning Approach for Cloud NoSQL Databases Performance Modeling

Victor A. E. Farias, Flávio R. C. Sousa, José G. R. Maia, João P. P. Gomes, Javam C. Machado
Computer Science Department
Federal University of Ceara
Fortaleza, Brazil
{victorfarias, flavio, gilvan, jpaulo, javam}@lia.ufc.br

***Abstract*—Cloud computing is a successful, emerging paradigm that supports on-demand services with pay-as-you-go model. With the exponential growth of data, NoSQL databases have been used to manage data in the cloud. In these newly emerging settings, mechanisms to guarantee Quality of Service heavily relies on performance predictability, i.e., the ability to estimate the impact of concurrent query execution on the performance of individual queries in a continuously evolving workload. This paper presents a performance modeling approach for NoSQL databases in terms of performance metrics which is capable of capturing the non-linear effects caused by concurrency and distribution aspects. Experimental results confirm that our performance modeling can accurately predict mean response time measurements under a wide range of workload configurations.**

***Keywords*-Cloud Computing, Performance Modeling, NoSQL**

I. Introduction

Cloud computing is a paradigm of remarkable success for service-oriented computing. Scalability, elasticity, pay-per-use pricing, and economy of scale are the major reasons for this success. Since most of cloud applications are data-driven, database management systems powering these applications are critical components in the cloud software stack [1].

Cloud NoSQL database systems are leading to data processing environments that concurrently execute heterogeneous query workloads. At the same time, these systems need to satisfy diverse performance expectations defined in service level agreements (SLAs). In these newly emerging settings, avoiding potential SLA violations heavily rely on performance predictability, i.e., the ability to estimate the impact of concurrent query execution on the performance of individual queries in a continuously evolving workload [2]. Moreover, the performance of a distributed system may not be linearly correlated with the amount of resources allocated. It has been shown that concurrency and distribution aspects can degrade the performance in a non-linear fashion, which is a challenge for performance modeling approaches [3].

Modeling the performance impact of complex interactions that arise when multiple queries share computing resources and data is difficult albeit critical for a number of tasks such as quality of service (QoS) manning cloud-based database platforms, effective resource allocation for time-sensitive processing tasks, and user experience management for interactive database systems. Performance prediciton has been addressed in the context of SQL DBMSs for isolated queries [4] and for concurrent workloads [2], [5], [6]. However, this problem has received little attention in the context of cloud NoSQL systems. Moreover, some solutions proposed for NoSQL databases are specific to a single NoSQL system [7].

To overcome these limitations, this paper presents a machine learning approach for performance modeling of NoSQL databases. The major contributions of this paper are as follows:

- We present a performance modeling approach for NoSQL database systems exploring machine learning techniques for regression in order to predict performance metrics considering concurrency and distribution aspects;
- We present a full prototype implementation of the proposed approach;
- We evaluate models constructed by our prototype to assess their predictive accuracy.

Organization: This paper is organized as follows. Section II explains our approach. Section III describes the experimental evaluation. Conclusions and future research steps are shown in Section IV.

II. Our Approach

Our goal is to predict the performance of NoSQL database systems considering the configuration of the database cluster and the incoming workload. The performance is quantified by means of performance metrics. The metric addressed in our work is intuitive from a stakeholder's point of view and reflects user's experience: mean response time per second.

The workload is defined as a sequence of requests that are issued to the system per second. This work considers a heterogeneous workload where it may contain distinct classes of requests as simple queries, range queries, writes and updates. These requests have different access patterns and their execution times can be different by orders of magnitude.

Several factors can affect the performance of a distributed NoSQL database system. We address three aspects that most contributed to the performance variation in our experiments:

- **Requests complexity**. Each single workload has a different access pattern and requires a different amount of disk operations.
- **Concurrency**. Concurrent execution of two distinct workload classes may generate interference caused by lock resource sharing [5].

- **Distribution**. Most NoSQL databases distribution strategies employ replication to improve availability and reliability. However, this implies on an overhead to maintain consistency. Some consistency strategies use distributed locks leading to waits and deadlocks, and, consequently, degrading the performance in a non-linear fashion [3].

Performance modeling is commonly addressed with analytical approaches where it analyses each single factor that impacts the performance on a specific database system. These approaches can become very complex for when considering the aforementioned characteristics and also they are not generic since they are system-specific. Hence, our approach relies on machine learning algorithms and we show, by experimentation, that our approach can capture the aforementioned aspects with an acceptable error. In particular, since the performance metrics are continuous and non-finite, we address the performance modeling problem as a regression problem. Regression methods are capable of producing predictive models fit for estimating a given phenomenon (system's performance) given a set of observations (training data set) under several scenarios (cluster and workload configurations).

Our approach is composed by two main steps:

1) Performance Data Set Generation. This step aims to produce the base performance training data set by monitoring performance in an extensive experimentation and to transform the data to achieve better accurate predictive models.

2) Feature Transformation. In general, the behavior of the performance is not linearly dependent on the features extracted from the data set. A suitable manipulation of the data set features can improve the quality of the predictive models. Adding non-linear features aids the learning methods to capture these effects.

A. Performance Data Set Generation

The Data Set Generation is divided into two sub steps: (1) Performance Measuring and (2) Filtering and Aggregating.

1) Performance Measuring: In order to understand the impact of the system configuration, we measure the performance metric under a wide range of system and workload configurations. We do so with offline experiments, i.e., we use a test environment to carry out various experiments.

The system's configurations are composed by parameters that are divided into two kinds: workload and cluster parameters. In our experiments we use YCSB benchmark [8]. This benchmark offers several workload parameters, but, in our experiments, we selected the parameters that most contribute to the performance behavior and that present similar dynamics to real life workloads. For the workload parameters we use: *target* parameter, an integer representing the target number of requests per second that YCSB issues; and *mix* parameter, a vector $[p_{read},\ p_{scan},\ p_{update},\ p_{insert}]$ where each element represents the percentage of queries falling into that category. For example, p_{read} represents the total percentage of issued queries that correspond to read requests.

The cluster parameters should be generic to any kind of NoSQL database system and have great effect on performance. NoSQL systems are designed to run in a distributed manner by taking advantage of a large resource pool. The data partitioning and management are database-specific. Thereby, we use the number of working nodes DB as a database system parameter because it happens to be the only generic parameter and it represents the capacity of the system.

Therefore, it is necessary to define the set of system parameters which are employed for experimentation. It is worth noting that the range of each parameter impacts in the predictive power of the resulting models. For example, if we test values of $target$ parameter on the range $(4000, 6000)$, it is expected that predictions for $target$ values greater than 6000 and less than 4000 are less accurate. Table I show the test values for each parameter.

Kind	**Parameter**	**Values**
Workload	p_{read}	0%,20%,40%,60%,80%,100%
	p_{scan}	0%,20%,40%,60%,80%,100%
	p_{insert}	0%,20%
	p_{update}	0%,20%
	target	1000,2000,3000,4000,5000, 6000,7000,8000,9000,10000
Cluster	DB	2,3,4

TABLE I: System's parameters testing values.

Our approach executes automatically one experiment with duration 360 for each parameter combination presented in Table I. Note that not all combinations are possible since $p_{read} + p_{scan} + p_{update} + p_{insert} = 1$. Thus, there exists 20 combinations of $mix = [p_{read}, p_{scan}, p_{update}, p_{insert}]$. Besides that, there are 10 possibilities for $target$ parameter and 3 possibilities for DB parameter. Therewith, we execute $20 \times 10 \times 3 = 600$ experiments in total.

In each experiment, we generate a log file where its records contain aggregated performance metrics. For each experiment second, the mean response time of all successfully executed by the system in this second is recorded to the log.

2) Filtering and Aggregation: The raw experiment logs are not useful for predictive analysis since all logs compound a reasonably big data volume and the logs are not in the input format for machine learning algorithms. A machine learning algorithm input training data set is composed by a feature matrix and a target vector. In our context, each experiment is represented as feature matrix line (or feature vector) has the following features: $< DB$, t , t_{read} , t_{scan} , t_{update} ,$t_{insert} >$ where DB is the number of working nodes, t is the $target$ parameter and $target_{type}$ is the target number of type request, e.g., $target_{read}$ is $target \times read$ on this experiment.

For each feature vector, there is a corresponding target value in target vector that holds the summarized value of the performance metric for the experiment corresponding to that feature vector. The summarized value of the performance metric is obtained by aggregating filtered log information of its corresponding experiment. Initially, the first 60 seconds and the last 60 seconds representing the warmup and cool-down time are removed from the log. After, we remove the metric measurements that are larger than $80th$ percentile and smaller than $20th$ percentile and hence the summarized value is the mean of the remaining measurements. This value captures the standard behavior of the metric and prevents noise from unexpected effects of scheduling and interference from others clients running in the same physical machine.

B. Feature Transformation

The models generated by the machine learning techniques should have generalization and extrapolation power in order to deliver reliable prediction for its users. It is a difficult task since this model should capture the database systems performance particularities as cited in the introduction of Section II.

Specifically, theoretical work [3] suggest that, for master copy replicated schemes, the number of working nodes and the number of issued requests per second impacts quadratically over performance due to distributed deadlock rate. We extend this concept for NoSQL databases since many of these systems employ replication to improve availability and read capacity.

There exist methods capable of modeling highly non-linear relations between the features and the target variable, such as the boosting approaches based on tree regressors [9]. But these methods lack extrapolation power [10]. In this sense, the linear regression models offer a higher extrapolation power. Nevertheless, such methods can find only linear relationships between the features and the target value. Thus, it is required a feature transformation to allow the linear models to capture non-linear relationships and interactions between variables.

Thereby we propose two modelings and we envisage to test both modelings with a variety of regression methods:

1) Linear features. The base data set generated in section II-A is directly given as input to a regression method. The feature DB represents the addition or removal of capacity when adding or removing nodes from the system. The target features ($target$, $target_{read}$, $target_{scan}$, $target_{update}$, $target_{insert}$) account for the contribution of each workload class to the performance, since requests from distinct classes have different levels of complexity.

2) Quadratic features. To model the effects of the concurrency and distribution on the performance metric, we modify the base data set designed in section II-A. In addition to the 6 original features $< DB , t , t_{read} , t_{scan} , t_{update} , t_{insert} >$ on the base data set, we add, to each sample:

- 15 pairwise product features of the target features (t , t_{read} , t_{scan} , t_{update} , t_{insert}) including their squared features. These features express the degradation of performance caused by resource sharing among workload classes.
- 5 products of $nodes$ feature with the target features ($target$, $target_{read}$, $target_{scan}$, $target_{update}$, $target_{insert}$). These features are useful to account the decay of performance by waits and deadlocks incurred by distributed locks for each workload class.

III. Experimental Evaluation

Our performance modeling approach aims to predict the performance of a NoSQL database by constructing a number of machine learning models. Hence, experiments were designed considering the following goals:

- Evaluate the predictive power of the models. This is assessed by means of traditional k-fold cross-validation adopting R^2 scoring metric.
- Determine the tradeoff between the adoption of linear or quadratic features.

A. Experimental setup

We implemented a prototype of our approach using Python and Bash Script languages. The machine learning methods implementation were provided by scikit-learn [11]. The YCSB benchmark 0.5.0 was set with the following configurations: 40 client threads, 1000000 records on database, 1-byte size record, uniform access pattern.

The test database in our experiments is MongoDB [12] set on Master/Slave replication mode. The *ReadPreference* parameter is set to *Nearest* which means that MongoDB driver sends read for a randomly chosen node while writes are always issued to the master. This prototype was deployed and tested in public cloud Amazon EC2. YCSB client was executed in a *m3.xlarge* instance and the MongoDB nodes were deployed in *t2.small* instances.

B. Experimental Results

We tested most of the regression methods contained in scikit-learn library to produce the predictive models along with *grid-search* hyperparameter optimization. We report the results of the two methods that produced the most interesting results: (i) Linear Regression with Regularization (LR) since its a simple and interpretable method that obtained reasonably good R^2 score; and (ii) Gradient Boosting Machine which is a *boosting* approach based on tree that obtained the highest R^2 scores in all cases.

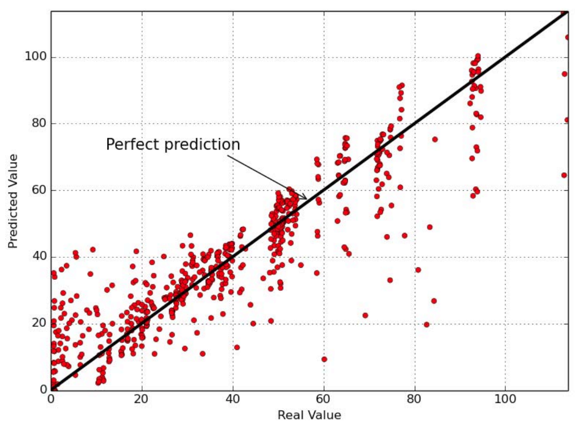
R^2 score may be counter intuitive from the perspective of a DBA. But observe that the maximum value of R^2 score is 1 which indicates perfect prediction and smaller R^2 scores (possibly negatives) indicate less accurate predictions.

Thus we apply our approach in order to predict mean response time (MRT) per second of all requests successfully executed by the system. We also compare the results of our approach when using linear feature (LF) or quadratic features (QF). We report the results in Table II.

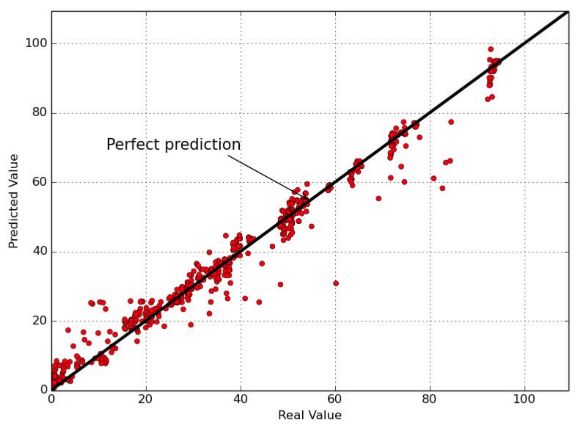
	LR	**GBM**
LF	0.49	0.85
QF	0.74	0.83

TABLE II: MRT: R^2 score for linear and quadratic features.

The LR methods achieved an average R^2 when employing linear features ($R^2 = 0.48$) suggesting a non-linearity relationship of MRT metric and the system's parameters. When using quadratic features LR methods improved its R^2 score to 0.74 which is a improvement of 51% when compared to its linear counterpart.



(a) Quadratic features - Linear model



(b) Quadratic features - Gradient Boosting Machine

Fig. 1: Relation between predicted value and real value

GBM method performed well in both cases, linear and quadratic features, because GBM can capture highly non-linear relationships. Consequently, the new features did not add any useful insight for this method. GBM performed better than LR with quadratic features because it captured other aspects of performance that LR with quadratic features could not consider when transforming the features on Section II-B.

We also show graphical means to measure the models accuracy. A scatter plot 1 presents the real target value for each sample in the training data set against the predicted value using the models generated by two methods: Linear Regression and Gradient Boosting Machine using quadratic features. Points near the line $y = x$ indicates perfect prediction and points far from this line mean inaccurate prediction. Thus, GBM model (Figure 1a) presents points closer to $y = x$ line than LR (Figure 1b) indicating a more accurate prediction. Furthermore, we can observe points far the the $y = x$ line on the right-lower quadrant in which both models have inaccurate predictions. This suggests that these points are outliers that prevent both models to achieve perfect prediction.

IV. Conclusion and Future Work

This paper presents a novel approach for performance modeling for NoSQL databases. This approach is able to construct predictive models that predicts performance metrics by capturing non-linear effects caused the aspects of distribution and concurrency model. We carried out a number of experiments in order to evaluate this approach where the target performance metric is mean response time per second of the request successfully executed by the system. Experimental results confirm that Gradient Boosting Machine accurately predicts performance regardless the feature transformation but it lacks extrapolation power while Linear Models achieved a reasonably good prediction with quadratic features. There is a number of research opportunities that derive from this work, including: a deeper investigation on outliers handling, test this approach for other performance metrics, upgrade to a online performance modeling of NoSQL databases and test scenarios consisting of heterogeneous resources and hybrid clouds. Further investigation on a resource provisioning based on performance models is now ongoing.

Acknowledgment

This Research was partially supported by LSBD/UFC and Capes/Brazil.

References

[1] A. J. Elmore, S. Das, D. Agrawal, and A. El Abbadi, “Zephyr: live migration in shared nothing databases for elastic cloud platforms,” in *SIGMOD ’11*, 2011, pp. 301–312.

[2] J. Duggan, U. Cetintemel, O. Papaemmanouil, and E. Upfal, “Performance prediction for concurrent database workloads,” in *ACM SIGMOD*, ser. SIGMOD ’11. ACM, 2011, pp. 337–348.

[3] J. Gray, P. Helland, P. O’Neil, and D. Shasha, “The dangers of replication and a solution,” in *ACM SIGMOD Record*, vol. 25, no. 2. ACM, 1996, pp. 173–182.

[4] A. Ganapathi, H. Kuno, U. Dayal, J. L. Wiener, A. Fox, M. Jordan, and D. Patterson, “Predicting multiple metrics for queries: Better decisions enabled by machine learning,” in *ICDE*. IEEE, 2009, pp. 592–603.

[5] B. Mozafari, C. Curino, A. Jindal, and S. Madden, “Performance and resource modeling in highly-concurrent oltp workloads,” in *ACM SIGMOD*. ACM, 2013, pp. 301–312.

[6] V. A. E. Farias, F. R. C. Sousa, J. G. R. Maia, J. a. P. P. Gomes, and J. C. Machado, “Elastic provisioning for cloud databases with uncertainty management,” in *ACM SAC*. (accepted for publication), 2016.

[7] D. Didona and P. Romano, “On bootstrapping machine learning performance predictors via analytical models,” *arXiv preprint arXiv:1410.5102*, 2014.

[8] 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*. ACM, 2010, pp. 143–154.

[9] J. H. Friedman, “Greedy function approximation: a gradient boosting machine,” *Annals of statistics*, pp. 1189–1232, 2001.

[10] W.-Y. Loh, C.-W. Chen, and W. Zheng, “Extrapolation errors in linear model trees,” *ACM Transactions on Knowledge Discovery from Data (TKDD)*, vol. 1, no. 2, p. 6, 2007.

[11] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, and et al., “Scikit-learn: Machine learning in Python,” *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[12] M. Inc, “*MongoDB*,” 2015, http://www.mongodb.com.