ActiveSLA: A Profit-Oriented Admission Control Framework for Database-as-a-Service Providers

Pengcheng Xiong*
Georgia Institute of Technology
266 Ferst Drive
Atlanta, GA 30332
xiong@gatech.edu

Yun Chi
NEC Laboratories America
10080 N. Wolfe Rd, SW3-350
Cupertino, CA 95014
ychi@sv.nec-labs.com

Shenghuo Zhu
NEC Laboratories America
10080 N. Wolfe Rd, SW3-350
Cupertino, CA 95014
zsh@sv.nec-labs.com

Junichi Tatemura
NEC Laboratories America
10080 N. Wolfe Rd, SW3-350
Cupertino, CA 95014
tatemura@sv.nec-labs.com

Calton Pu
Georgia Institute of Technology
266 Ferst Drive
Atlanta, GA 30332
calton@gatech.edu

Hakan Hacıgümüş
NEC Laboratories America
10080 N. Wolfe Rd, SW3-350
Cupertino, CA 95014
hakan@sv.nec-labs.com

ABSTRACT

The system overload is a common problem in a Database-as-a-Serice (DaaS) environment because of unpredictable and bursty workloads from various clients. Due to the service delivery nature of DaaS, such system overload usually has direct economic impact on the service provider, who has to pay penalties if the system performance does not meet clients' service level agreements (SLAs). In this paper, we investigate techniques that prevent system overload by using admission control. We propose a profit-oriented admission control framework, called ActiveSLA, for DaaS providers. ActiveSLA is an end-to-end framework that consists of two components. First, a prediction module estimates the probability for a new query to finish the execution before its deadline. Second, based on the predicted probability, a decision module determines whether or not to admit the given query into the database system. The decision is made with the profit optimization objective, where the expected profit is derived from the service level agreements between a service provider and its clients. We present extensive real system experiments with standard database benchmarks, under different traffic patterns, DBMS settings, and SLAs. The results demonstrate that ActiveSLA is able to make admission control decisions that are both more accurate and more profit-effective than several state-of-the-art methods.

Categories and Subject Descriptors

D.4.8 [**OPERATING SYSTEMS**]: Performance—*Measurements,Modeling and prediction*; K.6.0 [**MANAGEMENT OF COMPUTING AND INFORMATION SYSTEMS**]: General—*Economics*

General Terms

Algorithms, Experimentation, Performance

Keywords

Cloud computing, Database-as-a-service, Admission control, Machine learning

1. INTRODUCTION

Database-as-a-service (DaaS) is becoming an increasingly popular model for offering data management functions in a cost-effective way in the Cloud [7, 8, 9, 15]. Compared to traditional database systems, DaaS usually serves more diversified clients (e.g., through multi-tenancy [5]) therefore faces more unpredictable and more bursty workloads. Due to economic considerations, DaaS providers try to avoid resource overprovisioning while accounting for simultaneous peak workloads from a large number of clients. Consolidating multiple clients in shared infrastructures is a very commonly used approach by the DaaS providers. Such a consolidation affords greater economies of scale and fixed cost distribution for the DaaS providers. However, managing system overload becomes a much more crucial problem in such environments.

To prevent system overload, some existing solutions include dynamic provisioning [18, 24, 30, 32], queuing and scheduling [13, 17], and admission control [10, 27]. The following observations can be made for these approaches: (1) By dynamic provisioning, when the system is near the overload zone, more resources such as new nodes or new replicas are added to improve the situation. A major disadvantage of this method is that it involves additional adaptation cost such as data migration and buffer pool warmup [19]. (2) By queuing and scheduling, the incoming queries are held in a queue and then scheduled according to certain prioritization criteria. However, such a solution works only for short-term load peaks— by rearranging the order of query execution, a scheduling algorithm can only *postpone* the unavoidable performance degradation. It is important to note that such a

*The work was done while the author was at NEC Laboratories of America.

SOCC'11, October 27–28, 2011, Cascais, Portugal.

postponement may render the query results useless in many applications (e.g., a Web page returned to the client *after* the client browser's timeout). (3)With admission control, when the system is near an overload condition, new queries are either stalled (e.g., [10]) or rejected (e.g., [27]) until the system condition improves. We focus on admission control for DaaS providers in this paper.

Although existing admission control techniques are helpful to alleviate system overload, they do not work directly towards the main goal of DaaS providers—namely to maximize their profits by satisfying different SLAs for their clients. For example, most traditional admission control techniques mainly focus on the system performance at a macro level, e.g., using a feedback (or dynamic control) mechanism to keep the mean query execution time at a specific level or tuning the best multiple programming level (MPL) for a given workload [1, 23]. There have been recent proposals with a micro level focus, where the workload and the query execution time are considered on a query-by-query basis ([10, 14, 27]). Although these state-of-the-art admission control techniques take the individual query execution time into consideration, they do not directly address two important issues in profit optimization for service providers.

The first issue is that merely estimating the query execution time is not enough to make profit-oriented decisions. In Figure 1 we show the probability density functions (PDFs) of the execution time for two queries, where both queries have the same estimated execution time Est and the same deadline τ (specified by the SLA). Although $Est < \tau$, the chance for the two queries to miss their deadlines are dramatically different. Compared with q_1, about which we are more confident that it will meet the deadline, it is more difficult to tell whether q_2 will meet or miss the deadline. If we use the methodology in the traditional admission control, because Est is less than τ, we will admit both of the queries. However, as we can see from the PDF of q_2, it is much riskier to admit q_2 than to admit q_1. Thus, the probabilities of a query meeting and missing its deadline are crucial in making SLA-based admission control decisions.

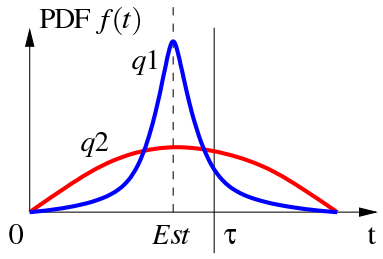

Figure 1: Two queries with the same query time estimation but different query time distributions.

The second issue is that because of SLAs, we may have to make different admission control decisions even when the queries have the same deadline and the same probability of meeting the deadline, as illustrated in the example in Figure 2. Assume that two queries q_1 and q_2 arrive simultaneously and according to certain estimation, for both the queries, the probabilities of meeting and missing their deadline are 60% and 40% respectively, as shown in Figure 2(a). In Figure 2(b) we show two different SLAs that are carried by q_1 and q_2, as described in the two profit profiles. For

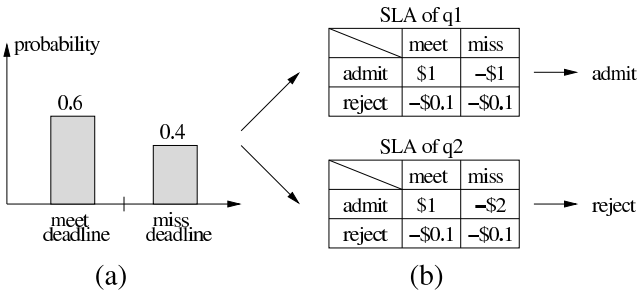

Figure 2: SLA-based admission control decisions.

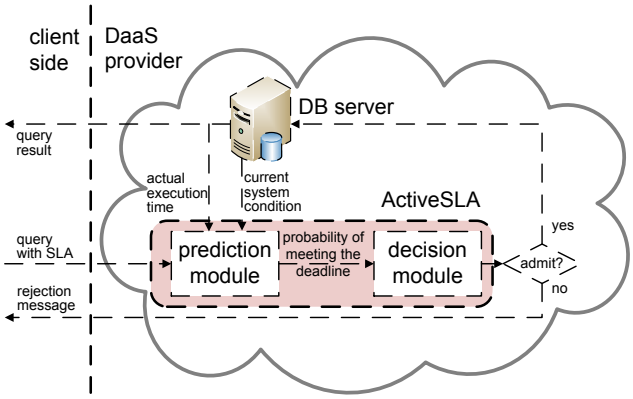

Figure 3: System architecture of ActiveSLA.

q_1, if the query is admitted and meets the deadline, the service provider earns \$1; else if the query is admitted but misses the deadline, the service provider loses \$1; otherwise, if the service provider decides to up-front reject the query, it pays a penalty of \$0.1. For q_2, it is the same as q_1 except that the penalty for missing the deadline is \$2. Simple derivations show that the expected profit for admitting q_1 is $0.6 \times \$1 + 0.4 \times (-\$1) = \$0.2$, which is better than the penalty for rejecting the query ($-\$0.1$). Thus, the service provider should admit q_1. However, the expected profit for admitting q_2 is $0.6 \times \$1 + 0.4 \times (-\$2) = -\$0.2$, which is worse than the penalty for rejecting the query. Thus, the service provider should reject q_2.

In order to address the above two issues, in this paper, we propose a framework, called ActiveSLA[1], for making prediction-based and profit-oriented admission control decisions, with a target of maximizing the expected profit of the DaaS providers. Our ActiveSLA framework is an end-to-end solution that consists of two main modules: a *prediction module* and a *decision module*, as shown in Figure 3 (from the DaaS provider's point of view at the database layer [9]). When a new query arrives, the query first enters the prediction module. The prediction module uses machine learning techniques and considers both the characteristics of the query and the current system conditions. The prediction module outputs the probability for the query to meet its deadline. The calculated probability and the query's SLA are sent to the decision module. The decision module decides either to admit the query or to reject the query up-front. Fi-

[1]ActiveSLA stands for: Admission Control for Profit Improving under Service Level Agreements.

nally, the result of each admitted query is returned to the client and the actual execution time is sent back to the prediction module in real time. This feedback information can further help the prediction module to improve the accuracy of its future predictions by introducing new training data.

We believe the consideration of profit optimization in admission control presents new challenges that we address in our solution. The main contributions of this paper are twofold:

1. We show, by using both theoretical reasoning and empirical validation, how appropriate machine learning techniques can be successfully used to answer a key question of admission control in DaaS: "What is the probability for a query to meet or miss its deadline?" We implement the solution in the prediction module of ActiveSLA. The machine learning techniques (1) take many query related features as well as database system related features into consideration, (2) recognize complex patterns from the data in an automatic way, and (3) provide detailed probabilities for different outcomes.

2. We develop an SLA-based, profit optimization approach in the decision module of ActiveSLA. Decisions are made in a holistic fashion by considering (1) the probability for a new query to meet its deadline under the current system condition, (2) the profit consequences of alternative actions and outcomes, and (3) the potential impact of admitting a just arrived query on the currently running queries as well as on the future queries.

The rest of this paper is organized as follows. We describe the prediction module of ActiveSLA and corresponding experimental studies in Sections 2 and 3, respectively. We describe the decision module of ActiveSLA and corresponding experimental studies in Sections 4 and 5, respectively. In Section 6, we survey related work. Finally, we give conclusion and future directions in Section 7.

2. PREDICTION MODULE

As described in the introduction, a critical parameter for making profit-oriented decisions is the probability for a query to meet its deadline. In this section, we introduce the prediction module of ActiveSLA, which estimates this probability in a real-time fashion.

Our approach for making such a prediction is based on machine learning techniques. Due to their data-driven characteristics, machine learning techniques can automatically recognize complex patterns from the data and provide models with remarkable performance, which is often comparable to that of from domain experts [12]. The algorithms used in this paper are all from the off-the-shelf machine learning package WEKA [16].

However, compared with the frameworks used in previous work (e.g., [10, 12, 27]), our prediction module is addressing a new data management problem in DaaS, namely predicting the probability for a query to meet its deadline. With this new problem in mind, in this section we discuss several design considerations, including (1) the model selection between linear and nonlinear models, between regression and classification models, and (2) the rich set of features used in our model. In the next section, we will provide detailed empirical studies to validate our design choices.

2.1 The Machine Learning Techniques

It is impractical to construct an accurate model that can perfectly predict the execution time of a query because of the numerous factors and their complicated interactions in modern database systems. Such a situation is captured by a well-known quote in the machine learning community, "All models are wrong, but some are useful". In this subsection, we show how we take the specific domain, namely profit-oriented admission control in DaaS, into consideration in order to select a "less wrong" model, which will be fully justified later in the experimental studies.

We compare our models with those of two state-of-the-art methods, namely, TYPE (a version of Gatekeeper [10] implemented by Tozer et al. [27] with load shedding added) and Q-Cop [27].

2.1.1 TYPE and Q-Cop, using Linear Regression

Both TYPE and Q-Cop approaches start by predicting the execution time of a query for each query type. Assuming that there are T query types (i.e., queries that share the same query template), both TYPE and Q-Cop build a model for each query type.

In TYPE, the estimated execution time of a query q_i of type i is $Est_i = e_i \times N + E_i$, where E_i is the query execution time of q_i in a dedicated server, N is the total number of other queries currently running in the system, and e_i is the extra delay that each additional currently running query brings to q_i. Note that TYPE is an *query-type-oblivious* approaches that is based merely on MPL and on the mean response time of queries over all requests.

Compared with TYPE, Q-Cop uses more detailed information. Instead of counting N, Q-Cop considers $\{n_1, \ldots, n_T\}$, i.e., the number of currently running queries of each query type (with $\sum_{j=1}^{T} n_j = N$), which is referred to as the query mix. Based on the query mix, Q-Cop uses a linear regression model to estimate the running time of q_i as $Est_i = (e_{i1} \times n_1) + (e_{i2} \times n_2) + \cdots + (e_{iT} \times n_T) + E_i$. Here e_{ij} is the extra delay that each additional currently running query of type j brings to q_i.

2.1.2 ActiveSLA, using Non-linear Classification

Compared with TYPE and Q-Cop, the prediction module of ActiveSLA uses a non-linear classification model to directly predict the probability of a new query meeting its deadline, as shown in Figure 4.

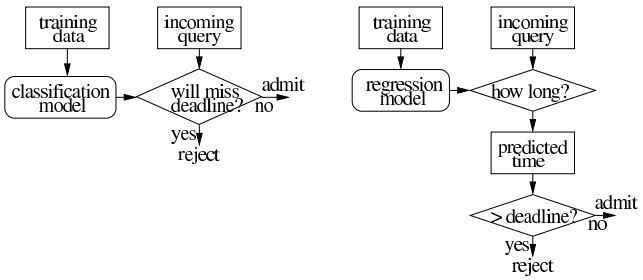

Figure 4: Comparison of machine learning models.

Linear vs. Nonlinear.

Both TYPE and Q-Cop use *linear* regression, which models the relationship between the input features and the out-

put variables by using *linear* functions. However, the execution time of a query depends on many factors in a *non-linear* fashion. In addition, many database settings, e.g., isolation levels and available buffer size, also influence query execution time in a non-linear fashion. Furthermore, it is well known in database and queuing theories that when a system is at a borderline overload condition, a small amount of additional workload will disproportionally degrade the system performance. Such reasoning motivates us to use *non-linear* models.

Regression vs. Classification.

There are two reasons why we prefer a classification model over a regression model.

First, for admission control decisions, we care about the probability for a query to meet its deadline or not rather than the *exact* execution time of the query. From the machine learning point of view, a direct model of classification usually outperforms a two-step approach (i.e., in step 1, regression is used to get an estimation on execution time, with an objective of minimizing the mean square error; and in step 2, this estimated value is compared with the deadline).

Secondly, regression models only give Est_i, i.e., the estimated execution time of q_i. This single point estimation does not contain enough information for us to make profit-aware decisions as illustrated in Section 1. In comparison, the classification model used in ActiveSLA provides us with the probabilities of a query meeting and missing its deadline. This information will be shown crucial for us to make SLA-based admission control decisions.

2.1.3 Non-linear Classification Models in ActiveSLA

The main machine learning technique that we used for non-linear classification is "LogitBoost" algorithm [11]. We also use another algorithm called "Additive Regression" for non-linear regression for a comparison purpose. Both of them are boosting approaches where a set of weak learners (namely, models that may not have exceptionally good performance by themselves but collectively contribute to the final performance) are iteratively learned and combined in order to obtain a strong classifier with good performance. For the weak learners, we use a standard tree model (REP [11]) which partitions the parameter space in a top-down and non-linear fashion. Both of them are implemented in the well-known off-the-shelf WEKA package [16].

2.2 ActiveSLA Features

A key to the accuracy of a machine learning model is the features used to learn the model. In addition to the features used by TYPE and Q-Cop (Section 2.2.1), ActiveSLA exploits a number of additional features from query characteristics (Section 2.2.2) and system conditions (Section 2.2.3).

2.2.1 Query Type and Mix (TYPE, Q-Cop, ActiveSLA)

Both TYPE and Q-Cop use the number of currently running queries as the feature in their model for each query type. For a query q_i of query type i, TYPE uses N, the total number of currently running queries in the system, as the only feature to predict query execution time of q_i. Q-Cop improves over TYPE by splitting N into a set of features $n_1, \ldots, n_T$, which are referred to as the query mix. That is, Q-Cop takes into consideration that different query type (e.g., j) may impact the execution time of q_i in different ways (reflected by e_{ij} in the Q-Cop model).

2.2.2 Query Features (ActiveSLA)

We believe that even for the queries of the same query type, the parameters of a query may affect its execution time, especially when the query contains aggregations or range selections. To extract features related to query execution time, we leverage query optimization techniques, which have been studied for the past decades in both research and practice, by looking into the details of the query plan and query cost estimation from the database system. We take PostgreSQL and MySQL as examples, although the same idea applies to other databases.

In PostgreSQL, the query cost estimation depends mainly on five features, i.e., the number of sequential I/O (seq_page), the number of non-sequential I/O($random\_page$), the number of CPU tuple operations (cpu_tuple), the number of CPU index operations (cpu_index), and the number of CPU operator operations ($cpu\_operator$). We extract these features from PostgreSQL using the light-weighted "explain" command before it executes the query, and we provide a more detailed explanation in the Appendix. Although these parameters are used mainly for PostgreSQL query optimizer to compare the relative costs among different query plans, we show the estimations of these features have strong correlation with the execution time of a query. Thus, we take these five estimations from the query optimizer as a set of features for the ActiveSLA prediction module. MySQL uses similar "explain" command to illustrate how MySQL handles the queries and again, we provide the details in the Appendix.

2.2.3 Database and System Conditions (ActiveSLA)

In addition to studying queries themselves, ActiveSLA also considers the environment in which the queries will be running. More specifically, ActiveSLA monitors the following features from the database server and operating system. We choose these features as they are the most dynamic features that can affect the query execution time. Other more static features like "CPU frequency", "Memory size", "Disk capacity" are related to the DaaS provider's assets and will be added in the future work.

Buffer cache: the fraction of pages of each table that are currently in the database buffer pool and therefore are available without accessing the disk;

System cache: the fraction of pages of each table that are currently in the operating system cache and therefore are available without accessing the disk;

Transaction isolation level: a boolean variable that indicates if the database is currently supporting Read Committed(FALSE) or Serializable(TRUE).

CPU, memory, and disk status: the current status of CPU, memory, and disk in the operating system.

A more detailed description of these features as well as the methods that we used to collect these features are provided in the Appendix. We summarize various approaches in Table 1, in terms of machine learning methods and features used.

Table 1: Comparison of different models.

	Learning model	Learning method	Features used
TYPE	linear	regression	query type
Q-Cop	linear	regression	query type/mix
ActiveSLA-R	non-linear	regression	query type/mix
ActiveSLA-C	non-linear	classification	query type/mix
ActiveSLA	non-linear	classification	query type/mix query features database/system conditions

2.3 The Summary of Models

As mentioned above, ActiveSLA has three unique characteristics: (1) it uses a non-linear learning method, (2) it is based on a classification model, and (3) it includes more comprehensive features. In order to distinguish the contribution of each of these characteristics, we also include two intermediate versions of ActiveSLA, named ActiveSLA-R and ActiveSLA-C. For a fair comparison, both of these intermediate versions use only the same feature as Q-Cop does. Compared to Q-Cop, ActiveSLA-R uses a regression model that is *non-linear*, to estimate the query execution time and then makes comparison with the deadline. Compared with ActiveSLA-R, ActiveSLA-C uses a non-linear *classification* model to directly estimate whether a query will miss the deadline. ActiveSLA-R shows the benefit of non-linear over linear models while ActiveSLA-C shows the benefit of classification over regression. ActiveSLA shows the gain of using more features in the model.

3. PREDICTION MODULE EVALUATION

In this section, we conduct empirical studies to evaluate the design choices that we made in the prediction module of ActiveSLA.

3.1 Experimental Settings

3.1.1 Query Sets

The experiments in this paper are conducted by using the relations in TPC-W benchmark [28]. Based on the TPC-W schema, we use three query sets in our experiments, i.e., TPC-W1, TPC-W2, and TPC-W3.

TPC-W1 *(browsing queries)* TPC-W1 query set follows the same query set that is used in a main baseline approach, i.e., Q-Cop [27]. We use exactly the same query set in order to make a fair comparison. The set includes the query types in the Browsing Mix distribution in TPC-W. The query types are extracted from the corresponding servlets, i.e., Q1(Author Search), Q2(Title Search), Q3(New Products), Q4(Related Products), Q5(Best Sellers Setup) and Q6(Best Sellers Main). This query set follows the Browsing Mix distribution specified by the benchmark. The query mix is skewed in that all the queries other than Q6 can be executed in a very short time (less than 10ms) whereas the execution time of Q6 is around 5 seconds.

TPC-W2 *(mixture of browsing and administrative queries)* For this query set, we replace the short queries in TPC-W1 by three other queries, which are variations of the administrative queries in the TPC-W benchmark, to evaluate the impact of longer running queries in the system.

Q6: select i_id, i_title, a_fname, a_lname from item, author, order_line where item.i_id = order_line.ol_i_id and item.i_a_id = author.a_id and order_line.ol_o_id > (select max(o_id)-3333 from orders) and item.i_subject = ? group by i_id, i_title, a_fname, a_lname order by sum(ol_qty) desc limit 50;

Q7: select c_fname, c_lname, c_email, o_sub_total from customer, orders where c_since > ? and c_id = o_c_id and o_sub_total > ? and o_sub_total < ?;

Q8: select i_title, i_cost, i_desc, o_id, o_status, o_total, ol_id, ol_qty, ol_discount from item, order_line, orders where i_pub_date > ? and i_id = ol_i_id and ol_o_id = o_id and o_total > ? and o_total < ?;

Q9: select co_id, cx_type, ol_discount, avg(ol_qty), avg(o_sub_total) from order_line, orders, cc_xacts, country where ol_o_id = o_id and o_id = cx_o_id and cx_co_id = co_id and ol_discount <= ? and o_total > ? and cx_type like '%?%' group by co_id, cx_type, ol_discount;

TPC-W3 *(mixture of browsing, administrative, and updating queries)* TPC-W1 and TPC-W2 only have read-only queries. We add the following update query Q10 to TPC-W2 to get TPC-W3. Q10 is extracted from servlet "Admin Confirm" in TPC-W benchmark.

Q10: update item set i_cost = ?, i_pub_date = ? i_image = ? where i_id = ?;

For all the query sets, we generate individual queries in the following way. When we want a query from a specific query type(template), we generate a random number according to the database size and use this random number to fill in the "?" part, in order to get a real query.

3.1.2 Workload Generators and System Setting

We design workload generators to provide two sets of workloads. The first set of workloads follow as faithful as possible to those used by Q-Cop [27], i.e., with Poisson arrival and static arrival rates. The second set of workloads are derived from a real trace – the Web trace from the 1998 World Cup site [4], with the dynamic arrival rate scaled proportionally to fit into the experimental environment. Compared with the first set of workload which has a *stationary* arrival rate, the second one has *non-stationary* arrival rates, and so is likely to be closer to the real DaaS scenario. In terms of the ratio of queries of different types, for TPC-W1 we follow that used in [27] (i.e., the same ratio as specified by the benchmark); for TPC-W2 and TPC-W3, we use a uniform distribution to randomly assign the query type to each query. We use PostgreSQL 9.0 as the database server. The total size of TPC-W database is 5.2GB.

We implemented, deployed, and evaluated the system on real machines. The physical machines to run the database server, the client, as well as ActiveSLA all have Intel Xeon E5620 2.4GHz Quad-Core CPU, 16GB of RAM, 1TB 7200rpm disk running Linux with kernel 2.6.18-164.15.1.el5. Machines are connected by Gigabit Ethernet switches. Note that, to further analyze the robustness of our system, we also repeated most of the experiments on lower-end machines with AMD244 1.8GHz Dual-Core, 2GB memory, and a 500GB 7200rpm disk. It turns out that the conclusions were very similar, and therefore we omit them in the paper.

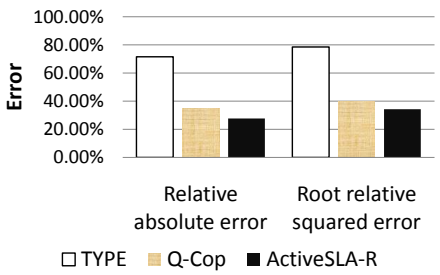

Figure 5: Regression errors for the TPC-W1 query set by different approaches.

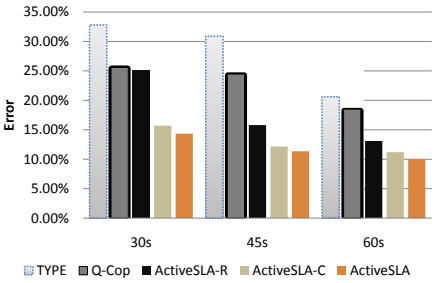

Figure 6: Prediction errors for the TPC-W2 query set by different approaches.

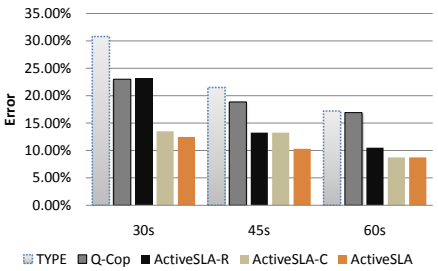

Figure 7: Prediction errors for the TPC-W3 query set by different approaches.

3.2 Result for TPC-W1 Query Set

Since TPC-W1 query set is the only query set used by Q-Cop, in order to make a fair comparison, we set the database settings as close as possible to those described in [27] (e.g., with the 5.2GB data fits in 16GB memory). We also use the same Latin Hyper-cube Sampling (LHS) protocol to sample the space of query mixes. As a result we collect around 12,000 data samples in total. To study the performance, we use the 10-fold cross validation, which is a standard approach in the machine learning area.

Figure 5 shows the comparison of relative absolute error and root relative squared error among TYPE, Q-Cop and ActiveSLA-R for queries of query type Q6 (Best Sellers Main) in terms of query execution time estimation. From the figure we can see that TYPE, which only considers the number of currently running queries, has very large error; Q-Cop, which considers the query mix by using linear regression, reduces the error rate by a half from that of TYPE; ActiveSLA-R, which is a non-linear regression model, further reduces the error[2].

3.3 Result for TPC-W2 Query Set

In this experiment, we reduce the memory size to 3GB so that the data set cannot fit in memory any more. We still use PostgreSQL and set the buffer pool size as 1GB and the effective cache size as 2GB. The four queries in TPC-W2 have comparable execution times, with difference within a factor of 2. We collect around 12,000 data samples in total. In the experiments, we study the cases where the query deadlines are set to 30, 45, and 60 seconds. (As a justification, the default limit time for a PHP script that connects to a database to run is 30 seconds. Safari browser uses a 60-second timeout.)

Because in the previous query set TPC-W1, we have already shown ActiveSLA-R can outperform Q-Cop in terms of regression error, in this experiment, we focus on the error rate of different approaches on predicting whether a query can be finished before its deadline. For this purpose, we use the following metrics.

False positive (N_{fp}): The number of queries that (1) were predicted to be meeting deadline but (2) actually miss deadline.

[2]For short queries in Q1-Q5, since the execution time is usually less than 10ms, the prediction module will predict with high probability that the queries will be finished on time before their long deadlines, e.g., 30s. So in most cases ActiveSLA will admit these queries.

False negative (N_{fn}): The number of queries that (1) were predicted to be missing deadline but (2) actually meet deadline.

Finally, we assume an equal weight between over- and under-prediction and define the prediction *error* as:

$$error = \frac{N_{fp} + N_{fn}}{NT}$$

where NT is the total number of queries in the testing set. For the performance study, we again used 10-fold cross validation.

Figure 6 shows the prediction errors of different approaches with deadlines of 30, 45, and 60 seconds, respectively. Note that in addition to the performance of ActiveSLA, we also report the performance of two intermediate versions of ActiveSLA, i.e., ActiveSLA-R and ActiveSLA-C. We can have the following observations. (1) The difference in prediction error between TYPE and Q-Cop for TPC-W2 is not as dramatic as that for TPC-W1. The main reason is that the execution time of the queries in TPC-W2 are very similar among different query types, and therefore counting the total number of queries will achieve almost the same effect as counting the number of queries per type. (2) ActiveSLA-R, which uses a non-linear regression model, improves over the linear regression model used by Q-Cop. The improvement is more distinct when the deadline is longer (e.g., 60 seconds vs. 30 seconds). This result suggests that when the query execution time is long (and when the system is likely to be heavily loaded), the benefits of a non-linear model are more apparent. (3) ActiveSLA-C, by using a classification model instead of the regression model used in ActiveSLA-R, consistently outperforms ActiveSLA-R. Note that because both models use the same sets of training and testing data, this performance improvement verifies our claim that a classification model (i.e., the one-step approach) has advantages over a regression model (i.e., the two-step approach) for admission control. (4) ActiveSLA, which takes all the features into consideration and builds a non-linear classification model, has the best performance in terms of minimizing the prediction error under all the deadline settings.

3.4 Result for TPC-W3 Query Set

The database and system settings for the experiments on the TPC-W3 query set are the same as those for the TPC-W2 query set. Because of the updating queries, we set up different isolation levels in the following way. In PostgreSQL, *Read Committed* is the default isolation level.

We use strict two phase locking to implement the *Serializable* Isolation. In particular, we use "LOCK TABLE *tables* IN SHARE MODE" for queries with types Q6 through Q9, whereas we use "LOCK TABLE *tables* IN EXCLUSIVE MODE" for queries of type Q10. The variable *tables* in the locking commands represents the tables required by the corresponding queries. For example, for queries of type Q6, *tables* = "item, author, order_line, orders" and for queries of type Q10, *tables* = "item", as shown in Table 2.

Table 2: Query type and the locking types/tables

	Locking(Tables)
Q6	slock(item, author, order_line, orders)
Q7	slock(customer,orders)
Q8	slock(item, order_line, orders)
Q9	slock(order_line, orders, cc_xacts, country)
Q10	xlock(item)

We collect around 6,000 data samples for *Read Committed* and another 6,000 data samples for *Serializable*. In Figure 8 we show the average lock contention delay for queries of each type, to illustrate the interference among queries of different types under *Serializable* Isolation. Q7 and Q9 query types have very short contention delay (less than 2ms) because they only require shared locks on the tables they need and because for these tables, there are no other queries requiring an exclusive lock. In comparison, the situation for Q6, Q8, and Q10 query types are very different. Because Q6 and Q8 query types require shared locks on the "item" table whereas Q10 type requires exclusive locks on the same table, lock contentions happen very often among these queries. Figure 7 shows the prediction errors of different approaches on the TPC-W3 query set. As can be seen, although different isolation levels are used, most conclusions we obtained from the TPC-W2 query set are still valid for the TPC-W3 query set.

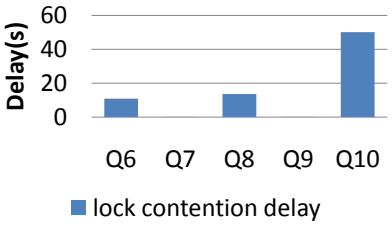

Figure 8: Lock contention delay.

3.5 Further Investigation

In this subsection, we provide some additional details about the prediction module of ActiveSLA, including some details on the classification model, an example feature that demonstrates non-linearity, and feature sensitivity as well as overhead analysis.

3.5.1 Details on the Machine Learning Model

We zoom into a segment of the learned REP decision tree model for TPC-W3, as shown in Figure 9, to illustrate how ActiveSLA makes predictions. In the REP tree, a leaf node represents the level to which the model believes that the query will *miss* its deadline (therefore negative values indicate it is very likely the query can be finished on time). The internal branches indicate the criteria used to make the decision. As can be seen from the tree, ActiveSLA first looks at CPU_wai, which is the percentage of time that the CPU is idle because the system had an outstanding disk I/O request. If CPU_wai is lower than 7.5%, the prediction stops and returns a value of -1. If CPU_wai is greater than 7.5%, ActiveSLA next looks at the *type* of the query. If the query is of type Q10, then based on the system isolation level (serializable or not), the model returns a value of 0.33 (very likely to miss deadline) or -2 (very unlikely). This segment of REP decision tree captures the fact that a query of type Q10 is an update query and so when the isolation level is read commit, the query almost never misses its deadline. On the other hand, if the isolation level is serializable, there will be a lock contention delay and so the query is very likely to miss its deadline.

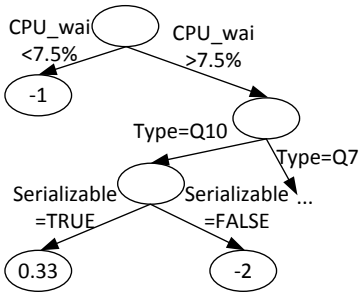

Figure 9: Part of a REP tree learned by ActiveSLA.

3.5.2 A Feature That Demonstrates Non-linearity

We use one particular feature to demonstrate the effectiveness and non-linearity of certain database features. The feature that we used is *the estimated number of pages of sequential scan according to the query plan*, i.e., *seq_page*. Figure 10 shows the scatter-plots of the number of pages of sequential scan estimated by the query plan (x-axis) vs. the query execution time (y-axis) for the queries of different types in the TPC-W2 query set.

From the figure we can obtain several observations. First, this particular feature has predictive power for Q7 and Q9 query types, which can be seen from the correlation between x-value and y-value in the corresponding sub-figures. The correlation coefficients are 0.62 and 0.52, respectively. Second, for Q6 type, this feature has almost no predictive power with a correlation coefficient as 0.20. Third, for Q8 type, when the number of pages of sequential scan is low (and therefore the query execution time is short), the feature has obvious predictive power with correlation coefficient as 0.78; however, when the number of pages of sequential scan grows larger than 350K, its predictive power disappears with a correlation coefficient as 0.17. This result illustrates the importance of the *non-linear* models used in ActiveSLA.

3.5.3 Sensitivity Analysis of Features

ActiveSLA uses three sets of features, namely, *query type/mix*, *query features*, and *database/system conditions*. To study the effectiveness of each feature set, we conduct the following experiment of sensitivity analysis.

For query sets TPC-W2 and TPC-W3, we compare the performance of (1) ActiveSLA with all three feature sets and (2) ActiveSLA with one feature set removed. Basically, we

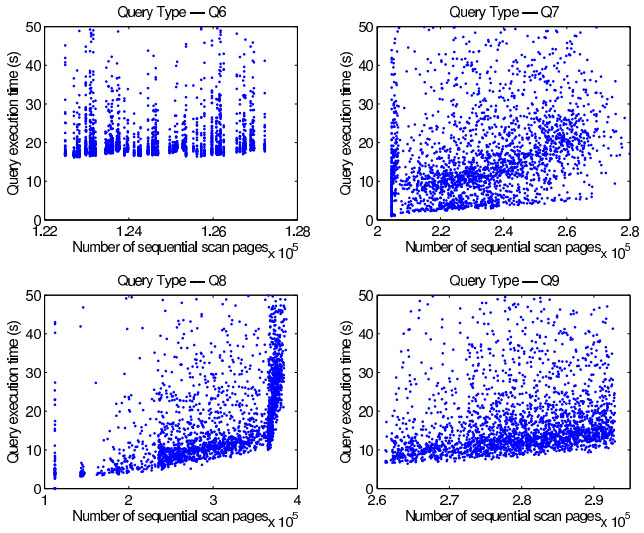

Figure 10: Influence of the number of seq_page scan on query execution time for different query types.

want to see the performance of ActiveSLA when a particular set of features are not available. The results for TPC-W2 and TPC-W3 are similar and summarized as below. (1) Removing any set of features will increase the prediction error, albeit to different degrees. (2) The importance of the feature sets, from the most important to the least important is: *query type/mix* > *query features* > *database/system conditions*, as shown in Figure 11.

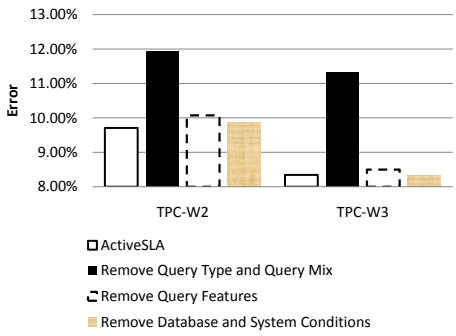

Figure 11: Sensitivity Analysis of Features when the deadline is 60s.

3.5.4 Training and Evaluation Overhead

The classifier is first trained and then evaluated. In the training stage, it takes approximately 72ms to build an initial model by using 12,000 samples. The model is rebuilt every time when there are another 100 new samples which are sent back from the admitted queries. In the evaluation stage, it takes approximately 8ms to predict the probability of the query to meet the deadline when a single query arrives at the prediction module. The prediction overhead is negligible compared with the mean query execution time and the deadline (i.e., 30s, 45s, or 60s).

4. DECISION MODULE

The decision module of ActiveSLA is responsible for making the final decision on whether a new query should be admitted. In this section, we first describe the SLAs that we assume. Next, we describe under a general SLA, how to make profit-oriented admission control decisions by using the standard decision theory. Then we show that under a commonly used SLA form, namely step-function SLA, the decisions can be made in a more efficient way. Finally, we show how our decision module takes into account the interference among clients (queries), who are competing with each other for the shared system resources.

4.1 Service Level Agreement (SLA)

A service level agreement (SLA) is a contract between a service provider and a client that determines the promised service performance and the corresponding profits (or penalties). SLAs in general depend on certain chosen criteria, such as service latency, throughput, availability, security, etc. In this paper, we focus on SLAs on query latency, i.e., query execution time. More specifically, we assume for each query q there is an associated SLA, which determines the profit that will be obtained by the service provider under different query execution times for q. An example of such an SLA is shown in Figure 12(a)(upper sub figure), where the profit is a function $s(t)$ over the query execution time t assuming the query is admitted. In addition, not explicitly shown in the figure is that if the query is rejected up-front, the service provider has to pay a penalty of -r.

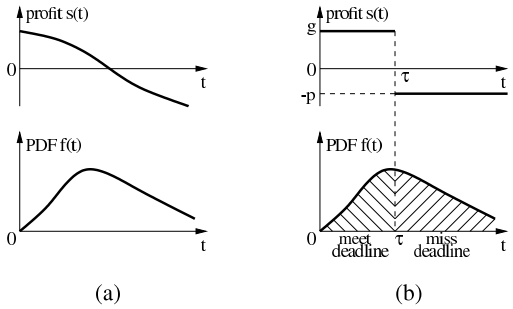

Figure 12: SLA-based admission control decisions for (a) general SLAs and (b) step-function SLAs.

Note that a more commonly used SLA is based on the quantile of the response time of queries from a single client. If this is preferred, there exist techniques (e.g., [13]) that directly map quantile-based SLAs to per-query SLAs. However, based on our extensive interactions with numerous business organizations that provide services to real clients, they desire to be able to manage SLAs at the finest granularity level (i.e., per query) with multiple levels of delivery defined in the SLAs (i.e., piecewise linear functions [6]). The observation is that currently majority of the service providers only give availability SLAs to their clients represented in the quantile form but not other types of SLAs such as latency, throughput etc. Also, lack of formal models and tooling to enable finer granularity level SLA management is a major inhibitor for businesses to adopt various types of SLAs and also varying levels. Our research aims at advancing the state-of-the art in that area and helping service providers by working with them and this paper is a part of that effort.

4.2 The Admission Decisions

The main goal of a DaaS provider is to maximize its profits by satisfying its client service level agreements (SLAs). Therefore, we use SLA profit as the final objective in our admission control strategies. In this subsection, we further assume that the probability density function (PDF) for the execution time of q, as $f(t)$ shown in Figure 12(a)(lower sub figure), is available to the service provider.

Given the above information, we can compute the expected profit $E[profit(q)]$ for admitting query q as

$$E[profit(q)] = \int_{t=0}^{\infty} s(t) \cdot f(t)dt.$$

Then by following the standard decision theory, the admission control decision that maximizes the expected SLA profit for query q should be

$$Decision = \begin{cases} Admit & \text{if } E[profit(q)] > -r \\ Reject & \text{otherwise.} \end{cases}$$

The SLA function $s(t)$ usually can be directly obtained from the service contract. However, there are still several other challenges. (1) It is very difficult to obtain the detailed query performance information for query q, i.e., the PDF in Figure 12(a), before the incoming query q is even executed. (2) Different incoming query q may have different SLAs as well as time-varying query performance information, given that the status of database management system and the operating system are constantly changing. (3) Because we assume that queries come in an online fashion and there is no prior knowledge about the future picture, we also should reserve resources for future "more profitable" queries. In the following, we address these challenges by extending the standard decision theory.

4.3 Single Query Decision

Although the SLA on query execution time may take different forms, a step function is commonly used in real contracts because it is easy to describe in natural language. We show such a step-function SLA in Figure 12(b) and Table 3. That is, for a given query q, if the query is admitted and its query execution is finished before the deadline τ, the service provider obtains a profit gain of g; else if the query misses the deadline τ, the service provider pays a penalty of -p. Otherwise the service provider may up-front reject the query and pay a penalty of -r.

Table 3: Step-function SLA: outcomes and profits.

	Meet Deadline	Miss Deadline
Admit	g	$-p$
Reject	$-r$	$-r$

Under the step-function SLA, we can simplify the expected profit for admitting q as (also illustrated in Figure 12(b))

$$\begin{aligned} E[profit(q)] &= \int_{t=0}^{\tau} g \cdot f(t)dt + \int_{t=\tau}^{\infty} (-p) \cdot f(t)dt \\ &= g \cdot \int_{t=0}^{\tau} f(t)dt - p \cdot \int_{t=\tau}^{\infty} f(t)dt \end{aligned}$$

The above result reveals that to compute the expected profit under the step-function SLA, we only need the *area* under $f(t)$ before τ and that after τ, which are actually the probabilities of meeting and missing the deadline. Such probabilities, as have been shown in the previous sections, are provided by the prediction module of ActiveSLA in a real-time fashion. If the probability for the query to meet its deadline is c, we can easily see that $\int_{t=0}^{\tau} f(t)dt = c$ and $\int_{t=\tau}^{\infty} f(t)dt = 1 - c$. As a result, we have $E[profit(q)] = g \cdot c - p \cdot (1 - c)$. Thus the exact PDF for the execution time of q is not necessary for admission decision anymore. The admission control decision is made as

$$Decision = \begin{cases} Admit & \text{if } g \cdot c - p \cdot (1 - c) > -r \\ Reject & \text{otherwise.} \end{cases}$$

4.4 Multiple Query Decision

The admission decision made in Section 4.3 is based on the expected profit for admitting a single query q. If we want to make the most profit from multiple queries, we have to take into consideration at least two additional hidden costs when we decide whether or not to admit q. (1) Admitting q into the database server may slow down the execution of other queries that are currently running in the server, since query q will consume system resources. Therefore, admitting q will potentially cause other running queries to miss their deadlines which they were able to meet. This will reduce the total profit of the DaaS provider. (2) Admitting q will consume system resources and change the system status. This may result in the rejection of the next query, which may otherwise be admitted and bring in a higher profit. The two additional hidden costs are closely related to the concept of *opportunity cost* in economics [17, 25]. We denote the opportunity cost as o, and we revise the decision module in ActiveSLA according to Table 4, in order to take the opportunity cost into account.

Table 4: Step-function SLA: outcomes and profits, with opportunity cost.

	Meet Deadline	Miss Deadline
Admit	$g - o$	$-p - o$
Reject	$-r$	$-r$

According to this new table, when $o > 0$, the admission control is relatively more aggressive in rejecting new queries, in order to protect the currently running queries and to reserve system resources for later queries with potentially higher profits. With such an extra cost term, the admission control decision becomes

$$Decision = \begin{cases} Admit & \text{if } g \cdot c - p \cdot (1 - c) - o > -r \\ Reject & \text{otherwise.} \end{cases}$$

In practice, the value o for opportunity cost can be either determined by the service provider (e.g., derived from certain business considerations) or learned through the ActiveSLA feedback channel over time. It is important to note that our goal in this work is to model the opportunity cost concept as a way of managing multiple query decisions and to show its effectiveness through experimental studies. The determination of the exact value of o in system settings is an orthogonal and interesting problem by itself, which is outside of the scope of this paper and is in our future plans.

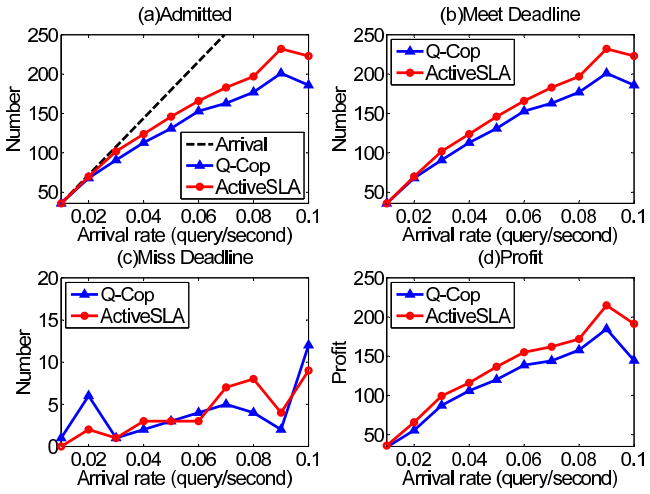

Figure 13: Result with stationary workload: number of queries that (a) are admitted, (b) meet deadline after admitted, (c) miss deadline after admitted, and (d) total profit.

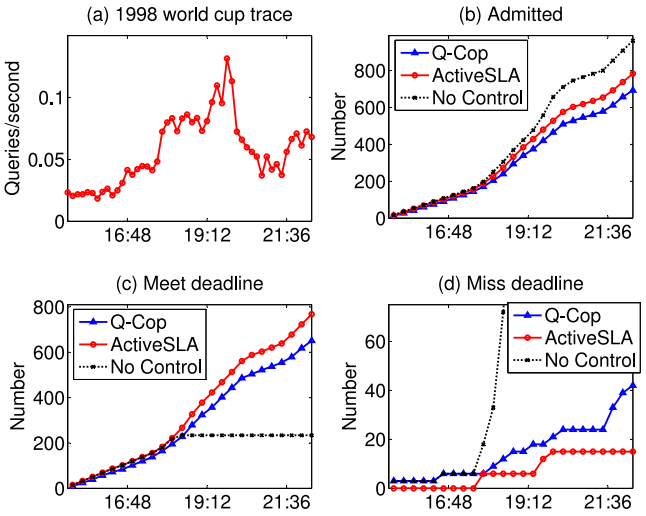

Figure 14: Dynamic workload: over time, (a) the rate of arrived queries, the cumulative numbers of (b) admitted queries, (c) queries that are admitted and meet their deadlines, and (d) queries that are admitted but miss their deadlines.

5. DECISION MODULE EVALUATION

We use the workload generators to generate different workload traces in an offline fashion. Then we test the workload traces in the real-time system to present the effectiveness of our system. We also compare ActiveSLA's performance with the previous work, Q-Cop. We report the results based on the TPC-W2 query set in this section and skip those for other query sets because the results are similar. For each test, unless stated otherwise, we repeat 5 times (with traces generated from different random seeds) and report the average performance.

5.1 Result with Stationary Workload

In this experiment, we use a stationary workload with arrival rates ranging from 0.01 request/second to 0.1 request/second. Each test runs 1 hour. For the SLA profit, we use the numbers for q_1 as shown in Figure 2. We can obtain the following observations from the results as shown in Figure 13. (1) When the arrival rate is less than 0.03 request/second, because the system is underloaded, both Q-Cop and ActiveSLA admit most of the queries. (2) When the arrival rate goes beyond 0.03 request/second, load shedding starts to take place more frequently. However, under all the arrival rates, ActiveSLA admits between 10% to 15% more queries than Q-Cop, and among the admitted queries, the numbers of queries that miss their deadline are comparable between ActiveSLA and Q-Cop. These results show that ActiveSLA makes more reasonable admission control decisions. (3) The advantage of ActiveSLA's better decisions is reflected in its higher SLA profits compared to Q-Cop.

5.2 Result with Non-stationary Workload

For the non-stationary workload, i.e., the World Cup trace from the 1998 World Cup site [4] from 15:00pm to 22:21pm, we study two experiments. In the first experiment, we assume all the queries have the same SLA. In the second experiment, we use two different profit profiles to show how ActiveSLA makes profit-oriented decision. More specifically, in the second experiment we assume half of the queries have one SLA and the other half have another SLA, in order to demonstrate that ActiveSLA is able to provide profit-oriented service differentiation.

5.2.1 *Profit-oriented decision*

Table 5: Comparison of SLA profit (with the total number of queries being 963).

	Admitted	Meet Deadline	Miss Deadline	Profit
$g = 1$(gain),$p = 1$(penalty),$r = 0.1$(reject penalty)				
Q-Cop	693	651	42	582
ActiveSLA	783	768	15	735
$g = 1$(gain),$p = 2$(penalty),$r = 0.1$(reject penalty)				
Q-Cop	693	651	42	540
ActiveSLA	744	744	0	722.1

We start by using again the SLA described by the first profit profile in Figure 2, i.e., $g = 1$, $p = 1$, $r = 0.1$, and $o = 0$. Figure 14 shows the experimental results. From these figures we can see that over time, ActiveSLA admitted more queries and fewer queries admitted by ActiveSLA missed their deadlines than those admitted by Q-Cop. In addition, in Table 5, we report the aggregated results over the whole period of the World Cup event. From the table we can see that during this event, ActiveSLA admitted 10% more queries than Q-Cop did and compared to Q-Cop, fewer queries admitted by ActiveSLA missed their deadline. Overall, ActiveSLA achieved 20% more profit than Q-Cop.

Next, we switch to the SLA described by the second profit profile Figure 2, i.e., $g = 1$, $p = 2$, $r = 0.1$, and $o = 0$. The aggregated results are also shown in Table 5. The fact that ActiveSLA outperforms Q-Cop remains valid. However, because of the higher penalty of missing a deadline, ActiveSLA became more conservative in that it admitted fewer queries and made less profit. In this case, the conservative admission control by ActiveSLA is well justified—among the queries admitted into the system by ActiveSLA,

none of them missed their deadlines and so the high penalty of missing deadlines has been avoided.

5.2.2 Profit-oriented service differentiation

In this experiment, we use the same world cup trace as used in the previous part. However, this time we randomly pick half of the queries and assign them SLAs with higher gain. More specifically, for those queries picked (which we refer to as Gold queries) we increase their g values in the SLA to 1.5 while for the rest queries (which we refer to as Silver queries), we keep their original g values of 1.

In addition, we study two scenarios, one scenario where there is no opportunity cost (i.e., $o = 0$) and the other with opportunity cost (with $o = 0.2$), in the decision module of ActiveSLA. Table 6 reports the results for these scenarios, where we separate the performance of Gold queries(G) and that of Silver ones(S). From the results we can have the following observations. (1) In both scenarios, ActiveSLA admitted more queries than Q-Cop and made more profit. (2) Because the potential profit gain for Gold queries is higher, ActiveSLA was more aggressively in admitting Gold queries than in admitting Silver queries (and resulted in more Gold queries missing their deadlines). But such aggressive decisions were rewarded by the higher profit. (3) When the opportunity cost is taken into consideration in the decision module, ActiveSLA admitted fewer Silver queries and at the same time, admitted more Gold queries (compared to the scenario when there was no opportunity cost). In addition, because of the protection effect of the opportunity cost, fewer Gold queries missed their deadlines once they were admitted into the system.

It is worth noting that we have only demonstrated that the opportunity cost o impacts the profit the way we had expected, and we have not provided a systematic way to set the value of o. In real implementations, o can be either a tuning factor that the service provider needs to set, or it can be automatically adjusted during the runtime through a feedback loop.

Table 6: Comparison of SLA profit with the total number of queries being 963(Gold/Silver).

	Admitted (G/S)	Meet Deadline (G/S)	Miss Deadline (G/S)	Profit
$g = 1.5/1$(gain),$p = 1$(penalty),$r = 0.1$(reject penalty)				
Q-Cop	(365/328)	(341/310)	(24/18)	752.5
ActiveSLA ($o = 0$)	(420/384)	(396/375)	(24/9)	920.1
ActiveSLA ($o = 0.2$)	(438/348)	(432/347)	(6/1)	970.3

In summary, the experimental results in this section demonstrate that compared to Q-Cop, by using ActiveSLA admission control, more queries are admitted, fewer admitted queries miss their deadlines, the gain and penalty in the SLAs are automatically weighed, and when needed, differentiated services are achieved.

6. RELATED WORKS

In this section we survey prior work in database management systems that handles system overload. We group prior approaches into two major categories, i.e., admission control and control-theoretic approaches.

Admission control.

The use of admission control as an overload management technique has been investigated by several systems. Most of the techniques are based on rejecting incoming work to a service by refusing to accept new requests. For example, Schroeder et al. [23] dynamically adjust the lowest MPL that corresponds to maximum system performance. Welsh and Culler propose an adaptive approach to overload control in the context of the SEDA Web server [31] to control the 90th-percentile response time of requests. Popovici and Wilkes [22] uses simulation to develop scheduling policies to make profits in the uncertain resource environment. Contrast to the admission control mechanisms which are oblivious to query types and query mixes, Q-Cop [27], QShuffler [3] and Gatekeeper [10] take into consideration the different requirement for different type of queries when admission control decisions are made. ActiveSLA has the advantage of [23] where the decision module dynamically tunes the best MPL as there are different optimal MPL for different workloads. However, ActiveSLA distinguishes itself from these works in two major aspects. (1) Driven by maximizing the expected profit for the DaaS providers, ActiveSLA makes decisions based on the probabilities for queries to meet or miss their deadlines as well as the query-specific SLAs. (2) In order to obtain accurate prediction on query performance, ActiveSLA takes into consideration many more features (at very low overheads) than the state-of-the-art approaches.

Control-theoretic approaches.

Control theory provides a formal framework for monitoring, modeling and management about the behavior of dynamic systems. Several control-theoretic approaches have focused on overload control. For example, Abdelzaher and Lu [2] describe a feedback PI controller scheme that attempts to maintain a CPU utilization target based on a simplistic linear model of server performance. Tu et al. [29] design a feedback controller for load shedding in order to meet quality requirements when overloaded. Although control theory provides a very useful set of tools for monitoring, modeling and management about systems subject to feedback, there are many challenges that must be addressed in order for these techniques to be applicable to DaaS. One of the greatest difficulties in control-theoretic approaches is that the designers of the controller always assume a linear model around the operation point [21], which may not be accurate in describing DaaS with widely distinct query workloads, time-varying caching and resource contentions. Moreover, when a system is subject to a borderline overload condition, we expect that a small amount of additional workload will disproportionally degrade the system performance. Compared with the linear model that is often used in control-theoretic approaches, ActiveSLA adopts a nonlinear classification model to build the prediction model by taking into consideration of query-specific features system-specific features, as well as the query type and mix in a comprehensive way. Moreover, the output of the prediction model well supports the decision module to make the profit-oriented decision.

7. CONCLUSION AND FUTURE WORK

In this paper, we proposed a framework, ActiveSLA, for admission control in cloud database systems. ActiveSLA differs from existing approaches in two aspects. First, compared with existing approaches that build regression models and return single point estimations of query execution time, ActiveSLA builds a non-linear classification model to predict the probability for each query to meet/miss its deadline directly. Moreover, besides query type and query mix that are used in existing work, ActiveSLA also takes into consideration query features as well as the database-specific and system-level metrics, which further help to improve the prediction accuracy. Second, the admission control decisions made by ActiveSLA are steered by service-level-agreements and expected profits. Therefore, differentiated services, which are very important in cloud databases, are provided.

We plan to improve our ActiveSLA in the future in the following aspects. (1) ActiveSLA relies on the query optimizer to obtain query features such as the number of sequential I/O. Although we observe positive predictive power of these query features, such numbers obtained from query optimizers are known to be notoriously inaccurate, e.g., due to the incorrect statistics and cardinality estimates of a query execution plan. In the future, we plan to repair the inaccuracy in real-time (e.g., similar to [26, 20]) to make better predictions. (2) In most DaaS deployments, different replication levels are provided to overcome the failures that may occur to commodity hardware. In the future, we plan to extend our prediction module by including the level of replication as one of the system variables used in non-linear classification. (3) We are also planning to extend our ActiveSLA to deal with different types of database systems to manage data and serve queries, e.g., NoSQL databases.

Acknowledgment

The Georgia Tech authors have been partially supported by National Science Foundation by IUCRC, CyberTrust, CISE/CRI, and NetSE programs, National Institutes of Health grant U54 RR 024380-01, PHS Grant (UL1 RR025008, KL2 RR025009 or TL1 RR025010) from the Clinical and Translational Science Award program, National Center for Research Resources, and gifts, grants, or contracts from Wipro Technologies, Fujitsu Labs, Amazon Web Services in Education program, and Georgia Tech Foundation through the John P. Imlay, Jr. Chair endowment. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation or other funding agencies and companies mentioned above.

8. REFERENCES

[1] R. K. Abbott and H. Garcia-Molina. Scheduling real-time transactions: A performance evaluation. *ACM Trans. Database Syst.*, 1992.

[2] T. F. Abdelzaher and C. Lu. Modeling and performance control of internet servers. In *Proc. of CDC*, 2000.

[3] M. Ahmad, A. Aboulanaga, and S. Babu. Modeling and exploiting query interactions in database systems. In *Proc. of CIKM*, 2010.

[4] M. Arlitt and T. Jin. Workload characterization of the 1998 World Cup web site. *HP Tech. Rep.*, 1999.

[5] S. Aulbach, T. Grust, D. Jacobs, A. Kemper, and J. Rittinger. Multi-tenant databases for software as a service: schema-mapping techniques. In *Proc. of SIGMOD*, 2008.

[6] Y. Chi, H. J. Moon, and H. Hacigumus. iCBS: Incremental cost-based scheduling under piecewise linear SLAs. *PVLDB*, 2011.

[7] B. Cooper, R. Ramakrishnan, U. Srivastava, A. Silberstein, P. Bohannon, H.-A. Jacobsen, N. Puz, D. Weaver, and R. Yernenin. Pnuts: Yahoo!'s hosted data serving platform. In *Proc. of VLDB*, 2008.

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

[9] C. Curino, Y. Zhang, E. P. C. Jones, and S. Madden. Schism: a workload-driven approach to database replication and partitioning. *PVLDB*, 2010.

[10] S. Elnikety, E. Nahum, J. Tracey, and W. Zwaenepoel. A method for transparent admission control and request scheduling in e-commerce web sites. In *Proc. of WWW*, 2004.

[11] J. Friedman, T. Hastie, and R. Tibshirani. Additive logistic regression: a statistical view of boosting. *Annals of Statistics*, 28(2):337–407, 2000.

[12] A. Ganapathi, H. Kuno, U. Dayal, J. L. Wiener, O. Fox, and M. Jordan. Predicting multiple metrics for queries: Better decisions enabled by machine learning. In *Proc. of ICDE*, 2009.

[13] D. Gmach, S. Krompass, A. Scholz, M. Wimmer, and A. Kemper. Adaptive quality of service management for enterprise services. *ACM Trans. Web*, 2008.

[14] C. Gupta, A. Mehta, and U. Dayal. Pqr: Predicting query execution times for autonomous workload management. In *Proc. of ICAC*, 2008.

[15] H. Hacigumus, B. Iyer, and S. Mehrotra. Providing database as a service. In *Proc. of ICDE*, 2002.

[16] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I. H. Witten. The WEKA data mining software: An update. *SIGKDD Explorations*, 11(1), 2009.

[17] D. E. Irwin, L. E. Grit, and J. S. Chase. Balancing risk and reward in a market-based task service. In *Proc. of HPDC*, 2004.

[18] D. Jiang, B. C. Ooi, L. Shi, and S. Wu. The performance of mapreduce: an in-depth study. *Proc. VLDB Endow.*, 2010.

[19] G. Jung, K. Joshi, M. Hiltunen, R. Schlichting, and C. Pu. A cost-sensitive adaptation engine for server consolidation of multi-tier applications. In *Proc. of Middleware*, 2009.

[20] G. Luo, J. F. Naughton, C. J. Ellmann, and M. W. Watzke. Toward a progress indicator for database queries. In *Proc. of SIGMOD*, 2004.

[21] P. Padala, K. Hou, X. Zhu, M. Uysal, Z. Wang, S. Singhal, A. Merchant, and K. Shin. Automated control of multiple virtualized resources. In *Proc. of EuroSys*, 2009.

[22] F. I. Popovici and J. Wilkes. Profitable services in an uncertain world. In *Proc. of SC*, 2005.

[23] B. Schroeder, M. Harchol-Balter, A. Iyengar, E. M. Nahum, and A. Wierman. How to determine a good

multi-programming level for external scheduling. In *Proc. of ICDE*, 2006.

[24] P. Shivam, A. Demberel, P. Gunda, D. Irwin, L. Grit, A. Yumerefendi, S. Babu, and J. Chase. Automated and on-demand provisioning of virtual machines for database applications. In *Proc. of SIGMOD*, 2007.

[25] E. A. Silver, D. F. Pyke, and R. Peterson. *Inventory Management and Production Planning and Scheduling*. Wiley, third edition, 1998.

[26] M. Stillger, G. M. Lohman, V. Markl, and M. Kandil. LEO - DB2's learning optimizer. In *VLDB*, 2001.

[27] S. Tozer, T. Brecht, and A. Aboulnaga. Q-cop: Avoiding bad query mixes to minimize client timeouts under heavy loads. In *Proc. of ICDE*, 2010.

[28] Transaction Processing Performance Council. TPC benchmark W (web commerce), February 2002.

[29] Y.-C. Tu, S. Liu, S. Prabhakar, and B. Yao. Load shedding in stream databases: a control-based approach. In *Proc. of VLDB*, 2006.

[30] B. Urgaonkar, P. Shenoy, and T. Roscoe. Resource overbooking and application profiling in shared hosting platforms. In *Proc. of OSDI*, 2002.

[31] M. Welsh and D. Culler. Adaptive overload control for busy internet servers. In *Proc. of USITS*, 2003.

[32] P. Xiong, Y. Chi, S. Zhu, H. J. Moon, C. Pu, and H. Hacigumus. Intelligent management of virtualized resources for database management systems in cloud environment. In *Proc. of ICDE*, 2011.

APPENDIX

In this appendix, we provide details on the features used in ActiveSLA and how they are obtained.

A. QUERY FEATURES

We use "explain" command in PostgreSQL and MySQL to provide the necessary support to obtain query specific features. For example, the query cost in PostgreSQL depends mainly on 5 parameters, i.e., the number of sequential I/O (*seq_page*), the number of non-sequential I/O(*random_page*), the number of CPU tuple operations (*cpu_tuple*), the number of CPU index operations (*cpu_index*), and the number of CPU operator operations (*cpu_operator*). Each operation is assigned a unit cost by PostgreSQL, e.g., by default these unit costs are set to 1.0, 4.0, 0.01, 0.001, and 0.0025, respectively. The total estimated cost for a query plan is

$$cost = 1.0 \times seq\_page + 4.0 \times random\_page + 0.01 \times cpu\_tuple + 0.005 \times cpu\_index + 0.0025 \times cpu\_operator$$

With this background information, we can either directly look into the detailed query cost, or indirectly infer the values of the 5 parameters of a query in the following way. For example, in order to get the query feature *seq_page*, we call the "explain" command twice, with 1.0 and (1.0+Δ) as the unit cost for *seq_page*. Assuming the results of the two are *cost* and *cost'*, with

$$cost' = (1.0 + \Delta) \times seq\_page + 4.0 \times random\_page + 0.01 \times cpu\_tuple + 0.005 \times cpu\_index + 0.0025 \times cpu\_operator,$$

If Δ is a very small and the best query plan does not change, then have $seq\_page = (cost' - cost)/\Delta$.

Similarly, the "explain" command in MySQL outputs a table, in which two important columns are 'type' and 'rows'. Column 'type' shows what kind of scan to be used, e.g., 'ALL' means a sequential scan of the whole table. Column 'rows' shows the estimation on how many rows will be returned.

B. DATABASE AND SYSTEM CONDITIONS

Here are database and system features that we collected.

Transaction isolation: The SQL standard defines four levels of transaction isolation, i.e., Read Uncommitted, Read Committed, Repeatable Read, and Serializable. In PostgreSQL, for example, Read Uncommitted is treated as Read Committed, while Repeatable Read is treated as Serializable. As a feature, we use a nominal variable FALSE,TRUE to denote whether Read Committed or Serializable is chosen.

Buffer cache: Each cache entry in buffer cache points to an 8KB block (sometimes called a page) of data. When a process requests a buffer, it calls BufferAlloc with what file/block it needs. If the block is already in the cache, it gets pinned and then returned. Otherwise, a new buffer must be found to hold this data. Therefore, for example, if a query is going to do a sequential scan of a table whose size is 100 pages and there are 50 pages in the buffer cache, we use 50 as the value for the feature *DB_buffer*. In order to obtain such information, for the buffer cache, we create a view called "pg_buffercache" to collect the number of pages of a table in DB buffer and we query this view to get the feature value.

System cache: Databases are designed to rely heavily on the operating system cache. The DB buffer and system cache usually work as follows: Backends that need to access tables first look for needed blocks in DB buffer. If they are already there, they are fetched right away. If not, an operating system request is made to load the requested blocks. The blocks are loaded either from the kernel disk buffer cache, or from disk. Therefore, for example, if a query is going to do a sequential scan of a table whose size is 100 pages among which 10 pages are in the system cache, then we use 10 as the value for the feature *Sys_cache*. We describe how we obtain it PostgreSQL and MySQL. Similar methods can be used to monitor system cache content for other OS and RDBMS. It mainly contains two steps. (1) Obtain the data file location. PostgreSQL uses a directory to store all the data in all the databases. The default location is "/usr/local/pgsql/data/base". An object in PostgreSQL has its unique oid. Assume that the database and the table that we are interested in have oid d_{oid} and t_{oid}, respectively, where t_{oid} and d_{oid} can be obtained from pg_database table. Then the filename to store the data of this table turns out to be "/usr/local/pgsql/data/base/d_{oid}/t_{oid}" [3]. The default data directory for MySQL is "/var/lib/mysql". Assume that the database and the table name that we are interested in are d_{name} and t_{name}, respectively. Then the file to store the data of this table is "/var/lib/mysql/d_{name}/t_{name}.MYD" [4]. (2) We wrote a Perl script to return the portion of the files in the system cache, from which we get the number of pages of a table in system cache.

Besides cache, we also collect some other general system metrics, such as CPU stats, memory stats and disk stats.

[3]If the table is larger than 1GB, there will be several files

[4]MyISAM storage engine

These metrics are obtained by running dstat and iostat. We summarize all the features that we use in ActiveSLA and the methods by which we obtained these features in Table 7.

Table 7: Features, description, and obtain methods.

Features	Description	Obtain methods
Query type and mix		
type	query type	
num_i,avg_i	number and average running time of queries of type i	Server
Query features		
seq_page	sequential I/O	PostgreSQL
rand_page	non-sequential I/O	PostgreSQL
cpu_tuple	CPU tuple operations	PostgreSQL
cpu_index	CPU index operations	PostgreSQL
cpu_operator	CPU operator operations	PostgreSQL
System features		
Transaction isolation	Read Commit(FALSE) Serializable(TRUE)	Server
DB_buffer	pages in DB buffer	PostgreSQL
Sys_cache	pages in System cache	Perl
CPU	CPU_usr,CPU_sys,CPU_idl CPU_wai,CPU_hiq,CPU_siq	dstat
MEM	MEM_used,MEM_free MEM_buff,MEM_cach	dstat
DISK	DISK_rrqm/s, DISK_wrqm/s DISK_r/s, DISK_w/s DISK_rsec/s, DISK_wsec/s DISK_rq, DISK_qu,DISK_await DISK_svctm, DISK_%util	iostat