Predicting Query Execution Time

Vamshi Pasunuru

Mid-term ME Project Report

Abstract

The ability to estimate the query execution time is central for a number of tasks in database system such as query scheduling, progress monitoring and costing during query optimization. Recent work has explored the use of statistical techniques in place of the manually constructed cost models used in query optimization. Such techniques, which require as training data along with the actual execution time, promises superior accuracies for they being able to account the for factors such as hardware characteristics and bias in cardinality estimates. However, such techniques fail to generalize i.e., produce poor estimates for queries that are not seen during the training.

In this work, we propose and evaluate predictive modeling techniques that learn query execution behavior at a fine grained operator level. For each operator, we consider different sets of features and build different models for them. Since there are only finitely many operators in database, this approach is practical and will be able to estimate any query as its a composition of many operators. We evaluate our approaches using TPC-H workload on PostgreSQL.

1 INTRODUCTION

Database systems can greatly benefit from accurate execution time predictions including:

- Admission control: Resource managers can use this metric to perform workload allocations such that the specific QoS are met [16].
- Query Scheduling: Knowing the execution time is crucial in deadline and latency aware scheduling
- Progress monitoring: Knowing the execution time of an incoming query can help avoid *rogue queries* that are submitted in error and take an unreasonably long time to execute [9].

Currently, execution time estimation is based on manually constructed models, which are part of the query optimizer and typically use combinations of weighted estimates of the number of tuples flowing through operators, column widths, etc. Unfortunately, such models often fail to capture several factors that affect the actual resource consumption. For example, they may not include detailed modeling of all of the various improvements made to database query processing such as nested loop optimizations [1, 4] which localize references in the inner subtree, and introduce partially blocking batch sorts on the outer side, thereby increasing the memory requirements and CPU time and reducing I/O compared to the traditional iterator model. Similarly, they may not accurately reflect specific hardware characteristics of the current production system or the impact of cardinality estimation errors. Analytical cost models predominantly used by the current generation of query optimizers cannot capture these interactions and complexity; in fact, they are not designed to do so. While they do a good job of comparing the costs of alternative query plans, they are poor predictors of plan execution latency. Recent work [1] showed this result for TPC-DS [13], and in this work we do the same for TPC-H [12] data and queries.

In this work, we utilize learning based models and prediction techniques which are promising reasonable accuracies in recent work [1, 2, 3]

2 Background : Model Based Prediction

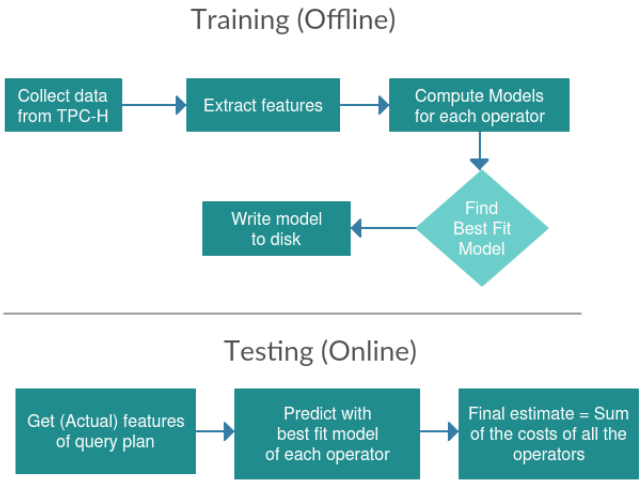
We use the term model to refer to any predictive function such as Linear Regression, Least Squares and

Support Vector Machines. Training a model involves using historical data sets to determine the best model instance that explains the available data. For example, fitting a function to a time series may yield a specific polynomial instance that can be used to predict future values. Model training (or building) requires selecting (a) the feature at- tributes, a subset of all attributes in the data set, and (b) a predic tion model, e.g., Linear Regression and Support Vector Machines (SVMs), to be used for modeling. In general, we *cannot* know which model type and feature set will produce the most accurate model for a given data set without building and testing multiple models. In some cases, a domain expert can manually specify the feature attributes. Alternatively, feature attributes can be learned automatically; however, given a set of n attributes, trying the power set is prohibitively expensive if n is not small or training is expensive thereby requiring heuristic solutions. Most approaches rank the candidate attributes (often based on their correlation to the prediction attribute using metrics such as information gain or correlation coefficients) and use this ranking to guide a heuristic search [**?**] to identify most predictive attributes tested over a disjoint test data set.

In this work we currently don't use any feature selection algorithm but rather rely on domain knowledge of SQL Query processing. For the purpose of determining the best fit curve for the given set of features we consider building multiple models of different types. In each one of the models we use a single type of prediction model, either Linear Regression or SVMs, that performs well.

Hypothesis testing and confidence interval estimations are two common techniques for determining predictive accuracy [11]. As mentioned, it is not possible to estimate a priori what model would be most predictive for a given data set without training/testing it. In this work, we are using disjoint training and testing datasets that are generated from qgen tool of TPC-H workload [12]. The mean relative error is used to estimate the accuracy of our prediction models.

3 Overview of Proposed Approach



In this section, we elaborate on the overview of the approach which consists of 2 phases. Like most other machine learning approaches, the first phase is an off-line training phase and the second is on-line testing phase. During the training phase, we construct number of different models for each type of physical database operator(e.g., Sort-Merge Join, Nested Loop, Index Nested Loop,..). Each operator will be associated with so called 'Best-Fit' model which will be determined based on the Operator model selection described in section 3.1. This model will be invoked for the purpose of estimating the execution time for a certain operator. Since there are only finite number of operators and only a couple of physical implementations for them, the space consumption is not excessive.

During the testing phase, we actually run the query to get the true cardinality estimates. While this seems to contradict the whole estimation problem, right now this is required because the focus is on solving the modeling problem given the actual cardinalities. The more practical scenario i.e., where we we only the know the estimated cardinalities, is a much harder problem as now the models also need to be Robust to the errors in cardinality estimates. This is the natural follow-up work to do. Once after the execution of the query, we need to extract features depending on the operator and invoke the Best Fit model (which is already computed during training phase) to get an estimate. We sum up costs for all the operator present in the plan tree to finally give an estimate of execution time.

TO-DO : 2. Describe the concept of multiple mod-

els for a single operator. 3. How do we select the best one among them. (Mention that current experiments don't use this idea, only one a simple non-linear model is used for all models with 2 features)

Analytical approaches problems:

counting the numbers of pages read is not sufficient as the discrepancy between random and sequential I/O is tremendous.

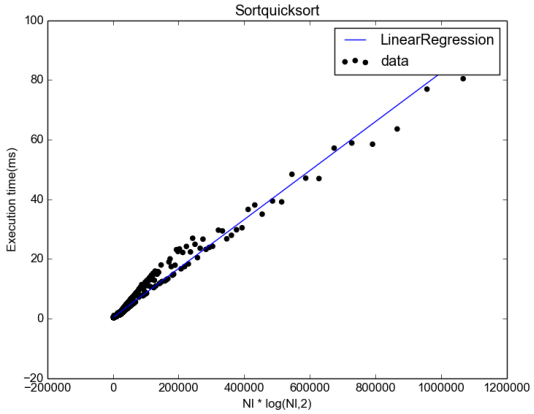
4 Training and Model selection

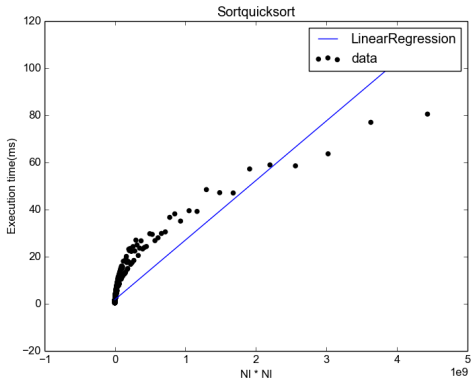
Before we describe the concept of multiple models for an operator, we shall present the motivation for doing so. While the statistical models can find the complex non-linear dependencies between features and the target, they cannot model the interaction among operators which might affect the target value.

Consider the classic example of Sorting, where in the target variable (execution time) is proportional to $N_l \log N_l$. Unless this is used as a feature, the accuracy of the resulting model can be disappointing. Since we don't know the right function to be used beforehand, we explore the possible set of functions listed bellow and select the one with the minimum estimation error. We've used functions that are similar to the one's defined in [5].

- Linear: $a_0 N_l + a_1 N_r$
- Quadratic: $a_0 N_l N_r$, $a_0 N_l N_l$, $a_0 N_r N_r$ ...
- Logarithmic: $a_0 N_l log(N_r), a_0 N_r log(N_l)$ ..
- Exponential: $a_0 N_l^{N_r}$

To illustrate the use of the functions in more details, we have plotted the curves with different possible functions for QuickSort as well as IndexNestedLoop operator. For the purpose of generating curves for QuickSort we have used the following Query template

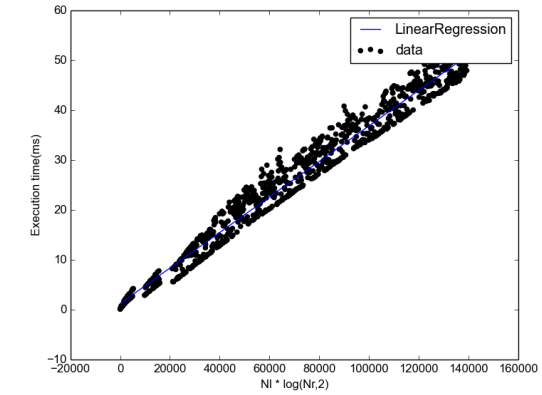


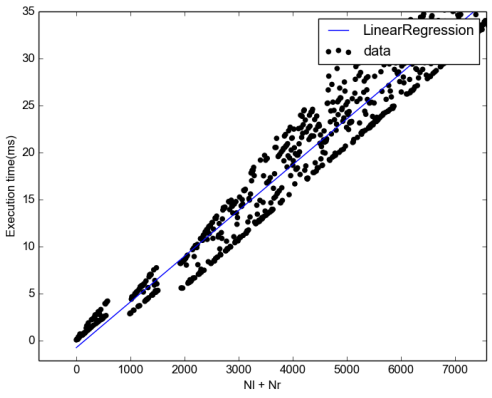


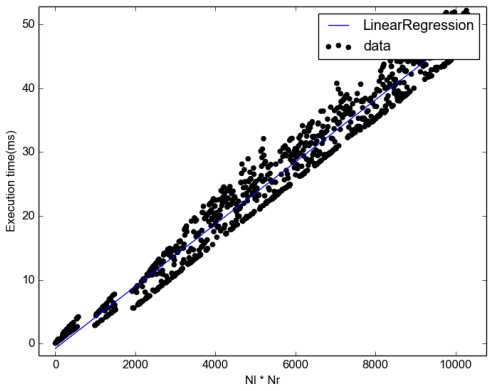
Unsurprisingly, logarithmic function fits better than the quadratic. While in some cases (Such as QuickSort) the appropriate function is obvious from SQL query processing, it does not hold for all operators. For example, consider Nested Loop Join which can be modeled with quadratic functions in general but when the inner relation has an index the same model produces estimates estimates which are "off" by orders of magnitude. In the following example, we take an exploratory approach to determine the best function for the Index Nested Loop operator. As in the case of QuickSort we have used the following query template to generate the required data.

```
Select * from orders
where o_orderkey <= :varies
order by o_totalprice
```

```
select * from lineitem,orders
where l_orderkey = o_orderkey
  and l_orderkey<= : varies;
```







Out of the above 3 functions, $N_l log(N_r)$ fits data better than other functions.

4.1 Best Fit Model

Best fit model M_O for an Operator O is the model which has the minimum estimation errors for the set of training queries. The estimation error is computed as follows,

$$\frac{1}{n}\sum_{i=1}^{n}|Actual_i - Estimated_i|$$

5 Preliminary Experiments

5.1 Setup

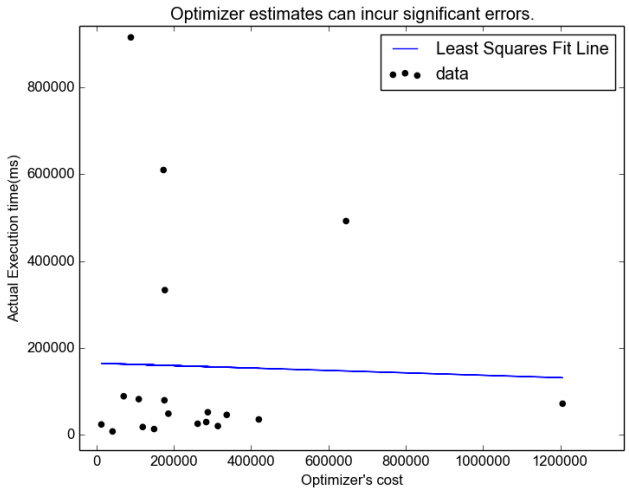
Our experimental study uses TPC-H decision support benchmark [12] on the top of PostgresSQL. The details are as follows,

- Database Management System : PostgreSQL 9.4
- Data sets and workload: We created 1GB TPC-H database according to the specification. The primary key indices as indicated in the TPC-H specification were created for both databases. We have excluded Query 1,21 To keep the training time under control, and Queries 15 as it creates a view which is not yet supported in our work. This resulted in 19 out of 22 TPC-H queries being used for training. We've used TPC-H qgen tool to generate 10 instances of each of the 19 queries, resulting in a total of 190 queries.
- Hardware: The queries were executed on a machine with 8GB RAM running Ubuntu. buffer pool size was set to 1GB (25% of the total RAM as the rule of thumb). All queries were executed sequentially with cold start (i.e., OS buffers flushed before the start of each query).
- Statistical Models: As of now, we have limited our experiments to at-most 2 features (Input cardinalities N_l, N_r). For evaluating the best-fit model we have used linear regression models (available from python's sckit-learn [14])

  TODO : 3. Performance operator wise. Box plot 4. Insights : which operators are hard to predict

5.2 Prediction with Optimizer cost models

We start with the results showing that Optimizer's estimates are a poor indication of the real execution time. For this purpose, we have taken 19 of 22 queries (To make a fair comparison with our approach we've excluded the 3 templates for the reason stated earlier). The actual execution time and prediction times are computed using *Explain Analyze* command of Postgres. The values collected are averaged over 5 runs.



We've plotted the linear regression line resulting from fitting these points using linear least-squares regression (which can be seen as an error-minimizing mapping of the optimizer-estimated cost (which is not measured in ms) to CPU- time). As we can see, even for the mapping chosen to minimize the overall error, there are significant differences between the estimated CPU cost and real CPU time for many queries. Similar observations have been made in other works as well [2, 3]

Note that while the final estimate made by the optimizer may not be an accurate reflection of real execution time, the estimates provided at a finer level (i.e., plan level, operator level etc.) are quite useful. In the next section, we show how we can use optimizer's estimates at operator level and produce an estimate for overall execution time.

5.3 Operator level modeling

PostgreSQL query optimizer provides the wide range of information at multiple levels. For example, for the Hash operator, it provides the following information.

Node Type	Hash
Parent Relationship	Inner
Startup Cost	4.45
Total Cost	4.45
Plan Rows	1
Plan Width	117
Actual Startup Time	0.017
Actual Total Time	0.017
Actual Rows	6
Actual Loops	1
Hash Buckets	1024
Hash Batches	1
Original Hash Batches	1
Peak Memory Usage	1

While all of these attributes may not be candidates for features (i.e may not impact execution time), some do. We can use the knowledge of SQL Query processing to select relevant features and determine the Best Fit Model as described earlier(Section 4.1). In this set of experiments, we are limiting ourselves to the feature 'Actual Rows' (i.e, Input cardinalities). In the case of binary operator nodes (such as Nested Loop), we are considering both left and right input cardinalities. Please note that a) the proposed feature set should not be considered complete as it may not capture all the properties that impact execution time, we shall review these set of features from time to time as we make progress in this work. b) More number of features correspond to increased training time and since the computation of Best Fit Model is exponential to the number of features, we should select as few features as possible to make the computations tractable.

To ensure that the trained model is not overfitting the data, we have generated training and testing queries do not contain identical instances of same query (i.e, Same query with same selectivities is not allowed).

Model	Minimum	Maximum	Mean
Optimizer	0.032	47.68	8.33
Our model	0.003	2.61	0.53

The benefits of modeling

6 Related work

Query-plan-level predictions have recently been studied [1]. In [1], authors consider plan-level query performance prediction for the following *static* query workloads: the TPC-DS query benchmark and a query workload obtained from a customer database. They

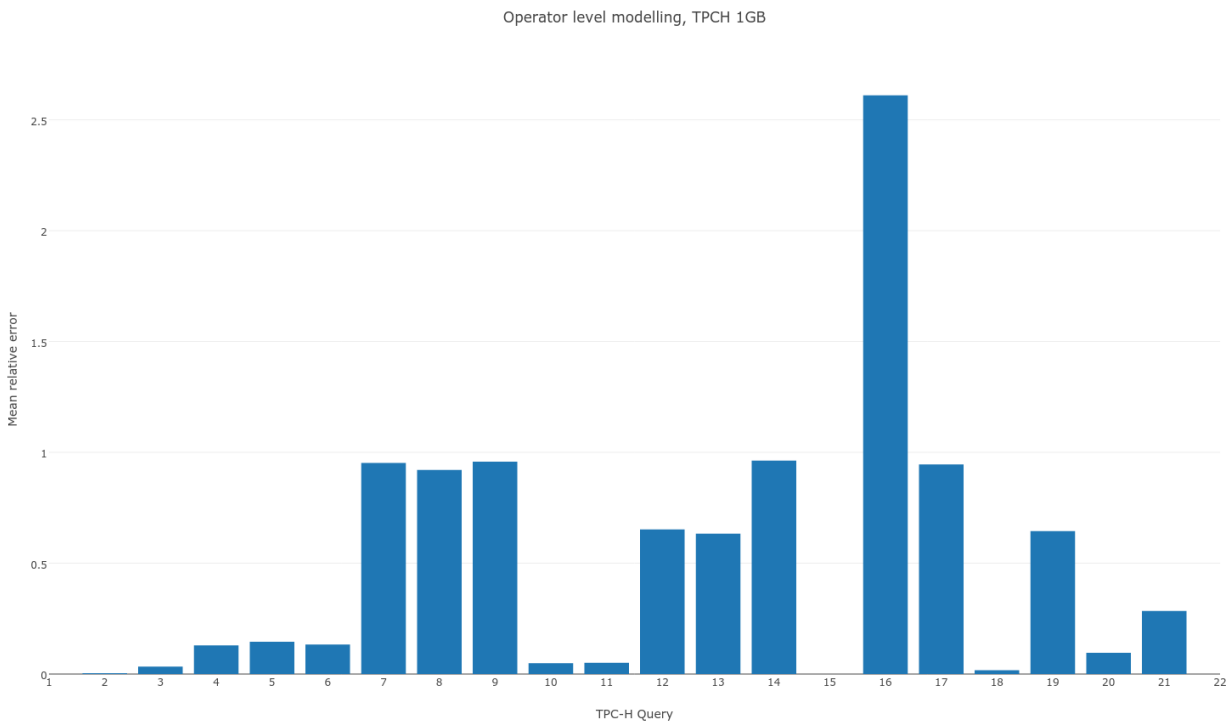


Figure 1: Operator-level modeling ,Errors by Template (1GB)

report that they can predict individual query execution times within 20% of the actual time for 85% of their test queries. However their approach produces poor estimates for queries which are *different* from training queries. The ability to adapt to dynamic workload is crucial for the cost model to continuously adapt the changes in the underlying databases and queries.

In [3, 2] author's proposed an approach which learns at a finer granularity than at plan level. They use statistical models such as SVM and Linear regression to learn the behavior of every operator. They have shown by that learning at this level, the model can now recognize queries which can differ largely from the queries seen during training. They have only considered 14 out of 22 templates and claimed that they were able to obtain 41% i.e., an MRE of 0.41. We are different from their approach in 2 ways a) we consider learning at *implementation level* i.e, for the operator *sort*, to implement it optimizer has a choice among multiple sorting methods such as quick sort ,merge sort top-N heapsort and external sort etc. No previous approach distinguishes the implementation methods of an operator which is clearly incorrect because there's a substantial difference between resource consumptions of quick-sort and external sort as the former is CPU intensive (consumes relatively less time) and while the latter is IO intensive (consumes much more time). b) we consider multiple learning models for every implementation and select the one which fits the best. This approach is often used when the relation between features and target is unclear.

In [8], author's have done an analytical cost modeling where in they compute CPU and IO time individually by obtaining the number of operations performed and number of pages(sequential & random) accessed respectively. They have also proposed a calibration phase where the cost model can account for the hardware changes which is absent in default Postgres model. In previous unpublished work by Phankuri, they have devised methods to compute random page cost which was left out in the work[8]. While this kind of approach seems natural, there's inherent ambiguity in knowing the number of pages accessed. Even after knowing all this, there's no certain way we can compute IO time as cost model can never know whether a page is sequentially accessed or randomly accessed which impacts final cost computed heavily as there is a magnitude of difference in access times of sequential and random pages. Finally, there has also been work

on query progress indicators [9]. Query progress indicators provide estimations for the comple- tion degrees of running queries. Such studies assume that the work done by individual query operators are transparent, i.e., externally visible. While these studies are also related to query execution performance, they do not provide predictions for the execution time of queries.

7 Conclusions and Future Work

In this work, we have presented learning based approach at a finer granularity than whats proposed earlier in the literature. By learning at operator level we have shown that the model can generalize to the queries which are unseen during the training because of the fundamental property that execution time is upper bounded by execution time of individual operators present in the plan tree. There remains a lot of work to be done. Currently feature set is limited, extending it will definitely improves accuracy and generality. However since the best-fit algorithm is exponential to number of features, it remains as a challenging work.

Also, interaction that happen within query makes accurate execution time prediction hard. For example, pipelining speeds up the query processing and is the primary source of over-estimation. Capturing these in the cost model is a challenging and has got the attention in recent years [6]. As mentioned earlier, we are not currently not addressing prediction in case of concurrent query execution. There is already some promising work in addressing this problem [7, 10]. It remains as an interesting future work to see how we can extend the features so that we can capture the execution time behavior under concurrent workloads.

References

[1] A. Ganapathi, H. A. Kuno, U. Dayal, J. L. Wiener, A. Fox, M. I. Jordan, and D. A. Patterson. Predicting multiple metrics for queries: Better decisions enabled by machine learning. In ICDE, 2009.

[2] Li, Jiexing, et al. "Robust estimation of resource consumption for sql queries using statistical techniques." Proceedings of the VLDB Endowment 5.11 (2012): 1555-1566.

[3] Akdere, Mert, et al. "Learning-based query performance modeling and prediction." Data Engineering (ICDE), 2012 IEEE 28th International Conference on. IEEE, 2012.

[4] S. Guirguis, M. A. Sharaf, P. K. Chrysanthis, A. Labrinidis, and K. Pruhs. Adaptive scheduling of web transactions. In ICDE, 2009.

[5] Harish, D., Pooja N. Darera, and Jayant R. Haritsa. "Identifying robust plans through plan diagram reduction." Proceedings of the VLDB Endowment 1.1 (2008): 1124-1140.

[6] Wu, Wentao, et al. "Towards predicting query execution time for concurrent and dynamic database workloads." Proceedings of the VLDB Endowment 6.10 (2013): 925-936.

[7] Ahmad, Mumtaz, et al. "Modeling and exploiting query interactions in database systems." Proceedings of the 17th ACM conference on Information and knowledge management. ACM, 2008.

[8] Wu, Wentao, et al. "Predicting query execution time: Are optimizer cost models really unusable?." Data Engineering (ICDE), 2013 IEEE 29th International Conference on. IEEE, 2013.

[9] Chaudhuri, S., Narasayya, V., and Ramamurthy, R. Estimating progress of execution for SQL queries. In Proceedings of the 2004 ACM SIGMOD international Conference on Management of Data

[10] Ahmad, Mumtaz, et al. "Qshuffler: Getting the query mix right." Data Engineering, 2008. ICDE 2008. IEEE 24th International Conference on. IEEE, 2008.

[11] Makridakis, Spyros, Steven C. Wheelwright, and Rob J. Hyndman. Forecasting methods and applications. John Wiley, 2008.

[12] TPC-H benchmark specification, http://www.tpc.org/tpch/

[13] R. Othayoth and M. Poess, The making of tpcds, in VLDB 06: Proceedings of the 32nd international conference on Very large data bases. VLDB Endowment, 2006, pp. 10491058

[14] Scikit-learn: Machine Learning in Python, Pedregosa et al., JMLR 12, pp. 2825-2830, 2011.

[15] Du, Weimin, Ravi Krishnamurthy, and Ming-Chien Shan. "Query optimization in a heterogeneous dbms." VLDB. Vol. 92. 1992.

[16] Xiong, Pengcheng, et al. "ActiveSLA: a profit-oriented admission control framework for database-as-a- service providers." Proceedings of the 2nd ACM Symposium on Cloud Computing. ACM, 2011.