On Exploiting Resource Diversity in the Public Cloud for Modeling Application Performance

Mark Meredith
Dept. of Comp. Sci. and Engg.
The Penn State University
Email: mwm126@cse.psu.edu

Bhuvan Urgaonkar
Dept. of Comp. Sci. and Engg.
The Penn State University
Email: bhuvan@cse.psu.edu

***Abstract*—Cloud computing platforms, such as Amazon EC2, Google Computing Engine, and Microsoft Azure, offer dozens of virtual machine (VM) types with a wide range of resource capacity vs. price trade-offs, requiring a customer to consider numerous resource configurations when evaluating service needs. We investigate the possibility of exploiting this diversity of VM types to predict the performance of workloads on new VM types using black box modeling. The performance model used is a multiple linear regression of the average application response time as a function of VM load (throughput in requests per second), the number of CPU cores, and main memory capacity. For three different types of data storage applications - Redis (key-value stores), Apache Cassandra (a NoSQL database) and MySQL (an ACID database) - the model accuracy improves when the training data spans more diverse VMs. E.g., for Redis, the $R^2_{predicted}$ measure of model efficacy improves from 0.4-0.5 with 2 VM types for training and 0.7 for 3 VM types to 0.8 for 4 VM types. These results suggest further interesting research challenges, such as the possibility of automating the process of calibrating performance models using diverse resource types on a public cloud leading to "performance modeling as a service."**

***Keywords*—public cloud; tenant workload; performance modeling**

I. INTRODUCTION

Many enterprises are migrating their information technology (IT) needs to public cloud computing platforms, a trend that is projected to continue unabated in the foreseeable future [11]. Procuring resources cost-effectively from a public cloud poses significant technical challenges. One such challenge concerns the problem of determining the set of IT resources (including their capacities) - virtual machines (VMs), the virtual network connecting these VMs, storage, etc. - that would be needed to cost-effectively meet the predicted workload of the tenant's software applications while offering satisfactory performance and availability to its users. In order to solve this problem, a tenant must first solve the problem of assessing the performance the users of its application software are likely to experience if the application were assigned a given set of IT resources to meet its predicted workload. Our interest in this paper is in this latter problem, often labeled *application performance modeling* [13] [17] [18] [21] [25] [29] [31] [34]. Of course, application performance modeling has many other uses besides cost optimal resource procurement, e.g., anomaly detection [7] [15] and capacity planning [19].

Whereas application performance modeling has been an area of extensive research for many decades across many communities, solving it for the public cloud ecosystem presents a tenant with non-trivial novel sources of complexity. In particular, most modeling solutions have traditionally been developed for settings involving privately owned and operated data centers or clusters. These solutions may not be readily adapted to a public cloud.

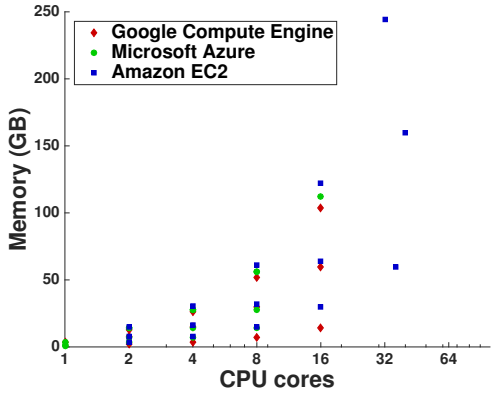


Fig. 1. An illustration of the diversity in VM capacities offered by popular public cloud providers.

One of the most important differences between these two settings (from the point of view of application performance modeling) is the immense *diversity of resources* that a typical public cloud offers. There are other important differences complementary to our focus in this paper and part of our future work. E.g., in a private setting, the user of a machine (tenant application) coincides with its owner while in a public setting the two are separated via virtualization techniques with implications for how much information about physical resource usage is available to the tenant's models. We focus on VMs in this work although our arguments likely apply to other resource types as well. To appreciate this diversity, let us consider some examples from the most prominent public cloud providers that offer many VM types since they need to cater to many different types of customers. Here, VM instance types are organized into groups based on use case. Instances within a group generally have the same CPU generation and clock speed, and vary by the number of CPUs and memory. Amazon EC2 offers over 40 VM types organized into eight different groups, varying in CPU, memory, network bandwidth, storage speed, and pricing [1]. Google Compute Engine offers 15 instance types organized into four groups:

 ISBN: 978-1-61208-529-6

Standard, High CPU, High Memory, and Shared Core (for lightweight applications) [12]. Finally, Microsoft Azure offers 30 instance types organized into 4 groups [33]. Fig. 1 shows 44 VM instance types from these three providers capturing the large spectrum of CPU cores and memory that their VMs pack. We show VMs with a wide range of CPU and memory capacities offered by Amazon EC2 [1], Google Compute Engine [12], and Microsoft Azure [33]. The number of cores on these VMs ranges from 1 to 32 whereas their memory capacity ranges from 0.75GB to 256GB.

A typical privately owned data center, on the other hand, is likely to possess a much smaller number of machine types. Keeping machines (and their software configurations) relatively homogeneous brings about significant benefits related to ease of system administration and cost savings (e.g., due to bulk purchase offers from IT vendors). Although factors, such as incremental procurement over time, meeting specialized needs (e.g., machines with GPUs), etc., do result in some differences among machine types even in a private data center, the overall degree of heterogeneity is significantly smaller than that seen in a public cloud.

This high diversity of resource types in a public cloud introduces an additional source of complexity into a tenant's VM autoscaling decision-making. Since the number of machine types in traditional IT environments is small and relatively fixed, performance models have conventionally been developed and calibrated using performance measurements ("profiling") on the same/similar type of machines on which the application would eventually execute. On the other hand, a tenant of a public cloud would be interested in predicting the performance its workload might experience on a wide variety of VM types that the provider offers. This is because the VM types most cost-effective for a tenant's workload might change over time due to: (i) changes in the tenant's own workload (e.g., many applications show periodic time-of-day or seasonal variations in their workload intensities) and (ii) dynamism and variety in the cloud provider's pricing schemes (e.g., Amazon EC2 offers spot pricing for most of its instance types and such spot instances are usually much cheaper than their on-demand counterparts). Additionally, existing work also shows that even during a period of stationary workload, procuring heterogeneous VMs (e.g., a combination of "small" and "large") can often offer a better cost vs. performance trade-off than procuring the same types of VMs (e.g., a larger number of only "small" or a smaller number of only "large") [35] Approaches based on calibrating a tenant's performance models separately on the dozens of resource types that public clouds offer are likely to not scale well.

We wish to explore if a tenant might actually be able to benefit from this diversity by deliberately and carefully exploiting it to ease the creation and calibration of its application performance models. The intuition underlying our premise is that choosing a small subset of the offered VMs may suffice for calibrating a tenant's performance models well if this subset were chosen carefully. In particular, this subset should capture well the overall diversity across the VMs offered by the provider.

Our Approach and Contributions: In this paper, we take a small first step towards exploring the above idea by evaluating the following hypothesis: *using a more diverse set of VMs for calibrating/training a performance model helps improve its accuracy*. Specifically, we devise a multiple linear regression modeling framework for predicting the performance of interactive data serving applications. We calibrate this model for three different types of real-world applications: (i) Redis [23], an open-source in-memory NoSQL key-value store, (ii) Apache Cassandra [5], a Table/key-value hybrid NoSQL database, and (iii) MySQL, a popular open-source ACID database [22]. We use "training sets" of varying sizes (i.e., numbers of VM types) for our calibration and investigate the impact of the training set size on model efficacy. Our results are promising. E.g., for Redis, we find that the $R^2_{predicted}$ measure of model efficacy improves from 0.4-0.5 with 2 VM types for training and 0.7 with 3 VM types to 0.8 for 4 VM types.

Whereas the benefits of exploiting heterogeneity have been explored in other contexts (most notably for cost/performance optimization in cloud settings [10] [16] [24] [35]), to the best of our knowledge, our paper is the first to systematically explore its role in aiding performance model calibration. Our work is complementary to traditional performance modeling research. At the same time, it opens up a promising new area for further exploration. As part of our own future work, we plan to investigate if/how public cloud providers could offer "performance modeling as a service," whereby all/many aspects of the model calibration by exploiting diversity would be offered as an automated facility to their tenants.

The rest of this paper is organized as follows. In Section II, we provide an overview of a generic cost-conscious tenant's decision-making and where application performance modeling fits within it. In Section III, we describe the performance modeling techniques that we employ. In Section IV, we present our empirical evaluation of our hypothesis using three real-world applications as our case studies. Finally, we discuss related work in Section V.

II. Context and Overview

The tenant would employ observations of its workload intensity in the past to predict its future workload. Consider the example of a key-value store or a database application that we employ in our evaluation in Section IV. Such an application might keep track of request arrival rates (possibly for different request classes) and then use a suitable prediction mechanism for estimating future arrival rate. Whereas some tenant applications exhibit significant predictability (e.g., captured well via Markovian or autoregressive models) [3] [6] [27] [28], others are known to exhibit poor predictability and must resort to short-term ("myopic") estimates [9] [14] [32]. Regardless, having made these predictions, the tenant must then ascertain the number and type of VMs that it must procure from the cloud to meet its performance needs (or deallocate from its existing resource pool).

 ISBN: 978-1-61208-529-6

We show one way of thinking about this decision-making, wherein the tenant determines (using its application performance model) multiple VM allocation choices that would allow it to meet its predicted workload with the requisite performance goals. These choices are then compared in terms of their costs (or expected profits, if the tenant wishes to maximize expected profits rather than minimizing costs) via an optimization problem that incorporates idiosyncrasies of the prices offered by the public cloud. The actual realization of this overall decision-making may be different from how we describe it here. Our description is deliberately designed to highlight the role of the application performance model. Finally, the most cost-effective choice identified by the optimizer is used as the basis for actually procuring the appropriate number and type of VMs from the cloud provider.

Significant research exists both on predicting workloads and on performance modeling (see Section V) for a wide variety of application types. Recall that our interest in this paper is not on devising new techniques for workload prediction or application performance modeling. Rather, we are interested in evaluating the role VM diversity might play in calibrating a *given* performance model. Towards this, we adapt a popular modeling approach as described next.

III. Our Modeling Methodology

This section describes the general performance modeling ideas used. In Section IV, this basic model is adapted to three application case studies on the Amazon AWS cloud. The primary interest in this paper is not in identifying the most accurate performance model but rather in exploring if diversity in the VMs used for calibrating the chosen model helps improve its efficacy. Therefore, although numerous modeling choices exist in the literature for such applications, we use a relatively simple multiple linear regression approach because: (i) it serves as a good starting point for evaluating the hypothesis, (ii) it is easy to cast, train, and evaluate, and (ii)it works well - especially under low/moderate throughputs for the normal operating regions of well-provisioned, performance-sensitive tenant workloads.

Multiple Linear Regression: Given a data set $\{y_i, x_{i1}, \ldots, x_{ip}\}_{i=1}^{n}$, a linear regression on multiple independent variables x_p and dependent variable y is a set of parameters β_i that model a linear relationship between y and x_i as $y_i = \beta_1 x_{i1} + \ldots + \beta_p x_{ip} + \varepsilon_i$ [26].

The parameters ε_i are the error terms, an unobserved random variable. The parameters β_i are chosen to minimize the values of ε_i for the entire data set. Specifically, the β_i are chosen to minimize the sum of squares $\sum_{i=1}^{n} \varepsilon_i^2$.

We choose as our dependent variable the average latency y_L and as our independent variables: (i) workload/application properties - throughput, degree of replication, and read/write ratio and (ii) resource capacity of the VMs being used - number of CPU cores, clock rate of each CPU, memory, network bandwidth, and type of storage (SSD vs. magnetic).

We define a training set $S = \{VM_i\}_{i=1}^{n}$ as a set of virtual machines, each characterized by x_p. In each of our experiments, we run the application whose performance we wish to model on VM_i for various x_T and measure the latency x_L. We then find a multiple linear regression M_S on $\{y_{iL}, x_{iT}, \ldots, x_{ip}\}_{i=1}^{n}$. (For a given instance VM_i, the values of x_{ip} are fixed for all measurements for that instance.)

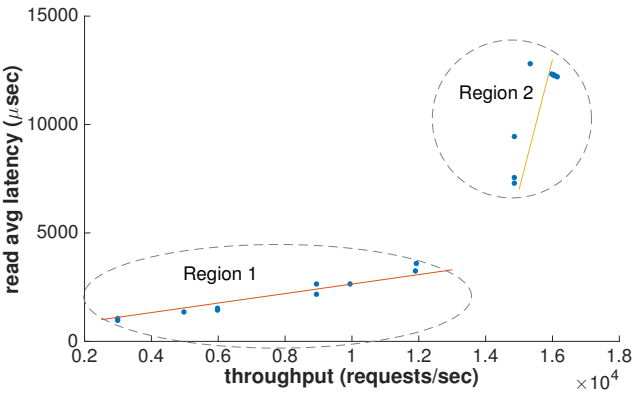


Fig. 2. Two regions of latency vs. throughput for Redis.

Measure of Model Efficacy: We use the predicted coefficient of multiple determination ($R^2_{predicted}$) as our measure of model accuracy which is defined as follows. For a test instance VM_{test} with $x_{test,i}$, $R^2_{predicted} = 1 - \frac{\sum_{i=1}^{n}(y_{test,i} - \hat{y}(x_{test,i}))^2}{\sum_{i=1}^{n}(y_{test,i} - \bar{y_{test}})^2}$, where $\hat{y}(x_{test,i}) = \sum_{i=1}^{n} \beta_i x_{test,i}$, and $\bar{y}_{test}$ is the mean of $y_{test,i}$.

To see evidence supporting our hypothesis, we expect to see the following behavior: for larger training sets S, the model should fit better to VM_{test}, corresponding to an increasing $R^2_{predicted}$, assuming sufficient variability in the values of x_p for $VM_j \in S$ to cover the values of x_p for VM_{test}.

Discussion: Our linear regression based model is known to perform poorly when queueing delays become dominant contributors to overall latency [29]. For example, if we were to model the entire set of latency observations (for experiments done using Redis, more details in Section IV-B) using our model, we would obtain a poorer predictor than the two separate linear regression models shown in Fig. 2, one each for the "low/moderate" (Region 1) and "high" (Region 2) throughput regions. This suggests two points: (i) using domain knowledge (e.g., the distinction between low and high throughput regions), a tenant may be able to use linear regression to obtain better models, and (ii) more sophisticated models may be warranted for the needs of certain tenants. Again, since our interest is in the impact of diversity on modeling accuracy, we focus only on modeling performance in Region 1 for the rest of this paper.

It is important to keep in mind the basic assumption of linear regression about the independent variables being independent of each other (i.e., the x_p for $VM_j \in S$ need to be independent). Interestingly, among the independent variables in our model, the number of cores and memory capacity are prone to be problematic on this front - typically larger VMs come both with more CPUs and more memory - see Fig. 1. To overcome this problem, we attempt to choose VM types in our experi-

 ISBN: 978-1-61208-529-6

TABLE I
AWS INSTANCE TYPES EMPLOYED IN OUR EVALUATION.

Instance name	Abbr.	# cores	Memory	Network
m3.large	VM_1	2	7.5 GB	Moderate
m3.xlarge	VM_2	4	15 GB	Moderate
m3.2xlarge	VM_3	8	30 GB	High
r3.large	VM_4	2	15 GB	Moderate
r3.xlarge	VM_5	4	30.5 GB	Moderate
r3.2xlarge	VM_6	8	61 GB	High

ments where this correlation is weak. Furthermore, the results we present in this paper are for a subset of our experimental findings wherein the entire working set fits in VM memory, rendering memory moot as a predictor of performance (we do incorporate memory in our more general experiments). Finally, the potential shortcomings of predicted R^2 as a measure of model accuracy should be kept in mind when interpreting our results [26].

IV. EVALUATION

A. Methodology and Setup

We carry out our evaluation on the EC2 public cloud offered by Amazon Web Services (AWS) [1]. We adapt our generic performance model from Section III for three different types of latency-sensitive data-serving applications: (i) Redis (an in-memory open-source NoSQL key-value store) [23], (ii) Apache Cassandra (a NoSQL key-value store that can be configured for different consistency levels), and the popular MySQL ACID database [22]. We use the open-source Yahoo! Cloud Serving Benchmark (YCSB) as our workload generator [8]. We run the YCSB client on a m4.2xlarge EC2 instance running Ubuntu Linux 14.04. We monitor the system load average on the client machine to verify that the client is not the bottleneck during our tests.

We describe a subset of our overall results wherein for each experiment we load the concerned database with 1,000,000 records, each containing ten fields of 100 bytes each (the default). This amounts to an overall working set of about 1GB. Each experiment consists of subjecting the database to a particular thoughput and recording the average latency (separately for reads and writes). YCSB defines several standard workloads that we experiment with. We experiment with different workloads offered by YCSB and present a subset of our overall results - workload "A" for MySQL and "B" for Redis and Cassandra. Workload A has 50% read and 50% write requests and employs a uniform popularity distribution. Workload B has 95% reads and 5% writes, and the popularity of requests is chosen based on a zipfian distribution. We repeat each experiment several times to achieve significantly tight confidence intervals. Amazon EC2 is hosted in multiple geographic regions around the world, and multiple zones within each region. We create our testing client and servers within the same region (us-west) and availability zone (2b) to minimize the effect of network latency. Finally, we pick all VMs having individual CPUs offering the same clock rate.

We select the VM instances listed in Table I. There are three instances from the M3 group (standard) and three from the R3 group (memory optimized). The memory/CPU ratio is the same within each group, with the R3 group having twice the memory/cpu as the M3 group. A more extensive study with more instances would allow the use of more independent variables in the model, e.g., including testing of instances in the C3 group (compute optimized), which have half as much memory/CPU as M3.

With the above choices, the measurements and modeling reported here effectively only employ a subset of all the independent variables listed in Section III: number of CPU cores, throughput, number of replicas, and read-write ratio. In particular, our working set of 1GB fits fully within any of the chosen VMs, effectively rendering memory capacity moot as a predictive variable. In our more general experiments, however, we explore a much larger set of workload choices.

B. Case Study 1: Redis

Redis is an open-source, key/value NoSQL database. Redis is in-memory and, therefore, very fast. Redis also optionally supports persistence, so unlike memcached it can be used as a primary database or as a cache. We deploy Redis on AWS using Amazon ElastiCache, a web service that abstracts the deployment and administration of the OS and database software. Elasticache supports up to five read replicas of the primary database. We report results with a single replica here.

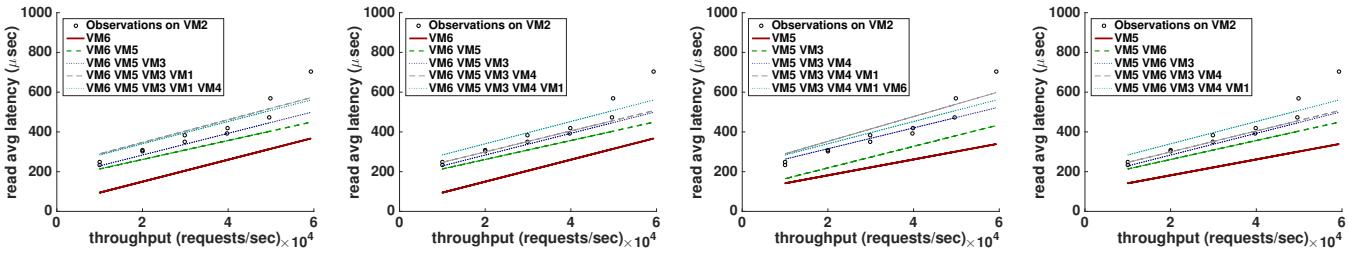
For a VM type on which we wish to predict Redis performance, we choose training sets of different sizes among the remaining VM types. We find that each time we add a new instance type to the training set, $R^2_{predicted}$ does improve for both read and write latency. Generally, we observe that a training set of only 3 VM types appears to offer high accuracy with further additions offering relatively low gains. This bodes well for cost-efficacy of our model calibration approach - instead of having to calibrate its performance model for dozens of VM types (with associated costs), a tenant may be able to achieve comparable model accuracy using a much smaller set. We present representative findings in Figs. 3 and 4.

C. Case Study 2: Apache Cassandra

Apache Cassandra is a Table/Key-Value hybrid NoSQL database. It is suitable for applications that require high availability provided by replication. In terms of the CAP theorem, Cassandra prioritizes availability and performance over consistency, making it highly performant and scalable, though consistency is eventual rather than strong, for typical Cassandra applications. We do our testing on Cassandra clusters with 5 nodes. We run our testing with a replication factor of three, so every database record is stored on three of the five nodes. We record report results both when using Cassandra's weak (or eventual) and strict consistency settings.

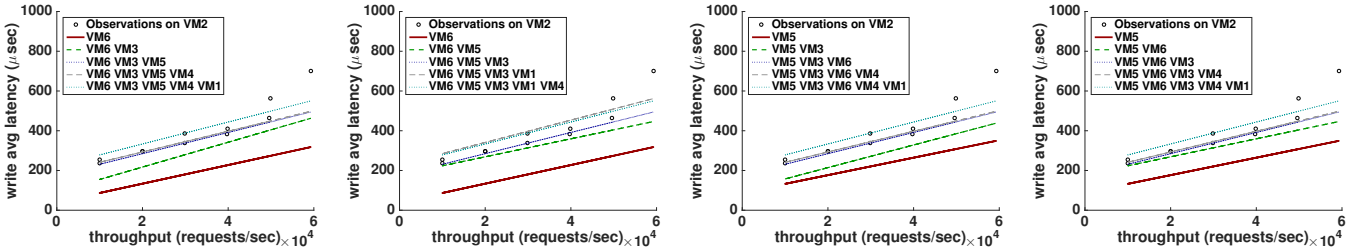
For a sample VM type that we want to predict the performance of Cassandra, we select training sets of increasing size from the remaining VM types. We select VM_4 for prediction,

 ISBN: 978-1-61208-529-6



(a) (b) (c) (d)

Fig. 3. Prediction of Redis read latency on VM_2 compared for model calibration using a variety of training sets ranging in size from 1 to 5 VM types.



(a) (b) (c) (d)

Fig. 4. Prediction of Redis write latency on VM_2 compared for model calibration using a variety of training sets ranging in size from 1 to 5 VM types.

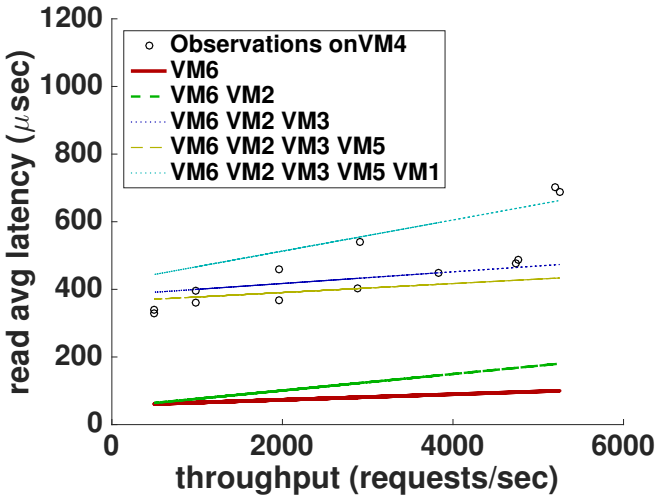


Fig. 5. Read latency/throughput plot for MySQL.

and do multiple linear regressions on the training set for sizes 1 through 5 for read latency and write latency data. Again, we observe that every time another VM type is added to the training set, the associated $R^2_{predicted}$ improves for VM_4 for both read and write latency (results omitted for space). We conclude that our evaluation offers supporting evidence for our second case study.

D. Case Study 3: MySQL

MySQL is an extremely popular ACID SQL database server, the backbone of numerous commercial applications. For our testing we deployed MySQL using Amazon Relational Database Service (Amazon RDS) [2], which abstracts away the deployment and administration of OS and relationship database software. We benchmark MySQL on the six VM types listed in Table I.

The MySQL data shows a higher variation in latency than our other case studies, and our linear regression model does not fit it as well as it does the previous two applications. We show a sample result for MySQL in Fig. 5. We do continue to see that the model accuracy as captured by $R^2_{predicted}$ does continue to improve with the addition of more VM types to the training set, although the value of $R^2_{predicted}$ is lower than observed for Redis and Cassandra.

This may be due to the higher write latency of SQL databases, or possibly something with Amazon's RDS architecture that made our YCSB testing method unsuitable. Another possibility is that the network availability for RDS varies over time, making consistent results harder to reproduce. This requires further investigation that forms part of our future work. Despite these inadequacies in our modeling, however, the basic expectation we have regarding the role of diversity does appear to hold.

V. Related Work

Performance modeling of software applications has a long history in a variety of domains. Approaches range from explicit modeling leveraging knowledge the internal workings of system (e.g., based on queueing theory or more general Markovian models [17] [18] [25] [31]), "black box" approaches (ranging from relatively simple regression [29] similar to this paper to more sophisticated statistical learning-based [13] [20]

[21] [34]), and combinations ("gray-box" approaches) [4] [30]. As explained earlier, we choose to work with a very simple modeling approach because our main interest was not in high accuracy modeling but on evaluating the improvements that can result from exploiting diversity. All existing work on modeling is complementary to our ideas and we hope to explore the efficacy of our hypothesis with more sophisticated modeling techniques.

A substantial body of work exists on exploiting different forms of heterogeneity (not just in cloud platforms but also in other types of systems) for cost and/or performance optimization [10] [16] [24] [35]. Our goal is different from these works in that our interest is in using diversity for improving modeling accuracy.

VI. Conclusions

The diversity of resource types offered by public clouds is much higher than in conventional privately-owned data centers. The complexity of options available makes decision on resource acquisition more complex, with the larger range of service available. Tenants may need to calibrate their application performance models for a large number of resource configurations. In a private cloud, the system on which such calibration is done is the same as the machines on which the application eventually runs. This paper investigates the possibility of exploiting this diversity to ease the tenant's performance modeling.

To explore this idea we applied a linear regression model to the relationship between latency and throughput for several popular database servers. The model expressed the average response time of a class of interactive server applications as a linear combination of the offered throughput (requests/s), the number of CPU cores and memory in the procured VM, and degree of replication employed by the application. For three different real-world applications - Redis, Apache Cassandra, and MySQL - the model accuracy increased for more diverse sets of VMs. For example, the $R^2_{predicted}$ measure of model efficacy for Redis improved from 0.4-0.5 with 2 VM types for training and 0.7 for 3 VM types to 0.8 for 4 VM types. Qualitatively similar results were observed for Apache Cassandra and MySQL. Although this modeling approach is very specific, the basic observation could be expanded to more accurately model more VM types in more detail. This would lead to more interesting research challenges, such as automating the process of calibrating performance models using diverse resource types on a public cloud allowing providers to offer "performance modeling as a service" to their tenants.

References

[1] Amazon EC2 Pricing. http://aws.amazon.com/ec2/pricing/, [last accessed October 2016].

[2] Amazon Relational Database Service. https://aws.amazon.com/rds/, [last accessed October 2016].

[3] M. F. Arlitt and C. L. Williamson. Internet web servers: Workload characterization and performance implications. *IEEE/ACM Trans. Netw.*, 5(5), Oct. 1997.

[4] A. C. Arpaci-Dusseau and R. H. Arpaci-Dusseau. Information and control in gray-box systems. In *Proc. ACM SOSP*, 2001.

[5] The Apache Cassandra Project. http://cassandra.apache.org/, [last accessed October 2016].

[6] A. Chandra, W. Gong, and P. J. Shenoy. Dynamic resource allocation for shared data centers using online measurements. In *Proc. IWQoS*, 2003.

[7] I. Cohen, M. Goldszmidt, T. Kelly, J. Symons, and J. S. Chase. Correlating instrumentation data to system states: A building block for automated diagnosis and control. In *Proc. USENIX OSDI*, 2004.

[8] B. F. Cooper, A. Silberstein, E. Tam, R. Ramakrishnan, and R. Sears. Benchmarking cloud serving systems with ycsb. In *Proc. ACM SOCC*, 2010.

[9] M. E. Crovella and A. Bestavros. Self-similarity in world wide web traffic: Evidence and possible causes. *IEEE/ACM Trans. Netw.*, 5(6), Dec. 1997.

[10] Farley, B. et al. More for your money: Exploiting performance heterogeneity in public clouds. In *Proc. ACM SOCC*, 2012.

[11] Forbes. http://www.forbes.com/sites/louiscolumbus/2015/04/05/predicting-the-future-of-cloud-service-providers/, [last accessed October 2016].

[12] Google Compute Engine: Machine Types. https://cloud.google.com/compute/docs/machine-types, [last accessed October 2016].

[13] H. Herodotou and S. Babu. A what-if engine for cost-based mapreduce optimization. *IEEE Data Eng. Bull.*, 36(1):5–14, 2013.

[14] J. Jung, B. Krishnamurthy, and M. Rabinovich. Flash crowds and denial of service attacks: Characterization and implications for cdns and web sites. In *Proc. WWW*, 2002.

[15] T. Kelly. Detecting performance anomalies in global applications. In *Proc. 2nd Conference on Real, Large Distributed Systems*, 2005.

[16] G. Lee and R. H. Katz. Heterogeneity-aware resource allocation and scheduling in the cloud. In *Proc. USENIX HotCloud*, 2011.

[17] D. A. Menascé. Response-time analysis of composite web services. *IEEE Internet Computing*, 8(1):90–92, 2004.

[18] D. A. Menascé and S. Bardhan. Queuing network models to predict the completion time of the map phase of mapreduce jobs. In *38. International Computer Measurement Group Conference, Las Vegas, NV, USA, December 3-7, 2012*, 2012.

[19] D. A. Menasce and P. Ngo. Understanding cloud computing: Experimentation and capacity planning. In *Proc. International Computer Measurement Group Conference*, 2009.

[20] M. Mesnier, M. Wachs, B. Salmon, and G. R. Ganger. Relative fitness models for storage. *SIGMETRICS Perform. Eval. Rev.*, 33(4), Mar. 2006.

[21] N. Mi, G. Casale, L. Cherkasova, and E. Smirni. Sizing multi-tier systems with temporal dependence: benchmarks and analytic models. *J. Internet Services and Applications*, 1(2):117–134, 2010.

[22] MySQL. https://www.mysql.com/, [last accessed October 2016].

[23] Redis IO. http://redis.io/, [last accessed October 2016].

[24] C. Reiss, A. Tumanov, G. R. Ganger, R. H. Katz, and M. A. Kozuch. Heterogeneity and dynamicity of clouds at scale: Google trace analysis. In *Proc. ACM SOCC*, 2012.

[25] B. Schroeder, A. Wierman, and M. Harchol-Balter. Open versus closed: A cautionary tale. In *Proc. USENIX NSDI*, 2006.

[26] C. R. Shaliz. *Advanced Data Analysis from an Elementary Point of View*. 2015. http://www.stat.cmu.edu/~cshalizi/ADAfaEPoV/ADAfaEPoV.pdf, [last accessed October 2016].

[27] D. Shen and J. L. Hellerstein. Predictive models for proactive network management: Application to a production web server. In *Proc. NOMS*, 2000.

[28] M. S. Squillante, D. D. Yao, and L. Zhang. Web traffic modeling and web server performance analysis. *SIGMETRICS Perform. Eval. Rev.*, 27(3), Dec. 1999.

[29] C. Stewart, T. Kelly, and A. Zhang. Exploiting nonstationarity for performance prediction. In *Proc. ACM SIGOPS EuroSys*, 2007.

[30] E. Thereska and G. R. Ganger. Ironmodel: Robust performance models in the wild. In *Proc. ACM SIGMETRICS*, 2008.

[31] B. Urgaonkar, G. Pacifici, P. Shenoy, M. Spreitzer, and A. Tantawi. An analytical model for multi-tier internet services and its applications. In *Proc. ACM SIGMETRICS*, 2005.

[32] Wang, C. et al. Recouping energy costs from cloud tenants: Tenant demand response aware pricing design. In *Proc. ACM e-Energy*, 2015.

[33] Microsoft Azure: Virtual Machines Pricing. https://azure.microsoft.com/en-us/pricing/details/virtual-machines/, [last accessed October 2016].

[34] Z. Zhang, L. Cherkasova, and B. T. Loo. Parameterizable benchmarking framework for designing a mapreduce performance model. *Concurrency and Computation: Practice and Experience*, 26(12), 2014.

 ISBN: 978-1-61208-529-6

[35] Z. Zhang, L. Cherkasova, and B. T. Loo. Exploiting cloud heterogeneity to optimize performance and cost of mapreduce processing. *SIGMETRICS Perform. Eval. Rev.*, 42(4), June 2015.

 ISBN: 978-1-61208-529-6