2017 IEEE 25th International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems

A measurement-based approach to performance prediction in NoSQL systems

Flora Karniavoura*† and Kostas Magoutis*‡
*Institute of Computer Science, Foundation for Research and Technology – Hellas, Heraklion 70013, Greece
†Computer Science Department, University of Crete, Heraklion 70013, Greece
‡Department of Computer Science and Engineering, University of Ioannina, Ioannina 45110, Greece

***Abstract*—In this paper we evaluate a measurement-based approach to performance prediction of data-intensive applications over NoSQL systems. While the use of systematic measurements for building performance prediction models is a well studied topic, little attention has been paid so far on the application space of data-intensive systems using NoSQL databases. Measurement-based performance prediction approaches are often limited by a relatively narrow range of hardware characteristics available within each organization's private infrastructure. An opportunity to change this fact is the emergence of federated, publicly-accessible, large-scale research infrastructures, such as Fed4FIRE and GENI, featuring a variety of heterogeneous hardware. This paper demonstrates accurate measurement-based performance prediction modeling for NoSQL systems over such public infrastructures. We consider three prominent regression techniques: Multivariate adaptive regression splines (MARS), support vector regression (SVR), and artificial neural network (ANN) regression, applied to the YCSB data-intensive benchmark over the MongoDB NoSQL data store. Our measurements are drawn from 1-, 3-, and 5-node clusters of four node types. Performance prediction using MARS yields the best results with an average accuracy of 97.85% vs. 94.16% and 90.39% with SVR and ANN respectively. The approach can seamlessly extend to a wider range of hardware specifications available in federated research infrastructures.**

I. Introduction

The rapid growth of Internet-scale applications in the early 2000s created the need for distributed data stores offering superior scalability and elasticity compared to what is possible with traditional SQL servers, giving rise to a new class of data stores referred to as NoSQL systems [1], [2], [3]. A recent trend in this space is controlled quality of service, where service providers offer performance guarantees, such as a certain level of throughput, while keeping response time below a threshold. An important enabler of this capability is accurate prediction of the amount and type of resources needed to sustain a desired level of service. While being an active area of research in the recent past [4], [5], [6], [7], [8], [9], performance prediction within the domain of data-intensive applications over NoSQL data stores has only recently been addressed [6], [10], [11], [12], [13], [14].

This paper aims to fill this gap by proposing a measurement-based performance prediction approach specifically designed for NoSQL data stores, implementing and evaluating it on a widely-deployed data store, MongoDB. Given the non-linear characteristics of I/O-intensive system performance, we employ and compare prediction with three prominent, competitive regression techniques that can handle non-linearity: Multivariate adaptive regression splines (MARS), support vector regression (SVR), and artificial neural networks (ANN). Our extensive evaluation highlights performance characteristics of MongoDB under Yahoo! Cloud Service Benchmark (YCSB) workloads and analyzes prediction accuracy under the three regression techniques for specific prediction needs.

Overall our contributions in this paper are:

- A measurement-based performance prediction approach geared towards the characteristics of NoSQL data stores, and comparison of the prediction accuracy of three regression techniques (MARS, SVR, ANN) in this task.
- Implementation of the approach over a federated, publicly accessible research infrastructure.
- An experimental evaluation based on representative synthetic workloads produced by YCSB over MongoDB.
- An analysis of the prediction accuracy of our approach when varying parameters such as load, node types and cluster size, demonstrating average accuracy of 97.85% for MARS, 94.16% for SVR and 90.39% for ANN.

The remainder of this paper is structured as follows: Section II provides background on NoSQL data stores and MongoDB in particular, previous approaches to performance modeling and prediction, and a brief introduction to the regression techniques used in this work. Section III outlines our methodology and experimental setup. Section IV describes the evaluation of the prediction accuracy of our approach. Finally in Section V we summarize our conclusions.

II. Background

NoSQL data stores [3] implement a data abstraction that resembles traditional database tables offering semantics that lie between those offered by traditional databases and parallel file systems. Early NoSQL systems [1], [3] used a data schema with a single primary key and a single column storing an opaque value. In this work we use MongoDB, a *document-oriented* data store [2], [15], in which stored values are structured binary-JSON (BSON) documents. MongoDB supports horizontal data partitioning across nodes (*sharding*) and uses B-Trees to index data within shards. Each shard may be replicated for high availability and failure recovery. *WiredTiger*, the default storage engine as of MongoDB version 3.2, features a journal (a transaction log) to record data modifications.

2375-0227/17 $31.00 © 2017 IEEE
DOI 10.1109/MASCOTS.2017.31

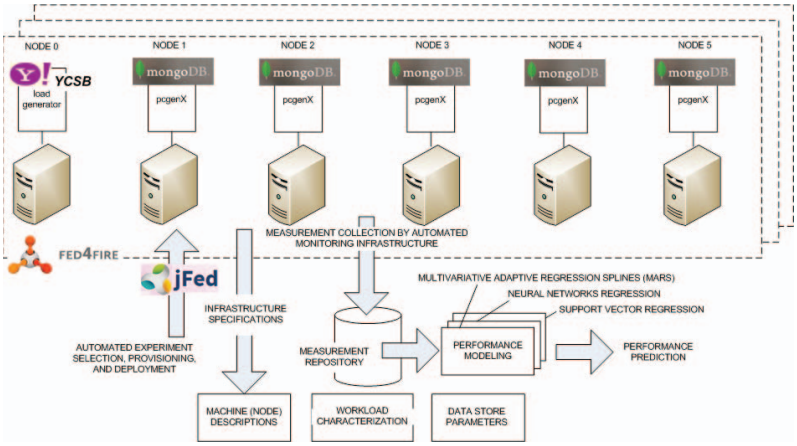

Fig. 1. Experimental testbed and methodology

A. Performance modeling and prediction

Performance evaluation techniques can be grouped into two major categories: *model-based* and *measurement-based*. Model-based approaches require in-depth understanding of the application [16]. Measurement-based techniques consider the application as a *black box* and thus are more straightforward and prevalent in practice [17]. They require methodological experimentation to collect a wide range of measurements on the system under study.

Westermann *et al.* [18] introduced Performance Cockpit, a system that provides adapters to define and automatically run experiments on a system, storing and analyzing measurements of its performance. Performance Cockpit can be used to conduct systematic measurements for performance evaluation and prediction [9], [19], [20] and help developers understand the impact of design and implementation decisions on performance [21]. Although automated, exploration of a large parameter space can take a significant amount of time. The task of efficiently navigating the parameter space is assigned to a *performance analyst*, a human expert. Workload parameter values relevant to real-world use-cases are captured by benchmarks such as SPEC [22] and YCSB [23].

Our approach shares certain aspects with those described above: We follow a similar workflow recognizing performance-related parameters, automating tests, and using the collected measurements to derive performance knowledge. We also utilize MARS regression [24] for predicting performance, a choice that has worked well in previous studies [25], [26], [27]. For example, in [27] different statistical inference techniques are evaluated, including MARS and CART [28] (also considered in Guo *et al.* [7]) with MARS achieving the best results. In this paper we extend the comparative evaluation of MARS in predicting system performance to two other powerful regression methods: Support Vector Regression and Artificial Neural Networks.

Noorshams *et al.* [26] focus on performance modeling for virtualized storage systems, predicting specific application metrics under different I/O workloads and access patterns. Koziolek *et al.* [29] surveyed performance-prediction techniques for component-based applications. Chen *et al.* [5] constructed a performance prediction model for component-based applications by combining profiling and benchmarking techniques. The Palladio Component Model (PCM) [4], [30] offers a domain-specific modeling language for component-based software architectures, specifing performance-relevant information. PCM is used to predict performance on distributed systems and is extended [31] to model the influence of internal component configurations on performance. In this work, we also include performance-sensitive internal datastore configuration settings in our parameter set.

Recently, Cruz *et al.* [6] studied resource usage prediction of distributed key-value data stores based on hardware-dependent resource usage models. Our work additionally focuses on the prediction of operation metrics (read/write latency).

B. Regression analysis and techniques

Regression analysis is a mathematical tool used to estimate relationships between variables and predict an outcome based on their values. “Training data” is used to build regression models that can predict the outcome of “test data”. The prediction accuracy of a model is evaluated by comparing predicted values to actual measured values. We employ three regression techniques in this work.

Multivariate Adaptive Regression Splines (MARS) [24] is a non-parametric regression technique that makes no assumptions about the underlying form of data and automatically models nonlinearities and interactions between variables. MARS has previously been used for performance prediction with promising results [25], [26], [27]. *Support Vector Machines* [32] fall under the category of supervised-learning models and can be used for classification and regression [33]. Support Vector Regression (SVR) is a data-driven, non-parametric regression method deriving from SVMs. SVR has been used to construct performance prediction models for shared storage infrastructures [34]. *Artificial Neural Networks* (ANN) [35]

are self-learning systems commonly used for regression, previously employed for performance prediction [36], [37], [38]. Other techniques applicable towards performance modeling and prediction include linear and decision-tree regression [36], CART [39], queuing theory analysis [10], [11], [12], [13], [40], and various kinds of interpolation [41].

III. Methodology

A. Experimental setup

In this section we describe our experimental setup and methodology, outlined in Figure 1. We implement three different cluster (1-, 3-, and 5-node) setups for each of four machine types allocated for experiments from the iMinds Virtual Wall infrastructure [42] through jFed [43]. Our workload generator is the Yahoo! Cloud Serving Benchmark (YCSB) [44], a configurable benchmark offering a variety of predefined workloads. We used the latest versions of MongoDB (3.2.9) and YCSB (0.10.0) available at the time of testing.

The first setup is a 1-node installation of the data store on a single machine. This setup, typically practiced by standard users with minimal requirements, uses node 1 in Figure 1. The second setup employs sharding over 3 nodes (nodes 1-3 in Figure 1), each hosting a shard (horizontal partition of the YCSB table). Each shard is replicated three times, with replicas placed on different nodes. Each one of the 3 nodes that comprise the data store cluster runs three instances of the data store server. The first one acts as a primary replica for one shard and the other two as secondary replicas of other shards (whose primary is stored on another machine). The third setup consists of 5 nodes (nodes 1-5 in Figure 1) each hosting a shard, using the same replication method (each machine stores one primary and two secondary replicas). These setups allow us to study the scaling behavior of the NoSQL system. The YCSB server is hosted on a dedicated machine with sufficient resources to produce the targeted load for each experiment.

Our experiments are conducted over the following four types of physical machines (denoted as "pcgenX" in Figure 1).

- A: AMD Opteron 2212 @2.00GHz, 4 cores, 4GB RAM
- B: Intel Xeon E5620 @2.40GHz, 16 cores, 12GB RAM
- C: Intel Xeon E5645 @2.40GHz, 24 cores, 24GB RAM
- D: Intel Xeon E3-1220v3@3.10GHz, 4 cores, 16GB RAM

The SPECint CPU2006 score of the machines are 9.34, 19.6, 21.9 and 34.6 respectively. Each of the 1, 3, and 5-node setups were fitted uniformly with machines of the same type (Figure 1). Each machine runs 64-bit Ubuntu Server 14.04 LTS and has a 16GB local disk.

B. Data collection process

A YCSB workload specification includes parameters such as the read/write operation mix, total dataset size (number of records), request (key) distribution (random, zipfian or latest) and number of requests to be performed [23]. After it populates the database and for each run, YCSB reports the average latency of each operation type, average throughput achieved during the test, and other metrics. A *test type* is defined by the node and workload characteristics as well as the data store configuration used, mapping to a set of model parameters. Empirical evidence using an incremental approach [25], [27] led us to conclude that 20–30 measurements of each test type (automated via scripts) are sufficient to maximize prediction accuracy with each method. Initial runs while the system is "warming up" are omitted from our gathered data.

To efficiently navigate a large parameter space and focus on areas of practical interest, we rely on human expertise and guidance as in previous work [18]. Our choice of workload parameters is driven by the industry-standard YCSB benchmark: Among a number of predefined workload mixes, we select a read-intensive and a write-intensive one. The read-heavy mix features 95% reads, 5% updates ("Workload B"), resembling workloads of applications such as online web stores. The write-heavy workload consists of 50% reads, 50% writes ("Workload A"). We used the Zipfian request distribution and apply all operations to all fields of a document. Due to disk space limitations, the YCSB table is populated with 2 million records, resulting to a relatively high cache hit ratio, especially on machines with large physical memory. Read operations are thus expected to be significantly faster than writes, since the latter require synchronous disk access.

Each run is configured for a specific *target throughput* (ops/sec), using three different targets on each system: 500, 1000, and 1500 ops/sec for Workload B and 50, 100, and 150 ops/sec for Workload A. Lower throughput targets in the latter are due to costly write operations taking up 50% of the request mix. The choice of targets was made by experimentation to represent three distinct ("low", "medium", "high") load levels.

C. Parameter selection

A key step in our approach was the selection of the parameters that have the highest impact on the performance of the systems under study based on prior experience and expertise in the domain of NoSQL data stores and data-intensive applications. Each group of performance tests used for performance evaluation and prediction is executed with a different mix of values of the selected parameters, which fall under three categories:

Node description parameters characterize the number and characteristics of the machines that host the data store (Table Ia). We experimented on four machine types, thus each node parameter has four possible values in our evaluation. Generally speaking, including measurements from a wider range of machine types would increase the diversity of the parameter values. This way, in a scenario where we aim to predict performance on a new node type X, our regression model would rely on measurements from nodes that exhibit higher similarity in their parameters to X's.

Workload characterization parameters (Table Ib) describe the YCSB workloads used in our experiments, described in Section III-A. An important parameter is the load level issued to the data store, which is kept to a stable rate using the YCSB *target throughput* option, taking values in $\{500, 1000, 1500\}$ ops/sec for Workload B and $\{50, 100, 150\}$

TABLE I
PARAMETERS

Parameter	Description	Measurement Unit
a) NODE DESCRIPTION		
CPU cores	Number of cores allocated to this node type	Integer
Cache size	CPU cache size of the node type	MB
SPECint_base2006 score	SPECint 2006 benchmark score of node type	Float
Memory size	RAM size of the node type	MB
Operating system	Operating system type	Enum
Cached read speed	Cached reads measured with `hdparm` command	MB/sec
Buffered disk read speed	Buffered disk reads measured with `hdparm` command	MB/sec
Disk write speed	Disk writes measured with `dd` command	MB/sec
Network bandwidth	Average network bandwidth between nodes	Gbits/sec
Number of nodes	Total number of (same type) nodes in a cluster	Integer
b) WORKLOAD CHARACTERIZATION		
Record count	Number of records initially populating the data store	Integer
Read proportion	Proportion of read operations in the test	Float in range [0,1]
Update proportion	Proportion of update operations in the test	Float in range [0,1]
Request distribution	Distribution of requests to the data store	Enum:[random,latest,zipfian]
Read-all-fields	A read operation affects all record fields	Boolean
Write-all-fields	A write operation affects all record fields	Boolean
Load	Target operations per second	Ops/sec
c) DATA STORE CONFIGURATION		
Number of shards	Number of shards in the cluster	Integer
Number of replicas	Number of replicas per shard	Integer
Cache size	Cache memory size used by each data store node	MB
Write concern	Level of acknowledgement requested	Enum:[(0..#replicas),majority]
Read concern	Which data to return from a query	Enum:[local,majority]

ops/sec for Workload A. Variations of target throughput allow us to observe system responses under increasing load.

DATASTORE CONFIGURATION parameters (Table Ic) describe internal configurations typical of clustered NoSQL data stores. The number of shards expresses the degree of horizontal partitioning and the number of replicas the degree of redundancy in the cluster. Cache size refers to the amount of memory used by each data store node to cache data.

Most NoSQL data stores provide configuration knobs to tune the consistency, durability, and performance of operations. Our general guideline in this study is to involve a majority of replicas in each operation before responding to the client. To achieve this we set MongoDB's *read concern* and *write concern* options to "majority". For reads this means that a majority of replicas is contacted after a cache miss. A write operation returns to the client after it has been propagated to a majority of replicas and written to their on-disk journal. We believe that the settings used in the measurement collection process represent reasonable choices for a large range of applications, aiming for a balance between reliability and performance.

D. Performance prediction

We build prediction models based on MARS, SVR, and ANN using an open-source implementation of MARS called Earth [45], scikit-learn for SVR [46], and the Keras [47] library for ANN, all implemented in Python. As a baseline, we build a simple linear prediction model based on scikit-learn's linear Ordinary Least Square (OLS) regression [48], [49].

Each regression technique has a number of tuning parameters that can be configured with values that work well on the target dataset. After cross-validation we found that MARS achieved high performance with the parameters "thres" set to 0.001 and "min_search_points" to 100, "smooth" set to "true" and "allow_linear" to "false", leaving the rest to their default values [50]. For SVR we used the non-linear radial basis function (RBF) kernel. We also set C to 10^2 and epsilon to 10^{-5} [51]. For ANN, a sigmoidal activation [52] function with an RMSprop optimizer was found to work best, with the learning rate parameter "lr" set to 0.001 and the rest of the optimizer parameters left to their default values, as suggested in the Keras documentation [53]. The number of hidden layer neurons is set to be equal to the number of predictor variables and "nb_epoch" to 15,000.

After training a regression model, we attempt predictions to questions such as: "What is the read latency of YCSB over MongoDB when deployed on machines of a given number and type (Table Ia), workload characteristics (Table Ib), and data store settings (Table Ic)?". We term such questions "prediction queries", namely parameter values upon which the model is asked to predict read/update latency and CPU utilization.

We use *k-fold cross validation* [54], [55] to create robust regression models and avoid *data overfitting* [56]. After model generation, accuracy scores of each predicted value are calculated using the formula $1-(|X-Y|/X) \in [0,1]$, where X is the average of the observed values for a test case and Y the predicted value in the same case.

IV. Evaluation

All measurements in this study were based on YCSB workloads A and B [23], [44] with Zipf-distributed requests on a collection of 2 million documents. Each test is conducted in 1, 3, and 5-node setups using machines of type A–D (Section III-A), for a total of 12 system combinations.

A key observation is that MongoDB update operations are significantly more expensive than reads in all setups. Reads are benefiting from high hit ratios in server caches due to the moderate dataset size, large server memories, and Zipf access distribution. In addition, our choice of safe, durable writes, using the on-disk journal introduces expensive disk-sync operations in the update path. Using machines with more computing power does not improve performance of updates much, since the bottleneck is on the disk.

Moving from a 3-node to a 5-node cluster significantly reduces update latency in most cases due to better parallelism. Setups involving more powerful machines B–D exhibit read latency below 5ms (higher for machines of type A) under light and medium loads. As expected, read/update latencies increase under the high load level for all machine types. Server CPU utilization is below 30% in all cases involving machines B–D and below 65% across A–D.

We next present performance prediction results under three types of prediction queries, calling for prediction of latency and average server CPU utilization when varying one of three parameters: number of nodes, level of load, write concern:

- *C*ase 1: The prediction query targets MongoDB clusters when varying the *number of nodes* parameter (Table Ia) of Type B machines, under Workload B, with the load parameter fixed at 1000 ops/sec.
- *C*ase 2: The prediction query targets a 5-node MongoDB cluster deployed on Type C machines, under Workload A, varying the *load* parameter (Table Ib).
- *C*ase 3: The prediction query targets a 1-node MongoDB setup on a Type A machine with a load of 50 ops/sec under Workload A. It varies MongoDB's *write concern* parameter (taking values in {"0", "1"}) (Table Ic).

All regression methods were trained with the same data and asked the same prediction queries. To strengthen the validity of our results we apply k-fold cross-validation on our measurements, selecting $k = 5$ based on prior experience [57].

We report average accuracy scores for all regression methods in Table II. MARS exhibits the best accuracy, scoring above 97% in all cases with an average of 97.85%. An important feature of MARS is that it makes no assumptions of any underlying functional relationship between predictor parameters and responses, but builds this relationship entirely based on the training dataset. The results clearly demonstrate that it performs well in all cases, including on points outside the training dataset and those that exhibit high nonlinearity.

SVR has an accuracy score of over 92% across all cases, with an average of 94.16%, exhibiting lower accuracy when asked prediction queries that involve points outside the training dataset. ANN average accuracy is 90.39%, staying close to 90% in all cases. Similar to SVR and especially in the second test case, ANN has low accuracy scores for prediction queries involving points outside the training dataset.

We present detailed results for all regression methods and prediction queries for Cases 1 to 3 in Table III, where average accuracy calculated over the 5 rounds of the 5-fold cross-validation and standard deviation are reported. Points (parameter values) outside the training dataset appear in bold.

A key feature of NoSQL databases is their scalability. Predicting performance for different cluster sizes can help initial provisioning or responding to workload variations. For Case 1, we predict performance under different clustered deployments of MongoDB. We train our model with measurement data of 2-, 3- and 5-node setups and ask prediction queries on them, as well as on a 4-node untrained setup. Accuracy scores of MARS uniformly exceed 92% with an average of 96.93%, maintaining high prediction accuracy even outside the training dataset (4-node cluster). SVR has an average accuracy of 93.97%, that is reduced outside the training set. ANN delivers the lowest accuracy score with an average of 92.11%.

For Case 2, we train our model at 50, 100, and 150 ops/sec and attempt latency and CPU utilization predictions at 50, 70, 100, 120, and 150 ops/sec. We find that MARS prediction achieves a solid average accuracy score of 97.81% across all cases. The average accuracy scores of SVR and ANN in this case are 92.42% and 89.85% respectively. However, SVR and ANN exhibit lower scores on points outside the training set, reaching 60% for ANN in the case of update latency for 70 ops/sec.

Another relevant question is the performance impact of internal data store parameters. In particular, durability settings are often a key determinant of performance. Case 3 targets performance prediction under different *write concern* values. For this test case, we extended our training dataset with measurements where *write concern* w is set to 0 (i.e., 0-ack writes, featuring reduced durability but better performance), while maintaining values for all other MongoDB internal parameters as set throughout the measurement collection phase. The average accuracy of MARS in this case is uniformly above 97% with an average of 98.82%. SVR also exhibits high accuracy with an average of 96.11%. ANN has an average accuracy score of 89.22%. Although there are no untrained parameter values here, the update latency between w=0 and w=1 differs significantly as durability settings critically affect write performance. This special case was harder to predict with ANN in comparison with the other two regression methods.

The standard deviation for all accuracy scores is low (under 1%) except for a few cases exhibiting small spikes up to 3.21% for MARS, 5.22% for SVR and 3.98% for ANN.

For comparison, linear OLS regression achieves accuracies of around (average) 88% and 70% for Cases 1 and 2, well below MARS, SVR and ANN due to the non-linearity in these cases. Case 3 differs in that it involves only two (and trained) prediction points, making it an easy prediction target for OLS, scoring similar to MARS (98%) in this case. SVR and ANN improve their accuracy in Case 3 when configured for linear

TABLE II
COMPARISON OF REGRESSION METHOD ACCURACY

Regression Method	Case 1 Accuracy	Case 2 Accuracy	Case 3 Accuracy	Average
MARS	96.93	97.81	98.82	97.85
SVR	93.97	92.42	96.11	94.16
ANN	92.11	89.85	89.22	90.39

TABLE III
PERFORMANCE PREDICTION FOR MONGODB, TYPE B MACHINES, WORKLOAD B: VARYING NUMBER OF NODES

Measurement type	Test case	Parameter	% k-round Average Accuracy ($\pm$ stdev)		
			MARS	SVR	ANN
Read latency	Case 1	2 nodes	96.84 (2.64)	97.90 (2.39)	97.86 (2.70)
		3 nodes	99.07 (0.77)	99.07 (0.24)	97.86 (0.47)
		4 nodes	94.57 (0.90)	91.77 (0.81)	93.09 (1.10)
		5 nodes	92.95 (3.21)	93.86 (5.22)	94.33 (3.98)
	Case 2	50 ops/sec	98.86 (0.60)	97.64 (0.65)	99.90 (0.13)
		70 ops/sec	98.91 (0.20)	96.25 (0.60)	90.43 (1.02)
		100 ops/sec	98.63 (1.48)	93.06 (1.24)	98.63 (1.91)
		120 ops/sec	95.75 (0.56)	85.75 (0.86)	81.98 (0.96)
		150 ops/sec	96.40 (2.25)	63.64 (2.40)	96.12 (2.31)
	Case 3	w=0	99.42 (0.39)	88.28 (1.00)	92.33 (0.90)
		w=1	97.81 (0.85)	98.21 (0.21)	98.41 (0.15)
Update latency	Case 1	2 nodes	99.14 (0.19)	98.36 (0.88)	82.58 (0.79)
		3 nodes	99.06 (0.92)	96.97 (0.29)	78.67 (0.26)
		4 nodes	97.05 (0.27)	81.67 (1.09)	95.15 (0.80)
		5 nodes	99.13 (0.73)	96.27 (0.53)	84.34 (0.17)
	Case 2	50 ops/sec	98.98 (0.67)	95.82 (0.20)	83.16 (2.15)
		70 ops/sec	99.34 (0.22)	87.26 (1.75)	60.80 (1.66)
		100 ops/sec	98.79 (0.75)	97.73 (1.29)	96.12 (2.05)
		120 ops/sec	92.86 (0.35)	86.97 (1.67)	76.12 (3.67)
		150 ops/sec	98.76 (1.18)	99.27 (0.32)	98.04 (0.35)
	Case 3	w=0	98.84 (0.95)	95.59 (0.94)	50.03 (0.07)
		w=1	98.67 (1.02)	96.51 (0.28)	96.59 (0.16)
CPU utilization	Case 1	2 nodes	98.56 (0.68)	91.65 (1.53)	92.90 (1.40)
		3 nodes	97.27 (2.80)	96.19 (2.00)	97.23 (2.23)
		4 nodes	94.40 (0.92)	88.17 (0.99)	99.44 (1.05)
		5 nodes	95.16 (2.56)	95.72 (3.26)	91.89 (4.03)
	Case 2	50 ops/sec	98.83 (1.07)	96.27 (0.44)	95.18 (0.61)
		70 ops/sec	97.79 (0.18)	98.47 (0.97)	88.09 (0.62)
		100 ops/sec	99.46 (0.30)	97.04 (1.37)	96.67 (1.44)
		120 ops/sec	95.48 (0.15)	93.56 (1.30)	95.21 (2.73)
		150 ops/sec	98.43 (1.25)	97.58 (1.42)	98.52 (0.30)
	Case 3	w=0	99.12 (0.93)	98.87 (0.55)	98.86 (0.49)
		w=1	99.07 (0.97)	99.18 (0.21)	99.14 (0.19)

regression (linear SVR kernel and ANN activation function).

V. CONCLUSIONS

This paper evaluates a measurement-based approach to performance prediction in NoSQL systems using systematic measurements from executions of the YCSB benchmark over single-node or clustered deployments of MongoDB, under two different types of workload, three levels of load, and 12 system configurations (3 cluster sizes $\times$ 4 physical machine types) on iMinds' Virtual Wall infrastructure. Comparing three regression techniques, MARS, SVR, and ANN, we determine that MARS performs best, yielding an average accuracy of 97.85% vs. 94.16% of SVR and 90.39% of ANN. We presented a detailed analysis of predictions varying parameters across three dimensions: nodes, workload, and data store characteristics. Our approach yields solid prediction results validating its potential in the application domain of NoSQL data stores. We advocate use of our approach along with more extensive measurement collection from SFA-enabled infrastructures (more node types, workload parameters, data store types) for goal-oriented resource provisioning and quality of service management for data-intensive applications.

ACKNOWLEDGMENT

We thankfully acknowledge support of the UNICORN H2020 (GA no. 731846) EU project. We also acknowledge support of the iMinds (imec Ghent) research institute for providing us with access to their Virtual Wall infrastructure.

REFERENCES

[1] G. DeCandia, D. Hastorun, M. Jampani, G. Kakulapati, A. Lakshman, A. Pilchin, S. Sivasubramanian, P. Vosshall, and W. Vogels, "Dynamo: Amazon's highly available key-value store," in *Proceedings of the 21st ACM SIGOPS Symposium on Operating Systems Principles (SOSP '07)*, Stevenson, Washington, USA, 2007.

[2] F. Gessert and N. Ritter, "Scalable data management: NoSQL data stores in research and practice," in *Proceedings of the 32nd IEEE International Conference on Data Engineering, ICDE 2016, Helsinki, Finland, May 16-20, 2016*, 2016.

[3] S. D. Gribble, E. A. Brewer, J. M. Hellerstein, and D. Culler, "Scalable, distributed data structures for internet service construction," in *Proceedings of the 4th Conference on Symposium on Operating System Design & Implementation (OSDI '00)*, San Diego, California, 2000.

[4] S. Becker, H. Koziolek, and R. Reussner, "Model-based performance prediction with the Palladio component model," in *Proceedings of the 6th International Workshop on Software and performance*. Buenos Aires, Argentina: ACM, 2007.

[5] S. Chen, Y. Liu, I. Gorton, and A. Liu, "Performance prediction of component-based applications," *Journal of Systems and Software*, vol. 74, no. 1, 2005.

[6] F. Cruz, F. Maia, M. Matos, R. Oliveira, J. Paulo, J. Pereira, and R. Vilaça, "Resource usage prediction in distributed key-value datastores," in *Proceedings of Distributed Applications and Interoperable Systems*. Springer, 2016.

[7] J. Guo, K. Czarnecki, S. Apel, N. Siegmund, and A. Wasowski, "Variability-aware performance prediction: A statistical learning approach," in *Automated Software Engineering (ASE), 2013 IEEE/ACM 28th International Conference on*. Silicon Valley, California, USA: IEEE, 2013.

[8] N. Siegmund, S. S. Kolesnikov, C. Kästner, S. Apel, D. Batory, M. Rosenmüller, and G. Saake, "Predicting performance via automated feature-interaction detection," in *Proceedings of the 1st International Conference on Cloud Computing and Services Science*. Noordwijkerhout, Netherlands: IEEE Press, 2012.

[9] D. Westermann and J. Happe, "Towards performance prediction of large enterprise applications based on systematic measurements," in *Proceedings of the Fifteenth International Workshop on Component-Oriented Programming (WCOP)*. Prague, Czech Republic: Citeseer, 2010.

[10] S. Dipietro, G. Casale, and G. Serazzi, "A queueing network model for performance prediction of apache cassandra," in *Proceedings of 10th EAI International Conference on Performance Evaluation Methodologies and Tools (Valuetools)*. Taormina, Italy: ACM, 2016.

[11] A. Gandini, M. Gribaudo, W. J. Knottenbelt, R. Osman, and P. Piazzolla, "Performance evaluation of nosql databases," in *Proceedings of the European Workshop on Performance Engineering*. Springer, 2014, pp. 16–29.

[12] R. Osman and P. Piazzolla, "Modelling replication in nosql datastores," in *Proceedings of the International Conference on Quantitative Evaluation of Systems*. Springer, 2014, pp. 194–209.

[13] S. Liu, S. Nguyen, J. Ganhotra, M. R. Rahman, I. Gupta, and J. Meseguer, "Quantitative analysis of consistency in nosql key-value stores," in *Proceedings of the International Conference on Quantitative Evaluation of Systems*. Springer, 2015, pp. 228–243.

[14] M. Chalkiadaki and K. Magoutis, "Managing Service Performance in the Cassandra Distributed Storage System," in *Proceedings of 5th IEEE International Conference on Cloud Computing Technology and Science (CloudCom)*, Bristol, UK, 2013.

[15] "MongoDB," https://www.mongodb.com/, accessed: 2017-4-10.

[16] S. Balsamo, A. Di Marco, P. Inverardi, and M. Simeoni, "Model-based performance prediction in software development: A survey," *IEEE Transactions on Software Engineering*, vol. 30, no. 5, 2004.

[17] J. Sankarasetty, K. Mobley, L. Foster, T. Hammer, and T. Calderone, "Software performance in the real world: personal lessons from the performance trauma team," in *Proceedings of the 6th international workshop on Software and performance*. Buenes Aires, Argentina: ACM, 2007.

[18] D. Westermann, J. Happe, M. Hauck, and C. Heupel, "The performance cockpit approach: A framework for systematic performance evaluations," in *The 36th EUROMICRO Conference on Software Engineering and Advanced Applications*. Lille, France: IEEE, 2010.

[19] D. Westermann and J. Happe, "Performance cockpit: systematic measurements and analyses," in *Proceedings of the 2nd ACM/SPEC International Conference on Performance engineering*. Karlsruhe, Germany: ACM, 2011.

[20] D. Westermann, R. Krebs, and J. Happe, "Efficient experiment selection in automated software performance evaluations," in *European Performance Engineering Workshop*. Borrowdale, UK: Springer, 2011.

[21] C. Weiss, D. Westermann, C. Heger, and M. Moser, "Systematic performance evaluation based on tailored benchmark applications," in *Proceedings of the 4th ACM/SPEC International Conference on Performance Engineering*. Prague, Czech Republic: ACM, 2013.

[22] "Standard Performance Evaluation Corporation (SPEC)," https://www.spec.org, accessed: 2017-4-10.

[23] "YCSB Core Workloads," https://github.com/brianfrankcooper/YCSB/wiki/Core-Workloads, accessed: 2017-4-10.

[24] J. H. Friedman, "Multivariate adaptive regression splines," *The Annals of Statistics*, vol. 19, no. 1, 03 1991.

[25] M. Courtois and M. Woodside, "Using regression splines for software performance analysis," in *Proceedings of the 2nd International Workshop on Software and Performance*, Ottawa, ON, Canada, 2000.

[26] 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*. New York, NY, USA: ACM, 2013.

[27] 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*. Essen, Germany: ACM, 2012.

[28] W.-Y. Loh, "Classification and regression trees," *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, vol. 1, no. 1, 2011.

[29] H. Koziolek, "Performance evaluation of component-based software systems: A survey," *Performance Evaluation*, vol. 67, no. 8, 2010.

[30] S. Becker, H. Koziolek, and R. Reussner, "The Palladio component model for model-driven performance prediction," *Journal of Systems and Software*, vol. 82, no. 1, 2009.

[31] H. Koziolek, S. Becker, and J. Happe, "Predicting the performance of component-based software architectures with different usage profiles," in *Proceedings of the 3rd International Conference on the Quality of Software Architectures (QoSA '07)*. Medford, MA, USA: Springer, 2007.

[32] V. Vapnik, *The nature of statistical learning theory*. Springer Science & Business Media, 2013.

[33] H. Drucker, C. J. Burges, L. Kaufman, A. Smola, and V. Vapnik, "Support vector regression machines," in *Advances in Neural Information Processing Systems 9*. MIT Press, 1997.

[34] S. Uttamchandani, L. Yin, G. A. Alvarez, J. Palmer, and G. A. Agha, "Chameleon: A self-evolving, fully-adaptive resource arbitrator for storage systems," in *Proceedings of the 2005 USENIX Technical Conference*. Anaheim, CA, USA: USENIX, 2005.

[35] M. Caudill, "Neural networks primer, part i," *AI expert*, vol. 2, no. 12, 1987.

[36] B. Mozafari, C. Curino, A. Jindal, and S. Madden, "Performance and resource modeling in highly-concurrent OLTP workloads," in *Proceedings of the 2013 acm sigmod international conference on management of data*. New York, NY, USA: ACM, 2013.

[37] E. Ipek, B. R. De Supinski, M. Schulz, and S. A. McKee, "An approach to performance prediction for parallel applications," in *Proceedings of the 11th International European Conference on Parallel Processing*. Lisbon, Portugal: Springer, 2005.

[38] M. Schulz, "Parameter-driven application performance prediction," http://www.research.ibm.com/actc/workshop/MartinSchultz.pdf, accessed: 2017-4-10.

[39] M. Wang, K. Au, A. Ailamaki, A. Brockwell, C. Faloutsos, and G. R. Ganger, "Storage device performance prediction with CART models," in *Proceedings of the 12th Annual International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunications Systems (MASCOTS, '04)*. IEEE, 2004.

[40] R. P. Martin and D. E. Culler, "NFS sensitivity to high performance networks," *ACM SIGMETRICS Performance Evaluation Review*, vol. 27, no. 1, 1999.

[41] E. A. July and E. Anderson, "Hpl–ssp–2001–4: Simple table-based modeling of storage devices," Tech. Rep., 2001.

[42] "iMinds Virtual Wall," https://www.fed4fire.eu/virtual-wall/, accessed: 2017-4-10.
[43] "jFed," http://jfed.iminds.be/, accessed: 2017-4-10.
[44] 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*. Indianapolis, Indiana, USA: ACM, 2010.
[45] "py-earth on GitHub," https://github.com/scikit-learn-contrib/py-earth, accessed: 2017-4-10.
[46] "Python scikit-learn," http://scikit-learn.org/stable/, accessed: 2017-4-10.
[47] "Keras," https://keras.io/, accessed: 2017-4-10.
[48] F. Galton, "Regression towards mediocrity in hereditary stature." *The Journal of the Anthropological Institute of Great Britain and Ireland*, vol. 15, pp. 246–263, 1886.
[49] "Python scikit-learn ordinary least squares regression," http://scikit-learn.org/stable/modules/linear_model.html, accessed: 2017-4-10.
[50] "py-earth API," http://contrib.scikit-learn.org/py-earth/content.html#api, accessed: 2017-4-10.
[51] "Python scikit-learn support vector regression," http://scikit-learn.org/stable/modules/generated/sklearn.svm.SVR.html, accessed: 2017-4-10.
[52] "Keras Activations," https://keras.io/activations/, accessed: 2017-4-10.
[53] "Keras Optimizers," https://keras.io/optimizers/, accessed: 2017-4-10.
[54] R. R. Picard and R. D. Cook, "Cross-validation of regression models," *Journal of the American Statistical Association*, vol. 79, no. 387, 1984.
[55] T. J. Hastie, R. J. Tibshirani, and J. H. Friedman, *The elements of statistical learning: data mining, inference, and prediction*. Springer, 2011.
[56] D. M. Hawkins, "The problem of overfitting," *Journal of chemical information and computer sciences*, vol. 44, no. 1, 2004.
[57] D. Anguita, L. Ghelardoni, A. Ghio, L. Oneto, and S. Ridella, "The 'K'in K-fold cross validation," in *Proceedings of the European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning.*, Bruges, Belgium, 2012.