Towards Building Performance Models for Data-intensive Workloads in Public Clouds

Rizwan Mian, Patrick Martin,
Farhana Zulkernine
School of Computing
Queen's University
Kingston, Ontario, Canada, K7L3N6
{mian, martin, farhana}@cs.queensu.ca

Jose Luis Vazquez-Poletti
Departamento de Arquitectura de Computadores y Automatica
Universidad Complutense de Madrid
28040. Madrid, Spain
jlvazquez@fdi.ucm

ABSTRACT

The cloud computing paradigm provides the "illusion" of infinite resources and, therefore, becomes a promising candidate for large-scale data-intensive computing. In this paper, we explore experiment-driven performance models for data-intensive workloads executing in an infrastructure-as-a-service (IaaS) public cloud. The performance models help in predicting the workload behaviour, and serve as a key component of a larger framework for resource provisioning in the cloud. We determine a suitable prediction technique after comparing popular regression methods. We also enumerate the variables that impact variance in the workload performance in a public cloud. Finally, we build a performance model for a multi-tenant data service in the Amazon cloud. We find that a linear classifier is sufficient in most cases. On a few occasions, a linear classifier is unsuitable and non-linear modeling is required, which is time consuming. Consequently, we recommend that a linear classifier be used in training the performance model in the first instance. If the resulting model is unsatisfactory, then non-linear modeling can be carried out in the next step.

Categories and Subject Descriptors

C.4 [**Performance of Systems**]: Modeling techniques; H.2.4 [**Database Management**]: Systems – *query processing, transaction processing, concurrency*.

Keywords

Performance Model, Performance Variables, Prediction Techniques, Multi-Tenancy.

1. INTRODUCTION

The pay-as-you-go flexibility and the absence of an up-front commitment in clouds are attractive to new businesses or companies seeking to lower their operational costs or wishing to experiment with new applications. Force.com, for example, currently supports over 55,000 organizations by hosting their applications and data on shared resources as tenants [35]. Tenants operate in virtual isolation from one another. The organizations are allowed controlled access to their objects. Further, they can use and customize their tenants, while the data and customizations remain secure and insulated from other tenants. Each tenant can service multiple workloads, and a workload can access multiple tenants with appropriate permissions. In our case, the tenants exist in a public cloud, which offers its infrastructure and computing resources to the general public over the internet.

Predicting the workload behaviour sets the expectations for execution time and cost. In particular, this enables more reliable guarantees towards service level agreements (SLA) prior to any workload execution. Knowing the workload behaviour a priori is also very useful for many administrative tasks such as capacity planning, admission control and effective scheduling of workloads.

We see a performance model as a key component when predicting the cost of workload execution in the clouds. We propose a framework [18] that uses search algorithms, and associated performance and cost models to minimize the cost of executing workloads in a cloud. A search algorithm in our framework hunts for a suitable resource configuration given a workload and an objective such as minimal dollar-cost. It employs a cost model to estimate the expense of a resource configuration, which in turn uses a performance model to predict the workload behaviour on the resource configuration.

In our previous work, we used a Queuing Network Model (QNM) as a performance model [19]. We found that the response times for queries on a Virtual Machine (VM) as predicted by simple single server centre models, varied by as much as 70% from the measured response times. A simple model does not capture the impact on the workload performance of the interactions among different query types. Developing a more detailed QNM for a VM is not feasible because of the difficulties in acquiring detailed performance parameters in a public cloud environment. Further, high performance variability in clouds poses a great challenge for database applications in guaranteeing SLAs [25].

In this paper, we explore an experiment-driven approach for creating performance models for data-centric workloads on Infrastructure-as-a-Service (IaaS) public clouds such as the Amazon EC2 [7], which appeared around 2006. These clouds pose unique challenges due to multi-tenancy, heterogeneity of the virtual machine types, non-linear changes in resources and the interplay among mixed workloads. Serving multiple tenants increases: (a) competition among different tenants over the shared resources (like memory), and (b) interference amongst the concurrently executing requests. We believe that our experimental-based approach is particularly suited to the variability of cloud environments. Our performance models predict throughputs for transactions, and response times for queries. We enumerate the variables that impact variance, but limit the scope of our performance models by accounting for a subset of these variables. Furthermore, we identify different data patterns in the measurements.

ICPE'13, April 21-24, 2013, Prague, Czech Republic.

We also compare different prediction techniques for their suitability as a base classifier, and choose the most promising one. We find that *linear classifiers* are suitable for most request types and are fast to build and validate. They require less involvement on a developer's part and can often be employed straight out-of-the-box with default parameters in a commonly used machine learning toolkit such as Weka [13]. However, the results are unsatisfactory where there are non-linear trends in the performance data. In such cases, we explore *non-linear modeling* methods, which require choosing a suitable kernel and searching for appropriate parameter values. The search time can be in the order of hours, which becomes prohibitive for large number of request types. As a result, we recommend that a linear classifier be used in training the performance model in the first instance. If the resulting model is unsatisfactory, then the non-linear modeling can be carried out as the next step. In both cases, the models obtained can be *serialized* into java *bytestream*, and used by other applications [1].

The paper is structured as follows. We survey related work in Section 2. The performance variables are discussed in Section 3. We customize the typical experiment-driven model building approach, determine the suitable number of samples to build a representative performance model, and compare different prediction techniques in Section 4. Section 5 contains evaluation of the performance model. We choose diverse request types from three tenant databases, and justify our choice; identify different data patterns observed during the construction of the performance models; and present the validation results with a linear classifier. Section 6 discusses non-linear modeling. We discuss the evaluation results in Section 7. Finally, we present the summary and conclusions in Section 8.

2. RELATED WORK

Analytical performance models have enjoyed great popularity in the database management systems (DBMSs) area. Weikum et al. [34] provide a survey of the advances in autonomic tuning in database technology. They conclude that self-tuning should be based on a feedback control loop and should use mathematical models with proper deployment into the system components. Abouzour et al. [2] use analytical modeling to set the multi-programming level (mpl) of a DBMS for improved throughput. Analytical models have also been used for answering "what-if" questions to study the effects of system changes, such as system upgrades and service migrations [27]. Analytical models, however, are hard to evolve with the underlying system and make simplifying assumptions that make them oblivious to the interactions of the dynamically changing workloads and their effects [26]. These effects are amplified by the variance in the cloud [19]. Therefore, there is increasing interest in experiment-driven machine learning and statistical modeling.

Ganapathi et al. [11] predict multiple performance metrics for individual query types with less than 20% error for 85% of the test cases. Their work however, focuses on single query types and ignore interactions and query mixes. Gupta et al. [12] study the problem of predicting the execution time of a query on a data warehouse with a dynamically changing workload. They use a machine learning approach that takes the query plan, combines it with the observed load vector of the system and uses the new vector to predict the execution time of the query.

Courtious et al. [10] propose a prediction technique, called *regression splines*, that builds a non-linear regression function piecewise using a set of linear functions. In addition, they automate building of a regression function up to a desired accuracy by performing progressive sampling and experimentation. They evaluate their work by predicting CPU demands of an event based server. We find their idea of automation useful but consider standard predictive models available in Weka in our case.

Some recent papers view the *transaction mix* as the combination of different transactions that execute during a time interval window without considering which of these transactions execute simultaneously. This is fundamentally different to our notion of a *concurrent* request mix, where request instances execute simultaneously at any given time. Transaction mix models have been used for capacity planning [40], workload management [39], preempting congestion [38] and detecting anomalies in performance [17].

Much of the above work does not consider interactions between the concurrently executing requests, which can have a significant impact on database system performance [3]. Ahmad et al. [4] develop an interaction-aware query scheduler that targets report-generation workloads in Business Intelligence (BI) settings. Under certain assumptions, the schedule found by this scheduler is within a constant factor of optimal, and consistently outperforms conventional schedulers that do not account for query interactions.

Ahmad et al. [5] use a combination of an offline statistical model trained on sample query mixes and an online interaction-aware simulator to estimate workload completion times. No prior assumptions are made about the internal workings of the database system or the cause of query interactions, making the models robust and portable.

Tozer et al. [28] use a linear regression response time model for throttling long running queries. A performance model built using linear regression is unable to model non-linear trends in the response times of a query. Sheikh et al. [26] propose performance modeling based on Gaussian Processes, which can model non-linear trends, update online and reuse prior knowledge.

The performance models used in the above literature are typically built for workloads accessing a single data tenant. Further, the performance models usually provide predictions for response time only, and are validated on a local server or a local VM. In contrast, our performance model predicts both throughput and response times for transactional and analytical workloads, and operates over a multi-tenant data-service. We propose usage of different classifiers that vary in their modeling scope and development effort. We believe that this is the first attempt to build such models in a public cloud.

3. VARIABLES IN BUILDING A PERFORMANCE MODEL

Cloud components such as CPU, memory and I/O suffer from high performance unpredictability, especially when compared to a physical machine in a local network [25]. This is a major problem in building a performance model for workload execution, which is used for providing SLAs [19].

Therefore, we discuss some variables that play an important role in performance variance and their possible values in building a performance model. As a rule of thumb, the wider the scope of the performance model, the greater the variance it has to capture, and the lower is its prediction accuracy. The possible combinations of these variables in our experimental environment are large so, for purposes of the presentation, we choose to use combinations that

provide us with modestly generic and accurate performance models. Hence, we acknowledge the variables' presence but explore a subset due to practical reasons.

We view a workload as a set of request types, where each request type has zero or more instances. All request instances are executed concurrently. Therefore, a workload is synonymous with a request mix that executes at a data service at any particular time, and we use them interchangeably. A performance model is trained and validated against a number of samples, where each sample is a different workload or request mix.

We consider the following variables when building a performance model for workloads executing at a data service in a cloud:

1. **Workload and tenant diversity:** The workload for a tenant may be entirely Online Transaction Processing (OLTP), entirely Online Analytical Processing (OLAP) or a mix of the two. The OLAP queries and OLTP transactions place significantly different requirements on a data service. The OLAP queries may take hours to execute, while OLTP transactions usually complete in a sub-second timeframe. In reality, a data service is rarely an analytical data source. Further, multi-tenancy increases the variance in the behaviour of a data service, since the tenants compete for shared resources such as memory. In this paper, we build performance models for *any type* of workload executing against a *multi*-tenant data service.

2. **Execution Platform:** The system attributes, such as number of cores, memory and I/O performance, of the cloud VMs do not vary linearly [6]. Even the servers hosting the same VM type do not necessarily have the same processor type. For example, the *xlarge* VM type in the Amazon cloud is powered by either a Xeon or an Optron processor. This amplifies the variance many folds, which can be about 35% compared to the 0.1% on a local physical machine for the CPU [25].

   Given the non-linearity of the VM types and the heterogeneity of the processors, the workload behaviour is likely to be non-linear across VM types and their host servers. In order to limit the variance, we decide to build models for specific VM and processor types in this paper.

3. **Day of the week:** Schad et al. [25] note that the CPU performance of a VM type also varies by the day of a week. This increases the variance in the training data for the performance model. Daily variance can be explicitly modelled using some additional attributes, say day of the week, or using techniques like time series. Either approach comes at the cost of additional complexity. Further, taking daily measurements is possible but labor-intensive. We want our performance model to be time independent. However, we leave the modelling of daily variance for future work. For now, we overlook the daily variance in building and validating our performance models.

4. **Model specificity:** Building a performance model specific to a subset of workloads reduces the variance in the training data. Consequently, a workload specific performance model is more accurate compared to a model built for any type of workload. However, the building effort is exponential in the number of workloads in the worst-case scenario. This becomes excessively brute force and highly synonymous to a lookup table. Instead, we use stratified sampling over the workload space to give us a modest coverage. This is discussed further in Section 4.1.

5. **Prediction technique:** The performance model can employ a number of regression methods or base classifiers such as linear regression or multi-layer perceptron. These classifiers vary in their ability to capture variance at the cost of training and runtime complexity. The training complexity places a requirement on the number of samples required for a *representative* model. Meanwhile, the runtime complexity can undermine the performance model if it is embarrassingly large. We compare different classifiers in Section 4.5.

4. BUILDING THE PERFORMANCE MODEL

Our approach is a typical experiment-driven performance modeling method customized to clouds. It consists of three stages: (a) sampling the space of possible request types and their instances for a request mix, (b) collecting data by executing possible request mixes or samples, and (c) pre-processing data and building performance models. In addition, we empirically determine a suitable number of samples for building a representative performance model, and we compare different prediction techniques to determine their suitability in our performance model. We believe that the latter two exercises need not be repeated every time a similar performance model is built.

4.1 Sampling the space of request mixes

The possible combinations of request mixes are exponential, so an effective sampling approach is essential. Similar to Tozer et al. [28], we randomly sample the N-dimensional space, where N is the number of request types, using a Latin Hypercube Sampling (LHS) protocol [15]. This protocol significantly reduces the number of experiments needed while providing a *normal* coverage of the possible request mixes. This is because the distribution of all the request instances, R, considering all the samples approximates a normal distribution around the mean value of R. This results in a somewhat narrow distribution of load, which is not desirable for building performance models for widely varying loads. That is why Sheikh et al. [26] perform uniform sampling across two dimensions: 1) total number of queries, and 2) the number of different types of concurrent queries. Nonetheless, we still settle for the LHS protocol to control the diversity in the samples.

4.2 Experiment-driven data collection

Once the samples are obtained, we execute them in a public cloud for each VM type. Both the client and the data-service exist in a public cloud to avoid communication delay over a WAN. We wrap up the tenant databases with the MySQL dbms and the ubuntu linux, and store that as an *image*.[1] This greatly simplifies the engineering process, and the workloads can start execution as soon as the compute and storage resources are available, i.e. when the image is instantiated on a VM. On instantiation, the buffer pool occupies 80% of the total memory of a VM instance, and is partitioned in proportion to the number of tenants. Each sample is executed for some time (say around 10m). The request mix remains constant throughout the execution of the sample. The

[1] Our image (ami-7bc16e12) is publicly available at: http://thecloudmarket.com/owner/966178113014. Once the image is instantiated, the clients can connect (ssh in) to the instance and access the MySQL dbms as root user with wlmgmt password.

client collects run time statistics such as throughput and response times for each request type in a sample. This is the most time-consuming part of the model building process and takes tens of hours to complete. Fortunately, this process need not be repeated for each request type unlike suitable parameter value search for non-linear modeling.

4.3 Constructing the request mix model

After all the samples are executed, we collect the sample execution results from the data repository of the client. We pre-process the raw data before training a performance model. The pre-processing involves (a) adjusting the scale of the units e.g. converting response time from milliseconds to seconds, (b) analyzing the data to identify any data patterns such as non-linear trends, and (c) cleaning the data e.g. removing outliers. Understanding and treating data patterns improves the quality of the performance model (discussed later in detail in Section 5.3). Then we can train a performance model on the pre-processed data. We can use any regression method such as linear regression or multi-layer perceptron (mlp). We compare different regression techniques on the basis of correlation as discussed in Section 4.5. Finally, we validate the performance model against new data (presented in Section 5.4).

4.4 Determining a suitable number of samples

Traditional wisdom says that more samples produce a more *representative* prediction model. Executing samples is an expensive exercise, and therefore, we need some way of determining an appropriate sample size that gives reasonable accuracy and confidence in the performance model. We see two approaches of determining the appropriate number of samples: (a) a theoretical approach such as estimating the number of samples based on the confidence level and interval, or (b) an empirical approach such as using experimentation to determine the appropriate number of samples).

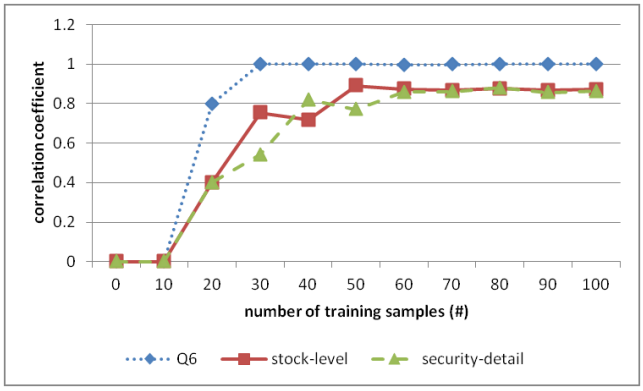

Figure 1: Observing changes in correlation coefficients for multi-variate linear regression (LR) on a hp-xl VM instance against number of training samples (up to 100).

In this paper, we explore the empirical approach further. We choose the High-CPU Extra Large (*hp-xl*) VM type in the Amazon cloud for analyzing the change in the quality of the performance model as the number of samples and their observed metrics (collectively called *training samples*) are increased. For example, we compare the correlation coefficient for three request types (Q6, stock-level and security-detail), each from a different database tenant (TPC-H [32], TPC-C [29] and TPC-E [31] respectively). Q6 is an analytical query, while the other two request types are transactions. The intention is to use diverse request types in our analysis.

We execute about 100 samples obtained using the LHS protocol (experimental setup similar to Section 5.1). We divide the training samples into 10 intervals. Each interval also contains the training samples of its predecessor. We choose multi-variate linear regression (LR) as the base classifier for the performance model. We build and validate the LR models on each interval using 10 folds cross-validation. We plot the correlation values of the LR models against the number of training samples as shown in Fig. 1. We see the greatest gains in accuracy when the numbers of training samples are in the first half of the plot. After that, there are diminishing returns, and the correlation coefficients seem to stabilize.

We observe a similar pattern for a number of other request types. We analyze the effect of the sample size (up to 620) on the correlation values of the same request types as shown in Fig 2. We see the greatest gains in correlation value for about the first 150 samples and limiting gains afterwards. This is similar to the observation of Sheikh et al. [26], who see greatest gains in accuracy for the first 100 samples and diminishing returns subsequently. Based on this analysis, the sample size of a few hundred should provide us with a representative prediction model.

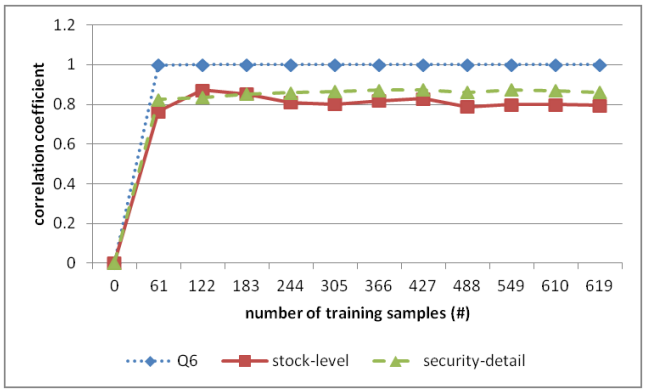

Figure 2: Observing changes in correlation coefficient for multi-variate linear regression (LR) on a hp-xl VM instance against the number of training samples (up to 620).

4.5 Comparison of prediction techniques

Cautious of high variance in the clouds, we consider a number of base classifiers for usage in our performance model. We compare four classification techniques: multi-variate linear-regression (LR), Gaussian Processes (GP), Multi-Layer Perceptrons (mlp) and Support Vector Machine (SVM). We consider two implementations of SVM (nuR [9] and SMO [22]). Tozer et al. [28] find LR to be sufficient as a base classifier for their performance model. LR is a simple regression model, and serves as a baseline in our case. Sheikh et al. find GP to be particularly accurate and adaptive to unseen request types, so we include that in our analysis. SVM has seen major development and fame in the last few years due to its robustness and transparency. Compared to other classifiers, mlp is a "black-box" type model and serves as an alternate comparison point.

GP and SVM can employ various kernel functions. The kernel functions used in SVM rearrange the original training samples into a high dimensional space using a set of mathematical functions. The motivation is to better identify the boundaries between the training samples. The trade-off is higher prediction accuracy at the cost of increased computational complexity and the risk of over-fitting.

We consider two kernel functions: (a) linear, and (b) Radial Basis Function (RBF) for both GP and SVM. Linear kernel is similar to an identify function, while Radial Basis Function is a popular kernel. We use both linear and RBF kernels for GP and SMO with their *default parameter values* in Weka. We consider the default parameter values for each classifier to be a good starting point. More importantly, Witten et al. [36] suggest that the Weka Explorer generally chooses sensible default values for parameters. We consider 150 training samples for comparison. We compare the correlation values of the classifiers obtained after 10-folds cross-validation, in the bar chart (as shown in Fig. 3). We group variations of the same technique. Each group is distinguished by a different shade of grey.

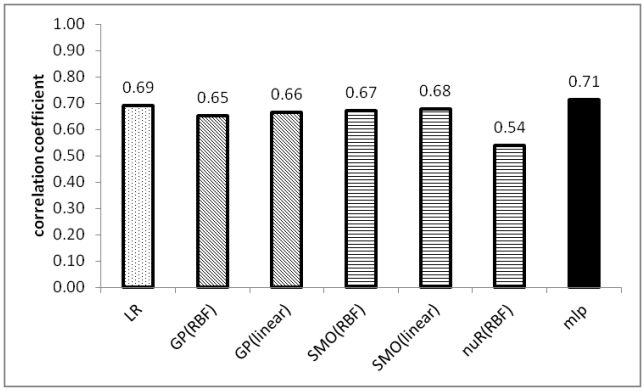

Figure 3: Comparing correlation values for trade-update. About 150 samples have been executed on a *large* VM instance. Variants of the same technique are grouped and distinguished by a different shade of grey.

The high coefficient value of mlp is attractive but possibly misleading. A major drawback of a mlp consisting of any hidden layers is that the hidden neurons are essentially opaque. Secondly, a mlp is prone to over-training. The next candidate is LR, which tends to exaggerate the errors in the general case since it minimizes the predictions' squared errors instead of absolute errors, which is the case with SMO with a linear kernel i.e. SMO (linear). Like LR, the basic idea of SMO (linear) is to find a function that approximates the training points well by minimizing the predictions' absolute errors [36]. The crucial difference is that all deviations up to a user-specified parameter are ignored. We feel that SMO (linear) is an upgrade of LR. It uses SVM constructs while minimizing over-fitting right from the outset. Linear classifiers are preferable over non-linear classifiers, because the latter usually require specification of additional parameter values and are prone to over-fitting. Incorrect parameter values lead to poor correlation values, and we see that in the case of nuR(RBF).

GP and its variants have been used for performance modelling [5; 26], and the accuracy of a GP can be further improved by using an appropriate kernel and parameter values. A GP is defined by a mean function and a co-variance function [24]. The co-variance function itself can have some parameters called hyper-parameters. Sheikh et al. [26] develop a configuration model to generate hyper-parameter values, which enables fast learning of unknown configurations. This is relevant for various types of unseen VMs and/or workloads, where prediction models have not been trained previously. Unfortunately, the GP implementation in Weka does not allow tuning of hyper-parameter values.

As we show in Section 5.4, SMO (linear) suffices as the base classifier in modeling performance for many request and VM types. However, it is unsuitable for modeling non-linear trends in data, and we use non-linear modeling in such cases. Standard SVM training has $O(n^3)$ time and $O(n^2)$ space complexities, where n is the training set size [33]. Platt's original Sequential Minimal Optimization (SMO) is linear in the amount of memory required for the training set size, and the training time of SMO empirically scales between O(n) and $O(n^{2.2})$ on various test cases [22].

5. EVALUATION

5.1 Experimental setup and validation method

The LHS protocol draws a specified number of samples from a multi-dimensional space given the lower and the upper bounds of each dimension (i.e. a request type). The minimum number of instances of a request type in a sample mix is 0. Meanwhile, we set the upper bound to be the optimal mpl value on a VM type (e.g. 14 for *small* VM type for the request type considered). We determine the optimal mpl value for each VM type experimentally.

We use the LHS protocol to generate two sets of samples with different random seeds. We consider a larger set (150 samples) for training and a smaller set (100 samples) for validation to be appropriate. We execute both sets in the Amazon cloud using separate VMs and clients. We execute each sample at the data service twice. The first round of sample execution is for warm-up, and the second round is for taking throughput and response time measurements. We employ SMO (linear) for learning. Then, we validate the performance model against the test set.

We use popular metrics from the literature for comparison, namely correlation coefficient [28] and mean prediction %errors [26][1]. Correlation quantifies the similarity between the actual and modeled trends – they may be far apart, and yet we can have excellent correlation. Meanwhile, prediction errors quantify the gap between the predicted and the measured values. Correlation coefficient and prediction accuracy are complementary, and we use both.

Table 1: Specifications of the VM types considered in the Amazon cloud

VM Type	Cores (#)	Memory (gb)	Cost/hr($)	Optimal mpl
Small	1	1.7	0.065	14
Large	2	7.5	0.260	75
Xlarge	4	15	0.520	115

High correlation coefficients (around 0.80 or above) and low prediction errors (around 20% or below) indicate the success of our performance model. We set these boundaries based on the existing literature [11; 26; 28]. The ideal value of a correlation coefficient is one, while the ideal value of mean-%error is zero.

We consider three heterogeneous VM types for evaluation: (a) *small*, (b) *large*, and (c) *xlarge*. They vary in price, processing power and capacity to hold data in memory. Further, xlarge VM type has either Xeon or Optron processor, and we build our

[1] Percentage-error (%error) = |measured value – predicted value|/measured value

models for Xeon processor only. The data service is configured to occupy most of the available memory on a VM type. The specifications of the VM types we used are stated in Table 1.

5.2 Tenant databases and selection of request types

We use databases in well-known benchmarks as tenant databases in evaluating our work. We consider databases of two transactional benchmarks (TPC-C [29] and TPC-E [31]), and the database of an analytical benchmark (TPC-H [32]). Our workloads consist of a mix of queries and transactions from the stated benchmarks (as shown in Table 2).

Table 2. Selected queries and transactions from the standard benchmarks

Benchmark	Request types
TPC-H (OLAP)	Q1, Q6, Q12, Q21
TPC-C (OLTP)	new-order, payment
TPC-E (OLTP)	trade-order, trade-update

A request type in a workload may have multiple request instances that execute concurrently. Our workloads are bound by time. Until then, a request instance is continuously re-submitted if it finishes early. This ensures that the request mix remains constant at a data service throughout the time bound or an experimental run.

We need *data-intensive* request types, which spend *significant* part of their execution time accessing (reading and/or writing) data. We also want diverse request types that place different requirements on the data service. Hence, we use both queries and (read/write) transactions in our workloads. Many request types cannot execute independently. For example, a payment transaction in the TPC-C benchmark assumes the presence of an unpaid order. In the absence of the unpaid order, the payment transaction fails and no change is made to the database. The payment failure is the exception rather than the norm, which also makes intuitive sense since the customer is not charged twice for the same order.

We execute the samples over tens of hours, which results in millions of transactions being executed. We want to avoid millions of failed transactions, or worse, a mix of successful and failed transactions. The latter skew the execution results, such as throughput. Avoiding such transaction failures places a constraint on the selection of our workload. That is, the request type is either independent, or at most dependent on the other transactions in the workload.

We select a subset of queries and transactions from the transactional and analytical benchmarks. We briefly describe each benchmark, and the role of the requests chosen from them. TPC-C models the principal activities (transactions) of an order-entry environment. These transactions include entering and delivering orders, recording payments, checking the status of orders, and monitoring the level of stocks at the warehouses. The payment transactions depend on the results of the new-order transactions, while the new-order transactions can execute independently. The pair of a new-order and a payment transaction can execute independently of the other transactions in the TPC-C benchmark. Therefore, we choose both of them for our workloads.

TPC-E models the activity of a brokerage firm that must manage customer accounts, execute customer trade-orders, and manage interactions of customers with financial markets. The TPC-E benchmark has many "read-only" transactions, but only four "read/write" transactions. Amongst the read/write transactions, the trade-order can execute independently, while the trade-update depends on the results of the trade-order transaction. The trade-order transactions represent buying or selling a security, and the trade-update transactions enable minor corrections or updates to a set of trades. We select them as candidate request types for our workloads.

TPC-H is a decision support benchmark. It consists of a suite of business oriented ad-hoc queries and concurrent data modifications. We experimentally profile data access of all the TPC-H queries and calculate the ratios of *data access* to *execution time* for all the queries. We choose the top four queries (Q1, Q6, Q12, Q21) according to the values of ratios ordered decreasingly. Q1 reports the amount of business that was billed, shipped, and returned. Q6 quantifies the increase in revenue because of the elimination of some discounts. Q12 determines whether cheaper transportation is adversely affecting the priority of orders. Q21 identifies suppliers whose shipments are late.

5.3 Data patterns: Identification and treatment

Our initial attempts at building a performance model with SMO (linear) were met with poor results. We discovered different patterns in the data upon investigation (as shown in Fig. 4).

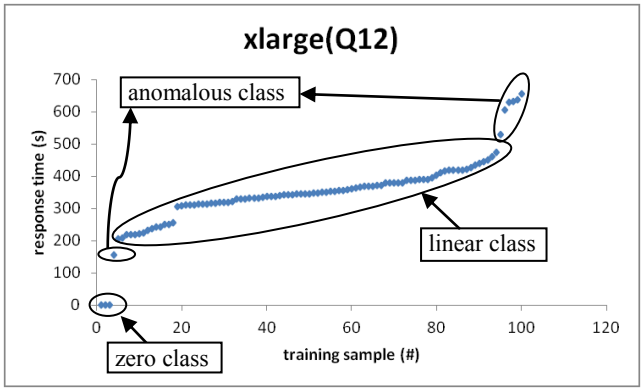

Figure 4: Different data classes in the response time measurements of Q12 on xlarge VM type.

Understanding patterns helps us to improve the quality of the performance models. For example, it allows us to distinguish between outliers and non-linear trends. Removing outliers can sometime improve the mean-%errors significantly. Non-linear trends may require data transformations and place advanced modeling requirements on the performance model.

We identify patterns in the execution results or the data, and the reasons for their existence. In hindsight, they uncover interactions among request types and their impacts on the performance metrics. For example, a large number of concurrent trade-update instances in a request mix reduce the trade-order throughput significantly due to locks on the shared tables and frequent accesses to disks. We describe the data pattern classes below, and their possible treatments before training a SVM.

5.3.1 Data classes

We categorize the data into four general classes: (a) zero, (b) linear, (c) anomalous, and (d) alternate. We also describe the heuristics for identifying each class. In this process, we use the Inter-Quartile Range (IQR) filter to identify any obvious oddities.

Zero class: The zero class represents the samples that have no instances of a request type (r) in the request mix, hence zero metric value for r (as shown in Fig. 4). This is normal behaviour, and we leave them in the data set. Unfortunately, we see some non-zero prediction values for the samples in the zero class. A simple adjustment to the performance model can fix this glitch. In this fix (called zero-fix), the performance model provides zero metric values for the samples where the request instance value is zero in the request mix. This fix considerably improves the correlation relationship between the predicted and measured metric values.

Linear class: The members of the linear class represent a near-linear change in the throughput or response time when sorted by the measured metric value in ascending order (as shown in Fig. 4). Naturally, the linear classifier performs best when the data set is mostly comprised of training samples in the linear class.

Anomalous class: The anomalous class represents training samples with an unusually high or unusually low value of a metric (as shown in Fig. 4). They are *few* in number, say around 2% or less of the entire data set. For example, the training samples in the anomalous class for Q1 exist with unusually low response times due to the smaller number of concurrent Q1 query instances in the request mix in the case of xlarge VM type (discussed in Section 5.4.3). We leave them in the data set unless they skew the validation results *significantly*. The reason for the divergence is sometimes unknown.

Alternate class: If there are *many* training samples with unusually high or low metric values not following the linear trend, then they belong to the alternate class. This is the normal behaviour and the alternate class training samples are left in the data set. The unusual values for the metrics stand out from the values in the linear class due to non-linear trends. For example, about 23% of the training data for the trade-order transaction on the large VM instance consists of unusually high but legitimate metric values. The throughput for trade-order decreases exponentially (non-linearly) with increasing number of concurrent trade-update instances in the request mix. Catering to the combination of alternate and linear classes requires non-linear modeling.

We collectively call all classes, except linear, the *irregular class*. We explored if existing clustering methods can give us the above (or different) classes. Wu et al. [37] put k-means in the second place in the top 10 algorithms in data mining. Unfortunately, we find that k-means and its extension x-means [21], are unable to differentiate the subtle boundaries between the classes. Raatikainen et al. [23] also find the workload classes obtained by k-means to be unsuitable. We want to classify the patterns based on the reasons for their existence, which is something not obvious to a clustering method. For example, members of *zero* and *alternate* classes can have zero and unusually small values respectively but exist due to different reasons. X-means, however, treats them as a single cluster. Therefore, we leave the exploration of other clustering methods for another venue, and currently identify classes manually. Fortunately, the IQR filter helps us *partially* in this job. We use an offset (of 1.5) on IQR over the entire data set to find the training samples that standout from the rest of the training samples.

5.4 Validation results

As mentioned in Section 4.5, we use SMO with a linear kernel as our base classifier. We build our performance model with the training set, and validate against the test set. We see that the linear classifier meets the validation criteria for most request types but performs poorly with non-linear trends. We deal with non-linear trends in Section 6.

We analyze the validation results[1] of the VM types in the following order: *large*, *small* and *xlarge*. This is because only two tenants fit in the memory of the large VM type, and it represents a "middle" case. All tenants fit in the memory of the xlarge VM type, and none fit in the memory of the small VM type. We discussed our choice for different request types in Section 5.2. Every request type needs a separate SVM. We build response time SVMs for the queries Q1, Q6, Q12, and Q21, and throughput SVMs for the transactions i.e., new-order, payment, trade-order, and trade-update.

5.4.1 Large VM type (optimal mpl=75)

We evaluate the SVMs of the queries first. We plot the frequency histogram of observed response times of Q1 after excluding members of the zero class (as shown in Fig. 5). We see that the distribution is normal-like. The frequency histograms for the remaining queries are similar to that in Fig. 5.

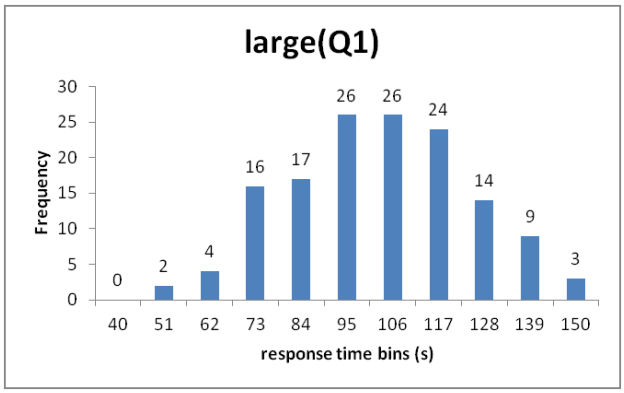

Figure 5: Frequency histogram for observed response times of Q1 on the large VM type instance.

The evaluation metrics are reported in Table 3. The SVMs for the queries have near-ideal correlation coefficients and single digit mean-%errors. We attribute the errors to a few anomalies which exist with unusually low response times due to the smaller number of concurrent queries in the request mix.

Table 3: Evaluation metrics for the response time SVMs built for the large VM type

	Q1	Q6	Q12	Q21
Correlation coefficient	0.99	1	0.96	0.97
mean-%error	3.86	2.71	9.28	5.11

For example, unusually low response times are observed for Q12 as there is a smaller number of concurrent Q12 queries in the request mix compared to the request mixes belonging to other classes. This situation apparently leads to less load on the data service and hence lower response times for Q12.

[1] The raw data used to build the performance model is present at: http://research.cs.queensu.ca/home/mian/index_files/Page485.htm

We plot the correlation relationship between the predicted and actual response times for Q1 and Q12 in Fig. 6 and Fig. 7, respectively.

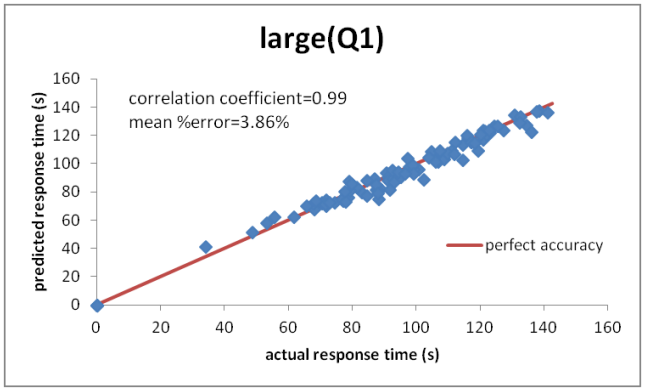

Figure 6: Predicted vs. measured response times for Q1 on a large VM type.

The correlation plots for the remaining queries are similar. The SVMs for all queries have excellent correlation coefficients (close to 1), and low mean-%errors. Therefore, they all meet the validation requirement.

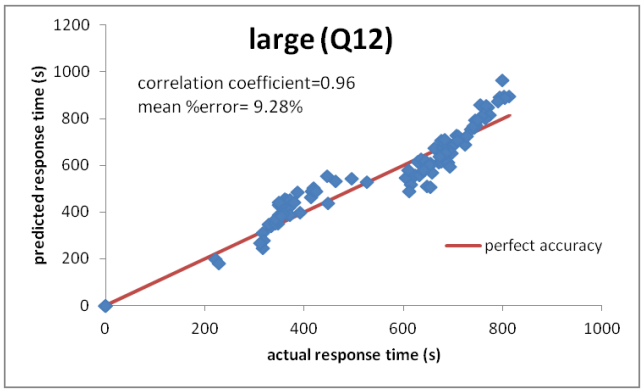

Figure 7: Predicted vs. measured response time for Q12 on a large VM type.

Next, we evaluate the SVMs for transactions. We state the evaluation metrics for the throughput SVMs in Table 4. The SVMs for the new-order and payment transactions have excellent correlation coefficients and acceptable mean-%errors. Therefore, they pass the validation test. The mean-%errors are generally higher than their query counter-part. The irregular class for new-order and payment mostly contains members of zero class and a few anomalies. The anomalies consist of unusually high throughputs, which are caused by a high number of concurrent new-order and payment transactions in a request mix.

Table 4: Evaluation metrics for the throughput SVMs built for the large VM type

	New-order	Payment	Trade-order	Trade-update
Correlation coefficient	0.97	0.97	***0.58***	0.80
mean-%error	14.79	14.65	***111.69***	13.61

The irregular class for trade-update also contains mostly members of zero class and a few anomalies. The anomalies have lower numbers of query instances compared to the training samples in the linear class. This suggests that the data service is under-loaded, and is able to execute more transactions. The trade-update has acceptable mean-%error but the correlation coefficient is on the borderline. We plot the frequency histogram of observed throughput for trade-update after excluding members of zero class in Fig. 8. We see that the distribution is positively skewed.

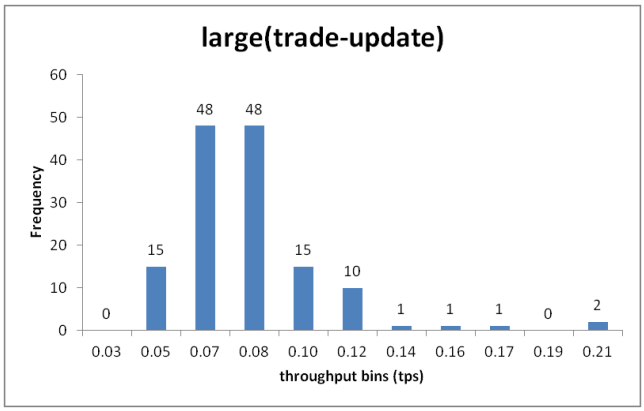

Figure 8: Frequency histogram for observed throughput for trade-update on the large VM type instance.

Regression assumes that variables have normal distributions, and not non-normally distributed variables (highly skewed or kurtotic variables, or variables with substantial outliers) can distort relationships and significance tests [20]. It is possible to employ transformations (e.g. square root, log, or inverse), to the improve normality, but this complicates the interpretation of the results, and should be used in an informed manner. Due to an acceptable mean-%error, we do not perform any data transformation and consider that the trade-update SVM passes the validation test on the border.

Meanwhile, trade-order has a large irregular class, which is mostly comprised of members from the alternate class. For example, we see 34 irregularities in the training set. This is about 23% of the training data, and represents an unusually large proportion. Inspecting the training data for the respective samples, we see that the number of concurrent request instances of trade-update is low in the alternate class. The trade-order and trade-update share some tables in their operations [31]. In addition, the trade-update transaction generates a high disk I/O because it looks for older records that are usually not in the buffer pool due to their age and frequency of access [30].

We suspect that the low trade-order throughput (in the linear class) is due to the high lock contention over the shared tables and frequent access to the disk by trade-update. We perform a simple experiment to confirm this suspicion, in which we set the trade-update instances to zero in all the samples. We see very high throughput values for trade-order, and this confirms our suspicion. The throughput for trade-order decreases exponentially (non-linearly) with increasing number of concurrent trade-update instances in the request mix.

We plot the frequency histogram of observed throughput for trade-order after excluding members of zero class in Fig. 9. We see that the distribution is heavily skewed with a long tail. We find that applying common transformations such as square root, log and inverse do not improve the normality of the measured throughput. In this case, the non-linearity in the data must be

explicitly modeled using a non-linear kernel for SVM, for example. We perform non-linear modeling in Section 6. The trade-order SVM fails the validation test with SMO (linear).

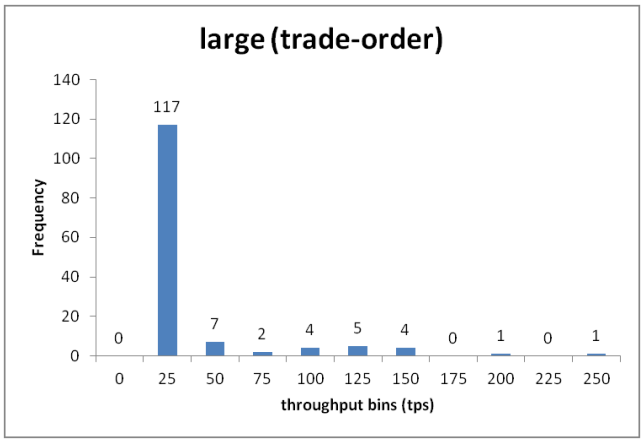

Figure 9: Frequency histogram for observed throughputs for trade-order on the large VM type instance.

5.4.2 Small VM type (optimal mpl=14)

First, we evaluate the SVMs of the queries (see Table 5). The SVMs for Q1, Q12 and Q21 have excellent correlation values and acceptable mean-%errors, consequently, passing the validation test.

Table 5: Evaluation metrics for the response time SVMs built for the small VM type

	Q1	Q6	Q12	Q21
Correlation coefficient	0.90	***0.86***	0.90	0.90
Mean-%error	15.65	***25.43***	13.07	17.56

As highlighted in Table 5, Q6 has a relatively poor correlation value and high mean-%error. We observe three classes in the Q6 data set: (a) zero (28 members), (b) linear (65 members), and (c) alternate (7 members). On average, the alternate class has a higher number of concurrent request instances compared to the linear class. We believe this situation leads to a greater load on the data service and hence higher response times for Q6. The alternate and linear classes represent non-linear trends, and the Q6 SVM does not meet validation requirement due to poor mean-%error.

Next, we evaluate the throughput SVMs for the transactions as shown in Table 6. The SVMs for new-order and payment have excellent correlation coefficients and acceptable mean-%errors. Therefore, they pass the validation test.

Table 6: Evaluation metrics for the throughput performance model built for the small VM type

	New-order	Payment	Trade-order	Trade-update
Correlation coefficient	0.96	0.97	***0.90***	***0.63***
Mean-%error	13.52	9.72	***1465.84***	***34.80***

Interestingly, trade-order's SVM has a high correlation value and an extremely poor mean-%error. Examining the samples (request mixes) in the test set, we see that the resulting throughput values belong to three classes: (a) linear (62 members), (b) zero (29 members), and (c) alternate (10 members). We find that the number of concurrent trade-update transactions is at most two in any request mix. Consequently, the trade-order has high throughput (in tens) for most samples in the test set (i.e. linear class). As noted earlier, providing a performance model for both linear and alternate classes with acceptable evaluation metric values requires modeling of non-linear trends. Therefore, the SVM for trade-order fails validation.

Similarly, trade-update consists of three classes in the test set : (a) linear (56 members), (b) zero (29 members), and (c) alternate (15 members). The respective SVM performs poorly and fails validation.

5.4.3 Xlarge VM type (optimal mpl=115)

First, we evaluate the SVMs of the queries as shown in Table 7. The SVMs for all the queries have excellent correlation values and mean-%errors within the set boundary. Therefore, they all pass the validation test. All SVMs except one (i.e. Q21) have single digit mean-%errors. The training samples for Q21 fall in three classes: (a) linear, (b) zero, and (c) anomalous. The anomalies exist with unusually low response times due to the smaller number of concurrent Q21 queries in the request mix.

Table 7: Evaluation metrics for the response time SVMs built for the xlarge VM type

	Q1	Q6	Q12	Q21
Correlation coefficient	0.99	0.99	0.93	0.93
mean-%error	5.46	3.21	5.86	11.72

Table 8: Evaluation metrics for the throughput SVMs built for the xlarge VM type

	New-order	Payment	Trade-order	Trade-update
Correlation coefficient	0.97	0.97	***0.46***	0.87
mean-%error	18.26	16.68	***94.77***	14.03

Next, we evaluate the throughput SVMs for the transactions as shown in Table 8. The SVMs for new-order and payment have excellent correlation coefficients and acceptable, though high, mean-%errors. They pass the validation requirements. We attribute the high mean-%errors for the new-order SVM to a few anomalies in the new-order test set. The anomalies exist due to a high number of concurrent new-order transactions and a low number of other concurrent request types. The same is true for the payment transaction.

We find a poor correlation coefficient for the SVM of trade-order. Like the data sets of trade-order in the case of large VM type, we see that many training samples belong to the alternate class. The throughput for trade-order decreases exponentially (non-linearly) with an increasing the number of concurrent trade-update instances in the request mix. The trade-order SVM fails the validation test with SMO (linear).

The irregular class for trade-update contains members of two classes: (a) zero, and (b) anomalous. The numbers of trade-order and trade-update request instances in the request mix are similar across anomalous and linear classes. However, the anomalous

class has a lower number of query instances compared to the samples in the linear class. This suggests that the data service is under-loaded, and is able to execute more transactions. This cannot be said conclusively given the small size of the anomalous class (only 3 members in the test set), and we leave it as an observation. We also leave all anomalies in the data sets for trade-update. We still get good correlation value with acceptable mean-%error. Therefore, the trade-update SVM passes the validation test.

6. MODELING NON-LINEAR BEHAVIOUR

We have seen that SMO (linear) is sufficient for many request types, particularly where an alternate trend in the performance is non-existent. We consider a popular (non-linear) RBF Kernel to cater to the combination of linear and alternate trends. We further explore request types that failed their validation test with SMO (linear). We demonstrate the use of the RBK Kernel with the small VM type, but the process is the same for all the VM types. We choose the small VM type for demonstration because the SMO (linear) for the small VM type fails for the highest number of requests amongst all the VM types considered.

Appropriate kernel and parameter settings can greatly improve the SVM classification accuracy. Suitable values for the penalty parameter, C, and the kernel function parameter, γ, are unknown beforehand. For medium-sized problems, the *grid search* approach is an efficient way to find the best C and γ [16]. In grid search, pairs of (C, γ) are tried and the one with the best cross-validation accuracy is chosen.

Before the grid search, we revise our data set to give us more training data out of the existing data set. This is possible using the *0.632 bootstrap* sampling method [14]. We aggregate the training and test data sets to provide us with a combined data set. We remove members of the zero class from the aggregated data set. This is because we can augment the performance model to provide zero values for samples belonging to the zero class, and therefore we do not need to train the performance model to cater for zero class. The revised aggregated dataset of *n* training samples is randomly sampled *n* times, with replacement, to give a *learning* dataset of *n* training samples. Because some elements in the learning dataset are (almost certainly) repeated, there must be some instances in the aggregated dataset that have not been picked by the random selection, which become part of the *validation* set. The size of the validation set is approximately $1/3^{rd}$ of *n*. In this way, we retain the property of unseen training samples for validation similar to the test set. As stated in Section 5.1, training and test samples were generated randomly using different seeds.

Then, we perform a grid search on the learning set using 10-folds cross-validation. In this search, we explore exponentially growing sequences of C (2^{-5}, 2^{-3}, …, 2^{15}) and γ (2^{-15}, 2^{-11}, …, 2^{5}), since Hsu et al. [16] find this to be a practical method to identify good parameter values. They further recommend a coarse grid search first, and then a finer grid search on a "promising" region. Once the search identifies *good* kernel parameters, we train a SMO (RBF) with these parameters using the entire learning set and validate against the validation set. We use this search, train and validate method for Q6.

We find further data transformations for trade-order and trade-update throughput to be appropriate. We discuss the transformation and the justification below. We sort the training samples of the trade-order transaction in the increasing order of throughput value. We plot throughput values against the instance numbers. Despite removing the zero class, the trade-order trend is still fairly non-uniform (as shown in Fig 10). While it is possible to model the trend, there is a large chance of over-fitting to obtain high accuracy given the number of bends and turns required for the curve fitting this trend.

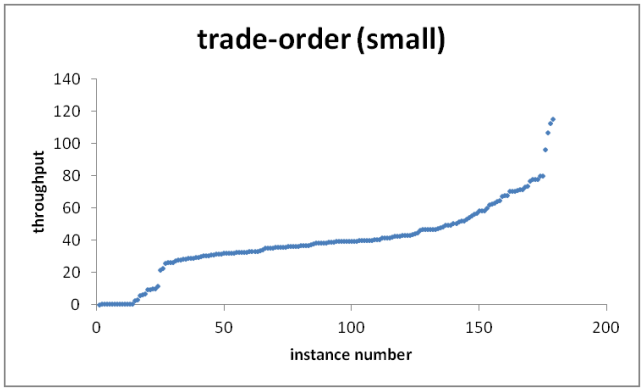

Figure 10: Trade-order throughput values sorted ascendingly.

Instead, the level of symmetry increases significantly if we apply a logarithmic function (as shown in Fig. 11). The bends are near-linear, and there is only one turn. We can train a SVM on the transformed trend without being overly concerned about over-fitting. Therefore, we find the above data transformations for trade-order and trade-update throughput to be appropriate and apply them to the aggregated data set prior to bootstrap sampling.

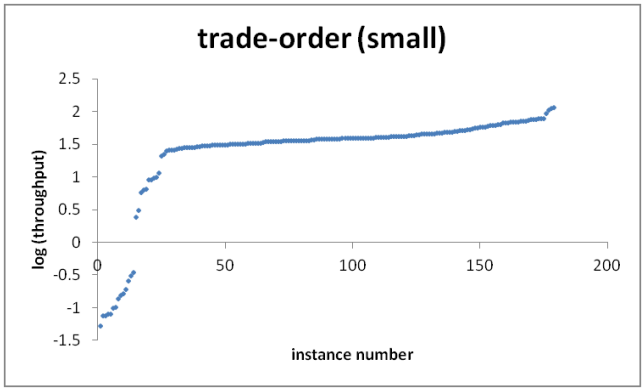

Figure 11: Ascendingly sorted logarithmic trade-order throughput values.

We state the evaluation metrics for the non-linear SVMs trained using the revised training and validation schemes in Table 9. We find that the mean-%errors greatly improve, and the correlation coefficients are excellent for the two request types when we explicitly model the non-linear trends.

Table 9: Evaluation metrics for the non-linear SVMs built for the small VM type

	Q6	Trade-order	Trade-update
Correlation coefficient	0.97	0.98	0.79
mean-%error	8.09	3.43	3.92

We can further improve the *representation* of the real trend in the model for the above request types. Presently, linear and alternate class sizes are not in balance. A good strategy for producing a high accuracy classifier on imbalanced data is to classify any

example as belonging to the majority class, which is called the majority-class classifier [8]. This is what we have done although it may not be very useful in practice. The problem with the success rate is that it assigns equal importance to errors made on examples belonging to the majority class and the minority class. To address the imbalance in the data, we need to assign different penalty parameter values for misclassification to each class.

We further observe that the workload behaviour changes dramatically depending on the amount of trade-update transactions in the request mix. For example, we see different phases in the trade-order throughput (as shown in Fig. 10). We can model each phase separately. This will likely: (a) improve the representation of the real trend in the model, and (b) avoid over-fitting. However, this comes at the cost of identifying phases and boundaries between them. The process of identification may require validation. Presently, we leave class size and phase aware modeling as part of the future work.

7. DISCUSSION

We validate the performance models for the VM types that vary in their system capacity, and in particular their physical memory. As a result, the optimal mpl values as well as the range on the number of request instances vary on these VM types. The behaviour of a request is affected by other concurrently executing requests both in terms of the request types and their number of instances. For example, a smaller number of query instances in the request mix results in less load, and consequently an overall lower response time and for queries high throughput for transactions. We also observed that lock contentions and interactions between concurrently executing requests can have a significant impact on the performance of a database system. This supports the claim by Ahmed et al. [3], that interactions between concurrently executing requests can have a significant impact on the performance of a database system.

We showed that the linear classifier is suitable for 19 out of 24 request types, and can be modeled using an out-of-the-box tool such as Weka. However, it fails where there are non-linear trends in the performance data. In such cases, we explore non-linear modeling methods that require choosing a suitable kernel and a search for appropriate parameter values. Efficient search approaches such as the grid search can take several hours. As a result, we suggest that the linear classifier be used first to train the performance model, and in the case of unsatisfactory results, non-linear modeling be used as the next step.

Osborne et al. [20] state a number of assumptions for multiple regression that the researchers should test. We see that their assumptions (e.g. variables are normally distributed) do not always hold in our case. For example, the throughput distribution for trade-order in case of large VM type is highly skewed. Simple transformations, such as inverse, do not improve the normality of the distribution. Instead, explicit modeling of non-linear trends is required. We find grid search over RBF kernel promising, and see significant improvements in the evaluation metrics.

Our workloads contain at most eight request types each with a different number of instances. This is reasonable since TPC-C and TPC-E benchmarks have five and ten transactions, respectively, although TPC-H has 22 queries. We believe a realistic data service is rarely a read-only or a write-only service. It usually serves a combination of transactional and analytical workloads.

8. SUMMARY

We employ an experiment-driven approach for building a performance model applicable in a cloud environment. The samples are generated using stratified sampling, and measurements are collected by executing these samples in a public cloud. We provide a comparison of different underlying prediction techniques based on accuracy, and justify our choice. Some data patterns are identified and their possible treatments are suggested. Then, we train our performance models using the measured and treated data. The performance models are judged against multiple evaluation metrics, and validated against fresh data. Finally, we analyze the performance models built for different types of Amazon VMs accessible on their public IaaS EC2 cloud.

Recent literature typically builds performance models for a single database tenant, provides response time predictions only, and validates the models on a local server or a local VM. Our performance model predicts throughput for transactions, and response times for queries. The performance model is built for workloads executing at a multi-tenant data-service hosted in a cloud.

Presently, the performance model provides *raw* predictions without expressing any confidence in them. This is an important issue since the errors are cumulative in our framework, and we need some method of managing the errors across the framework components. Therefore, we intend to look at mechanisms which manage the error levels across the framework.

We also intend to make the performance model adapt online, which is particularly relevant for provide predictions for unseen request types and the cloud environment. The magnitude of the errors may be large in the beginning but would reduce over time as the models learns the new setting.

9. ACKNOWLEDGMENTS

The authors acknowledge research support from National Science and Engineering Research Council of Canada (NSERC), MEDIANET (Comunidad de Madrid S2009/TIC-1468), HPCcloud (MICINN TIN2009-07146) and 4CaaSt (European Commission's IST priority of the 7th Framework Programme under contract number 258862). The authors also thank Dr. Skillicorn and Dr. Hussain for their input on non-linear modeling and class identification and validation.

10. REFERENCES

[1] *Use WEKA in your Java code*. http://weka.wikispaces.com/Use+WEKA+in+your+Java+code.

[2] Abouzour, M., Salem, K., and Bumbulis, P., 2010. Automatic tuning of the multiprogramming level in Sybase SQL Anywhere. In *2010 IEEE 26th International Conference on Data Engineering Workshops (ICDEW)*, Long Beach, California, USA, 99-104.

[3] Ahmad, M., Aboulnaga, A., and Babu, S., 2009. Query interactions in database workloads. In *Proceedings of the Second International Workshop on Testing Database Systems* ACM, Providence, Rhode Island, US, 1-6.

[4] Ahmad, M., Aboulnaga, A., Babu, S., and Munagala, K., 2008. Modeling and exploiting query interactions in database systems. In *Proceedings of the 17th ACM conference on Information and knowledge management* ACM, Napa Valley, California, USA, 183-192.

[5] Ahmad, M., Duan, S., Aboulnaga, A., and Babu, S., 2011. Predicting completion times of batch query workloads using interaction-aware models and simulation. In *Proceedings of the*

14th International Conference on Extending Database Technology (EDBT'11) ACM, Uppsala, Sweden, 449-460.
[6] Amazon, *EC2 Instance Types*. http://aws.amazon.com/ec2/instance-types/.
[7] Amazon, *Elastic Compute Cloud (EC2)*. http://aws.amazon.com/ec2/.
[8] Ben-Hur, A. and Weston, J., 2010. A user's guide to support vector machines. *Methods in Molecular Biology 609*, 2, 223-239.
[9] Chang, C.-C. and Lin, C.-J., 2011. LIBSVM: A library for support vector machines. *ACM Transactions on Intelligent Systems and Technology (TIST) 2*, 3, 1-27.
[10] Courtois, M. and Woodside, M., 2000. Using regression splines for software performance analysis. In *Proceedings of the 2nd international workshop on Software and performance* ACM, Ottawa, Ontario, Canada, 105-114.
[11] Ganapathi, A., Kuno, H., Dayal, U., Wiener, J.L., Fox, A., Jordan, M., and Patterson, D., 2009. Predicting Multiple Metrics for Queries: Better Decisions Enabled by Machine Learning. In *IEEE 25th International Conference on Data Engineering, 2009. (ICDE '09).* IEEE, Shanghai, China, 592-603. http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=4812438&tag=1.
[12] Gupta, C., Mehta, A., and Dayal, U., 2008. PQR: Predicting Query Execution Times for Autonomous Workload Management. In *International Conference on Autonomic Computing, 2008. (ICAC '08).* IEEE, Chicago, IL 13-22.
[13] Hall, M., Frank, E., Holmes, G., Pfahringer, B., Reutemann, P., and Witten, I.H., 2009. The WEKA data mining software: An update. *ACM SIGKDD Explorations Newsletter 11*, 1, 10-18.
[14] Han, J., Kamber, M., and Pei, J., 2012. *Data mining: concepts and techniques (Third Edition)*. Morgan Kaufmann.
[15] Hicks, C.R. and Turner Jr, K., 1999. *Fundamental concepts in the design of experiments*. Oxford University Press, New York.
[16] Hsu, C.W., Chang, C.C., and Lin, C.J., 2003. *A practical guide to support vector classification.* National Taiwan University. http://www.csie.ntu.edu.tw/~cjlin/papers/guide/guide.pdf.
[17] Kelly, T., 2005. Detecting performance anomalies in global applications. In *Proceedings of the 2nd conference on Real, Large Distributed Systems - Volume 2* USENIX Association, San Francisco, CA, 42-47.
[18] Mian, R. and Martin, P., 2012. Executing data-intensive workloads in a Cloud. In *CCGrid Doctoral Symposium 2012 in conjuction with 12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid)*, Ottawa, Canada, 758-763.
[19] Mian, R., Martin, P., and Vazquez-Poletti, J.L., 2012. Provisioning data analytic workloads in a cloud. *Future Generation Computer Systems (FGCS)*, in press http://dx.doi.org/10.1016/j.future.2012.1001.1008.
[20] Osborne, J.W. and Waters, E., 2002. Four assumptions of multiple regression that researchers should always test. *Practical Assessment, Research & Evaluation 8*, 2, 1-9.
[21] Pelleg, D. and Moore, A.W., 2000. X-means: Extending K-means with Efficient Estimation of the Number of Clusters. In *Proceedings of the Seventeenth International Conference on Machine Learning* Morgan Kaufmann Publishers Inc., 727-734.
[22] Platt, J., 1998. *Sequential Minimal Optimization (SMO): A fast algorithm for training support vector machines.* Microsoft Research. http://www.bradblock.com/Sequential_Minimal_Optimization_A_Fast_Algorithm_for_Training_Support_Vector_Machine.pdf.
[23] Raatikainen, K.E.E., 1993. Cluster analysis and workload classification. *SIGMETRICS Perform. Eval. Rev. 20*, 4, 24-30.
[24] Rasmussen, C.E. and Williams, C.K.I., 2006. *Gaussian Processes for Machine Learning*. The MIT Press.
[25] Schad, J., Dittrich, J., and Quiane-Ruiz, J.-A., 2010. Runtime measurements in the cloud: observing, analyzing, and reducing variance. *Proc. VLDB Endow. 3*, 1-2, 460-471.
[26] Sheikh, M.B., Minhas, U.F., Khan, O.Z., Aboulnaga, A., Poupart, P., and Taylor, D.J., 2011. A bayesian approach to online performance modeling for database appliances using gaussian models. In *8th ACM international conference on Autonomic computing (ICAC)* ACM, Karlsruhe, Germany, 121-130.
[27] Thereska, E., Narayanan, D., and Ganger, G.R., 2006. Towards self-predicting systems: What if you could ask 'what-if'? *The Knowledge Engineering Review 21*, 03, 261-267.
[28] Tozer, S., Brecht, T., and Aboulnaga, A., 2010. Q-Cop: Avoiding bad query mixes to minimize client timeouts under heavy loads. In *IEEE 26th International Conference on Data Engineering (ICDE)*, Long Beach, CA, USA, 397-408.
[29] TPC-C, *Order Processing Benchmark*. http://www.tpc.org/tpcc/.
[30] TPC-E, *Detailed description*. http://www.tpc.org/tpce/.
[31] TPC-E, *Trading Benchmark*. http://www.tpc.org/tpce/.
[32] TPC-H, *Decision Support Benchmark*. http://www.tpc.org/tpch/.
[33] Tsang, I.W., Kwok, J.T., and Cheung, P.-M., 2005. Core vector machines: Fast SVM training on very large data sets. *Journal of Machine Learning Research 6*, 363-392.
[34] Weikum, G., Moenkeberg, A., Hasse, C., and Zabback, P., 2002. Self-tuning database technology and information services: from wishful thinking to viable engineering. In *Proceedings of the 28th international conference on Very Large Data Bases* VLDB Endowment, Hong Kong, China, 20-31.
[35] Weissman, C.D. and Bobrowski, S., 2009. The design of the force.com multitenant internet application development platform. In *Proceedings of the 35th SIGMOD international conference on Management of data* ACM, Providence, Rhode Island, USA. http://dl.acm.org/citation.cfm?id=1559942.
[36] Witten, I.H., Frank, E., and Hall, M.A., 2011. *Data Mining: Practical machine learning tools and techniques (3rd edition)*. Morgan Kaufmann Pub.
[37] Wu, X., Kumar, V., Ross Quinlan, J., Ghosh, J., Yang, Q., Motoda, H., McLachlan, G., Ng, A., Liu, B., Yu, P., Zhou, Z.-H., Steinbach, M., Hand, D., and Steinberg, D., 2008. Top 10 algorithms in data mining. *Knowledge and Information Systems 14*, 1, 1-37.
[38] Zhang, M., Martin, P., Powley, W., Bird, P., and McDonald, K., 2012. Discovering Indicators for Congestion in DBMSs. In *Proceedings of the International Workshop on Self-Managing Database Systems (SMDB'12) in Conjunction with the International Conference on Data Engineering (ICDE'12)*, Washington, DC, USA, in press.
[39] Zhang, M., Niu, B., Martin, P., Powley, W., Bird, P., and McDonald, K., 2011. Utility Function-based Workload Management for DBMSs. In *Proceedings of the 7th International Conference on Autonomic and Autonomous Systems (ICAS 2011)*, Mestre, Italy, 116-121.
[40] Zhang, Q., Cherkasova, L., Mathews, G., Greene, W., and Smirni, E., 2007. R-Capriccio: A Capacity Planning and Anomaly Detection Tool for Enterprise Services with Live Workloads Middleware 2007. *Lecture Notes in Computer Science 4834*, 244-265.