Re-optimization for Multi-objective Cloud Database Query Processing using Machine Learning

Chenxiao Wang, Zach Arani, Le Gruenwald, Laurent d'Orazio, Eleazar Leal

RE-OPTIMIZATION FOR MULTI-OBJECTIVE CLOUD DATABASE QUERY PROCESSING USING MACHINE LEARNING

Chenxiao Wang, Zach Arani, Le Gruenwald, Laurent d'Orazio and Eleazar Leal

Department Computer Science, University of Oklahoma, USA.

ABSTRACT

In cloud environments, hardware configurations, data usage, and workload allocations are continuously changing. These changes make it difficult for the query optimizer of a cloud database management system (DBMS) to select an optimal query execution plan (QEP). In order to optimize a query with a more accurate cost estimation, performing query re-optimizations during the query execution has been proposed in the literature. However, some of there-optimizations may not provide any performance gain in terms of query response time or monetary costs, which are the two optimization objectives for cloud databases, and may also have negative impacts on the performance due to their overheads. This raises the question of how to determine when are-optimization is beneficial. In this paper, we present a technique called ReOptML that uses machine learning to enable effective re-optimizations. This technique executes a query in stages, employs a machine learning model to predict whether a query re-optimization is beneficial after a stage is executed, and invokes the query optimizer to perform the re-optimization automatically. The experiments comparing ReOptML with existing query re-optimization algorithms show that ReOptML improves query response time from 13% to 35% for skew data and from 13% to 21% for uniform data, and improves monetary cost paid to cloud service providers from 17% to 35% on skewdata.

KEYWORDS

Query Optimization, Cloud Databases, Machine Learning, Query Re-optimization.

1. INTRODUCTION

One key difference between query optimization in cloud databases and in conventional databases is that query optimization in cloud databases seeks to reduce the monetary cost paid to cloud service providers in addition to the query response time [32]. However, the time and monetary costs needed to execute a query are estimated based on the data statistics available to the query optimizer at the moment when the query optimization is performed. These statistics are often approximate, which may result in inaccurate estimates for the time and monetary costs needed to execute the query [1, 2]. Thus, the query execution plans (QEPs) generated by the query optimizer based on those statistics before the query execution may not be the best.

One approach that can be applied to address the above issue is adaptive query processing [3]. This strategy consists in not executing queries as a whole at one time, but instead dividing the execution of each query into multiple stages and then re-running the query optimizer between each stage. By doing this, the query optimizer can collect more accurate statistics between stage executions, which may allow for changing the QEP at runtime, thus possibly improving query performance [1, 4]. Operators that do not rely on the completion of others are grouped together and such groups are called “Stages”. For example, if a query plan has a JOIN operator, its left and right sides are each executed in a separate stage. After the completion of each stage of the QEP,

the data statistics are updated, so that the query optimizer can make use of the latest statistics to generate improved (i.e. re-optimized) QEPs for those stages that remain to be executed. As a result of query reoptimization, the QEPs of the stages that have not yet been executed may change because the operators in these QEPs might be replaced by others, or because any stage might be re-scheduled to run on a different machine. Such changes in QEPs might produce different query response times and different monetary costs. However, calling the query optimizer multiples times during query execution has an associated time overhead, which in turn produces additional monetary costs. For this reason, it is desirable to re-optimize a query only if the cost improvements of the re-optimized QEP over the original QEP can offset the cost incurred in calling the optimizer multiple times.

At any given stage of the execution of a query, deciding if a re-optimization will likely bring performance improvements is not an easy task. In early work [3], such a decision is made by rule based heuristic. Several check points are placed manually between certain types of operators. The difference between the estimated cost and the actual cost of executing a query after a check point is examined. If the difference exceeds a pre-defined threshold, then a re-optimization takes place. The problem of this technique is that the rules for placing check points and the threshold are fixed. Due to the dynamic of cloud environments, timing of re-optimization decided by this technique is not accurate enough to reduce the query's execution time. Our early work [5] presented a query processing algorithm that performs query re-optimization after the completion of each stage based on the technique proposed in [1]. However, our work [5] shows that many of these re-optimization calls produced no change in the underlying QEP, which means that the query re-optimization was performed unnecessarily. This was because the stages were not aligned with the best timing to apply the re-optimization. For example, after running the example Query 1 given in Figure 1, we observed that out of the 10 times that the optimizer was called for re-optimization during the execution of this query, only 2 out of these calls changed the QEP for the remaining stages; therefore, the majority of the re-optimization calls produced no improvement on either the time or the monetary cost of running this query. The details on these findings are reported in Section 3.

Naturally, calling the re-optimization routine unnecessarily increases both the query response time and monetary cost. The problem therefore lies in determining the most appropriate time when to call for re-optimization, and in determining those occasions where re-optimization can negatively impact query performance. To address this problem, this paper presents a new machine learningbased algorithm for query re-optimization in the cloud. The key idea behind this algorithm consists in using past query executions in order to learn to predict the effectiveness of query reoptimizations, and this is done with the purpose of helping the query optimizer avoid unnecessary query re-optimizations for future queries.

```
SELECT R.p_id, R.p_name, R.sc, S.p_hr
FROM (SELECT p_id, p_name, AVG(p_bp) AS sc
FROM patient GROUP BY p_id, p_name) AS R
JOIN (SELECT p_id, p_hr
FROM patient
WHERE UDF(p_id,p_hr) > 80
) AS S
ON R.p_id = S.p_i
```

Figure 1. Query 1

While machine learning has been used to improve query processing in a number of recent works, such as [6, 7, 8, 9, 10, 11], to the best of our knowledge, it has not been used to avoid unnecessary query re-optimization calls in adaptive query processing. Among the issues that need to be addressed when using machine learning for this purpose are the following. The first one consists in the many features that influence query cost estimations, such as selectivity, cardinality, min and max values of a column, most frequent value of a column, histogram, etc. The difficulty here lies in selecting the most appropriate feature subset out of all these features. The second issue consists in the large number of possible machine learning models. Supervised learning algorithms like Random Forest [12], Neural Network [13], and Support Vector Machine [14] are widely used but need to be studied carefully for our purpose. The third issue is about the collection of the historical data on the selected subset of features that is needed to train the prediction model constructed using the selected machine learning algorithm. The fourth issue consists in measuring the effectiveness of the learning algorithm. Some works such as [8, 15, 16] show the learning algorithm is effective for their own purposes, for example to improve cost estimation, but actually, none of them demonstrates that they are effective in actual query execution performance. Our proposed technique addresses all these issues.

In this paper we make the following contributions:

- We present a novel machine learning-based query re-optimization algorithm for a relational DBMS in the cloud [9] to optimize query response time and monetary costs. We discuss the feature selection, training data collection, machine learning model selection, and integration of the selected machine learning model into query re-optimization.

- We present a comprehensive experimental study evaluating the accuracy of different machine learning models, the query response time and monetary costs of the proposed query re-optimization algorithm when operating under different machine learning models, and comparing the proposed query re-optimization algorithm with existing query re-optimization algorithms.

The remaining of this paper is organized as follows: Section 2 discusses the related work; Section 3 provides the details of work on query re-optimization and its results that motivate this research; Section 4 presents the proposed machine learning-based query re-optimization algorithm; Section 5 discusses the experimental performance evaluation model and the results; and finally Section 6 presents the conclusions and discusses future research directions.

2. RELATED WORK

The problem of query re-optimization has been studied in the literature. In early days, heuristics were used to decide when to re-optimize a query or how to do the re-optimization. Usually, these heuristics were based on cost estimations which were not accurate at the time when query reoptimization takes place. Besides that, sometimes, a human-in-the-loop was needed in order to analyze and adjust these heuristics [2, 4, 10, 17]. These add additional overheads caused by query re-optimization to the overall performance of queries. Recently, learning techniques have been introduced to improve query optimization [6, 7, 8, 9, 10, 11]. Feedback from the executions of past queries can be used to guide the query optimizer to produce better QEPs, so that the performance of future queries can be improved as a result. In this section, we review the work in the two related areas of adaptive query optimization and machine learning-based query optimization.

2.1. Adaptive Query Optimization

The idea of adaptive query optimization is that data statistics such as selectivity, cardinality, min and max values, and histograms are monitored during query execution. The execution of a query can be paused whenever the algorithm decides to re-optimize the query. Then the new data statistics and intermediate results are used by the query optimizer to generate a new QEP and the query execution continues following the new QEP. In the works presented in [18, 19], the checkpoints are manually set between certain operators of a QEP. These set points are placed at certain positions such as before or after a join operator based on some rules, but whether or not a re-optimization call should be triggered is still based on whether the difference between the actual and estimated costs of executing a query exceeds a threshold. In these works, the threshold is fixed at 20% and applied to all the check points. Re-optimizations that take place at these check points are necessary but are still not accurate. For example, in some application scenarios, reoptimization should be triggered even the difference is less than 20%. Using a fixed threshold is difficult to adapt the re-optimization decision to all application scenarios. The work in [1] proposes a query optimization method where the query is re-optimized multiple times during its execution based on stages. Every time one stage is finished, the query is re-optimized. We implemented this idea and found that many re-optimizations are wasted [5], which we report in detail in Section 3. Adaptive query optimization suffers from the following drawbacks: 1) It is hard to decide when a query should be re-optimized. Cost estimation can be a heuristic for the optimizer to make decision but it is not accurate. Since re-optimization has a considerable overhead, re-optimizing a query during its execution when such re-optimization leads to no QEP changes can negatively impact the overall performance; and 2) Adjusting the QEP is difficult, especially in a cloud environment [20]. Some works such as [10, 21] manually adjust the QEP, but these works suffer from a large search space for exploring the best adjustment for the current QEP. In another work, Tukwila [18], re-optimization is decided by a heuristic-based rule. After the execution of each operator, the actual data statistics are collected and compared with the previous estimated data statistics. If the difference between these two data statistics exceeds a threshold, a query re-optimization is conducted. As the rule is heuristic and does not adjust adaptively regarding how the data or hardware changes, the rule is not always useful which then generates some unnecessary re-optimizations.

2.2. Machine Learning-Based Query Re-Optimization

Machine learning techniques have been used recently in query optimizations [9, 15, 22, 23, 24] for different purposes. In earlier works, Leo [23] learns from the feedback of executing past queries by adjusting its cardinality estimations over time, but this algorithm requires human-tuned heuristics, and still, it only adjusts the cardinality estimation model for selecting the best join order. More recently, the work in [24] presents a machine learning-based approach to learn cardinality models from previous job executions and these models are then used to predict the cardinalities in future jobs. In this work, only join orders, not the entire query, are optimized. In [21], the authors examine the use of deep learning techniques in database research. Since then, reinforcement learning is also used. The work in [15] proposes a deep learning approach for cardinality estimation that is specifically designed to capture join-crossing correlations. SkinnerDB [22] and ReJoin [9] are two other works that use a regret-bounded reinforcement learning algorithm to adjust the join order during query execution. Also, the algorithm proposed in [28] uses variational auto encoder to make an accurate estimation of query execution time but still, this algorithm does not perform end-to-end query processing. None of these machine learning-based query optimization algorithms is designed for predicting whether a query reoptimization is beneficial in terms of query response time and monetary cost, which is the goal of our proposed algorithm. In our work, we use a machine learning model that learns from the past query re-optimizations to do the re-optimization prediction and integrate this model into the

endto-end query processing to achieve improvements on query response time and monetary costs. To select a proper machine learning model, we investigate what kinds of features that can be used to build such a model, how to generate training queries and training databases, and how major existing machine learning models perform in terms of prediction accuracy, query response time, and monetary costs.

3. QUERY RE-OPTIMIZATION

To provide more details to support our motivations for the work proposed in this paper, in this section we report the findings we obtained when performing query re-optimization without using machine learning. In our previous work [5], after a query is submitted to the DBMS, a regular query optimizer first generates an initial QEP. Then this QEP will be divided into stages and executed by the execution engine stage by stage. After a stage is finished, the data statistics are updated. These statistics include the cardinality, selectivity, and max and min values for each attribute in each database table. By updating these statistics, the estimation of the resulting data size used in the next stages is updated accordingly. The rest of the stages in the QEP are also sent to the query optimizer for re-optimization using the updated statistics. Also, in that system, multiple machines with different hardware configurations are used in parallel to execute query operators. Those machines are referred to as containers in this paper. Executing query on different containers results in different query response time and monetary cost, and the best QEP selected needs to take both into consideration. In order to do so, we use the Normalized Weighted Sum Model developed in [25] to select the best plan. In this model, every possible QEP alternative is rated by a score that combines both the objectives, query response time and monetary cost, with the weights defined by the user and the environment for each objective, and the user-defined acceptable maximum value for each objective. The following function is used to compute the score of a QEP:

$$A_i^{WSM-score} = \sum_{j=1}^{n} w_j \frac{a_{ij}}{m_j}$$

a_{ij} is the value of QEP alternative i (QEP_i) for objective j, m_j is the user-defined acceptable maximum value for objective j, and W_j the normalized composite weight of user and environment for objective j which is defined as follows:

$$w_j = \frac{uw_j * ew_j}{\sum(uw * ew)}$$

Where uw_j and ew_j describe the weight of the user and the environmental weight for objective j, respectively. These weights are user-defined. Since the different objectives are representative of different costs, the model chooses the alternative with the lowest core to minimize the costs.

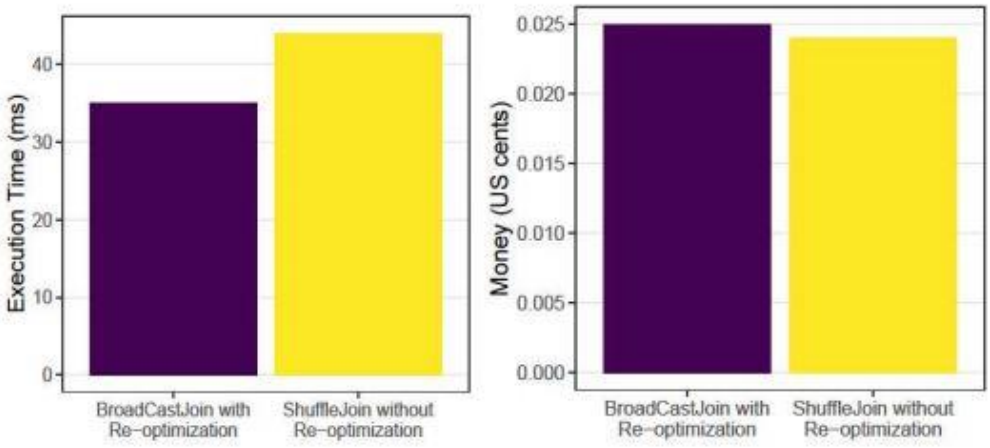

Figure 2. Impacts of physical operators on time and monetary costs for execution of Query 1

We conducted experiments comparing the query performance resulted from using query reoptimization vs. using no query re-optimization. In the experiments, 1200 queries were created using the query templates presented in [1] (Query 1 shown in Figure 1 is a query created from one of these templates). The results show that using re-optimization has approximately 20% improvement on average in terms of the overall time cost over using no re-optimization, while the monetary costs of the two approaches are close, with only a 4% difference. This increase of monetary cost is due to the fact that the more powerful containers that are selected to run the queries are the containers for which the cloud providers charge more hourly to run a query. However, we noticed that a large number of query re-optimizations is unnecessary. An unnecessary re-optimization for a QEP occurs when the QEP does not change after the reoptimization is performed. Note that in these experiments, after a stage in a QEP is executed, a re-optimization is automatically conducted regardless of whether the data statistics have changed after the stage is executed, resulting in many re-optimizations that are unnecessary. For example, the experiments show that when Query 1 is executed, 8 of the 10 query re-optimizations are unnecessary. The only 2 necessary re-optimizations happen after the subquery is executed. Except for those, the QEPs after the TableScan or Aggregate operator is executed do not change at all after the re-optimizations. Performing re-optimization incurs overheads and unnecessary reoptimizations increase query response time and monetary costs. In these experiments, we observed that nearly 60% of the query re-optimizations are unnecessary and performing one query re-optimization costs around 0.5% of the total query response time. Thus, avoiding unnecessary query re-optimizations is important to further improve the performance of query execution on both time and monetary costs.

If a query is re-optimized only when changes to its QEP after the re-optimization can be guaranteed, then there will not be any unnecessary re-optimization. In order to detect such changes, in the next section, we present a new machine learning-based re-optimization technique to predict if a QEP will change after a re-optimization based on the historical query execution data is performed, and conduct a re-optimization for the QEP only when such a change is predicted.

4. PROPOSED MACHINE LEARNING - BASED QUERY RE - OPTIMIZATION

In this section, we describe how machine learning is used in our proposed algorithm, ReOptML, to predict when a query re-optimization is beneficial for the performance of queries. We first

present an overview of our approach (Section 4.1), then we present its four parts: the feature selection (Section 4.2); the training data collection (Section 4.3); the machine learning model selection (Section 4.4); and the query re-optimization algorithm that integrates with the selected machine learning model to optimize query response time and monetary cost (Section 4.5).

4.1. Overview

Figure 3 shows the major steps in query processing when ReOptML is incorporated for query reoptimization. First, the optimizer receives a query and records the current data statistics. Then the query is compiled into a QEP with the stage information. The first stage in the QEP is executed and removed from the QEP. During the execution, the data statistics are monitored and updated. After the execution of the first stage, these updated data statistics are compared with the current data statistics that were recorded before the stage was executed. The machine learning model is used here to take the difference between the current data statistics and the new data statistics as input, and produce the re-optimization decision ("YES" or "NO") as output. The query is reoptimized if the decision is "YES" and the current first stage in the new QEP after the reoptimization is executed; otherwise, if the decision is "NO", the QEP remains the same and its next stage is executed. This procedure continues until there is no stage left.

4.2. Feature Selection

In this section, we discuss what statistics (features) we collect to train our machine learning model.

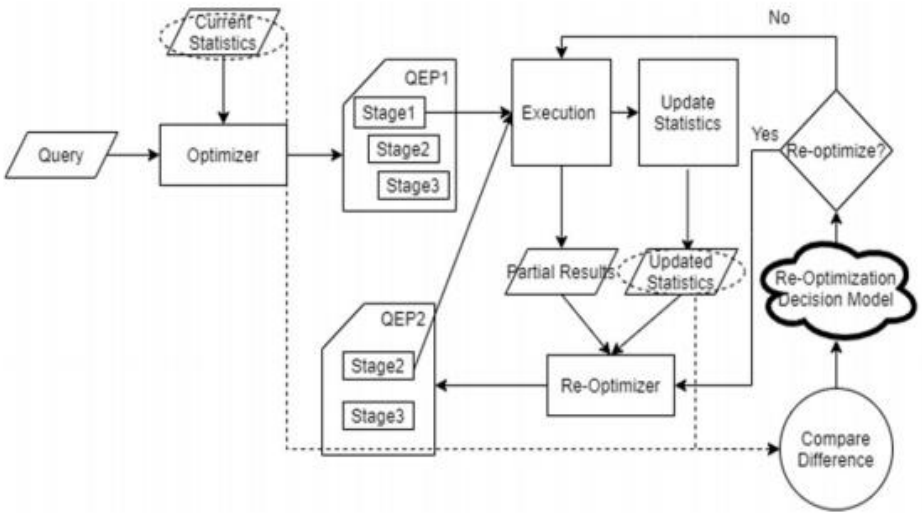

Figure 3. Query processing with machine learning-based query re-optimization

A change in a QEP after a re-optimization implies that the re-optimization is beneficial. We define such a change to be one of the following types: 1) changes in the physical operator types, 2) changes in the number of containers, or 3) changes in the types of containers. This means that if any of these three types of changes occurs, then the re-optimization should be allowed to take place.

A change in the physical operator types means that if there exists any physical operator in the current QEP that is different from the physical operators in the previous QEP, then the QEP has

changed. For example, in our previous experiments, the change in the physical operator from Shuffle Join to Broadcast Join is defined as a change in the physical operator types. This change highly influences query execution time. Thus, by detecting such changes in the QEP after a reoptimization, this re-optimization will probably be beneficial, and thus the re-optimization will be applied if a similar situation is encountered.

A change in the number or types of containers means that the total number of containers used to execute the current QEP is different from that of the previous QEP. Such changes are also called changes in the degree of parallelism. For example, the TableScan operator is assigned to four containers before the re-optimization and uses only three containers after the re-optimization. This change highly influences the monetary cost of query execution. Thus, re-optimizations are useful if such changes are detected. Similarly, a change in the types of containers means that after the re-optimization, the operators are assigned to different types of containers from the ones that the operators were assigned to before the re-optimization. These new containers may be more or less powerful than the old ones. Detecting such changes may influence the monetary cost as well.

These three types of changes occur whenever the estimated data size has also changed. This is because the query optimizer uses these estimations to decide how to execute the query and how many containers should be used. Thus, in order to tell whether the re-optimization will be beneficial, we use the data features that are relevant to the changes in data size estimation.

Assume that in the current DBMS, there exist the C1,C2,….,Cn columns in all the tables. The differences in the selectivity (DIFF_SELECTIVITY), in the number of distinct values (DIFF_NDV) and in the histograms (DIFF_HISTOGRAM) of each column before and after a stage is executed are used as the data features in the training data used for prediction as shown in Table 1. The binary value YES/NO is used as the predicted class in the training data, where YES means that the re-optimization is predicted to be useful and NO otherwise. Many works show that the selectivity, number of distinct values and the histogram influence the data size estimation [15, 17, 26]. Thus, the differences in these three features before and after a stage is executed result in changes in the data size estimation of the intermediate results. Hence, they become relevant in deciding the effectiveness of re-optimization.

4.3. Model Training

First, we generate queries for model training by running random queries generated from all 22 types of queries in the TPC-H benchmark [27] on our system and recording the data statistics, which are the values of the features we have selected in Section 4.2 above. This way the prediction model can be applied to all queries. If re-optimization is only for the costliest/most representative queries, then in this first step, the training data should be collected from running only the random but most costly/representative queries.

Figure 4 shows the procedure for the training data collection. In order to better explain in detail how the training data is collected, we demonstrate an example of executing the following example Query 2 shown in Figure 5.

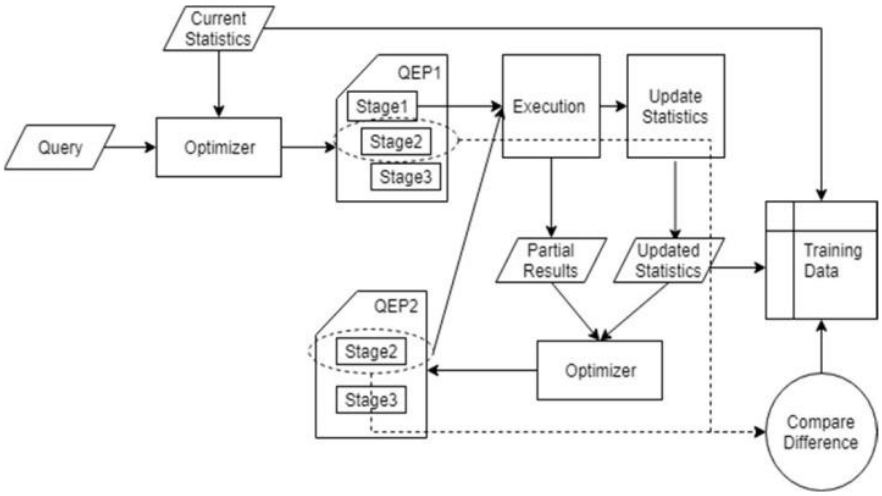

Figure 4. The procedure for collecting training data

```
SELECT Department, COUNT(Name)
FROM STUDENT
GROUP BY Department
WHERE Grade <= 'C';
```

Figure 5. Query2

After the query is submitted, we record the current data statistics gathered from the system logs. These current statistics are called Statcurr. Then, the query is sent to the optimizer to generate a QEP. This QEP includes the stage information and the nodes on which these stages will be executed. Figure 6 shows the QEP generated by the query optimizer for Query 2. In Figure 6, each node stands for a different query operator. The arrows indicate the dataflow between the operators. The QEP is divided into stages, each of which is denoted by a rectangular. TS, SOR,

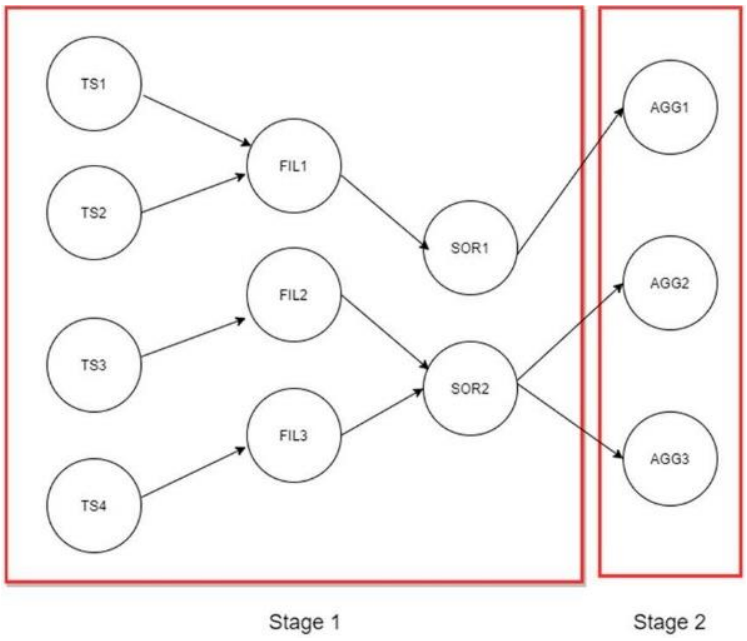

Figure 6. QEP divided into different stages generated by the query optimizer for Query 2

FIL, and AGG, standing for TableScan, Sort, Filter and Aggregate operators, respectively. In a cloud database system, as data are distributed among different containers, the subscripts distinguish the same operators that are executed in parallel on different data on different containers.

Then, Stage 1 is sent to the query execution engine. During the execution, we update the data statistics using the method presented in [1]. In this method, data statistics are collected during the execution and updated after the operators in one vertex finish. The vertex is similar to our stage. We call these updated statistics $Stat_{update}$. Since these statistics are collected from the actual running query, $Stat_{update}$ is more accurate than $Stat_{curr}$ which is obtained from the estimation. The difference between $Stat_{update}$ and $Stat_{curr}$, is called $Stat_{diff.}$ $Stat_{diff}$ includes the values of the features used as the training data. For example, the current selectivity and the updated selectivity of column A are 0.5 and 0.1, respectively, then the difference 0.4 is added as the value of the DIFF_SELECTIVITY feature in the training dataset. This process is applied to all the features. The selected features are shown in Table 1.

Table 1. List of Selected Features

DIFF_SELECTIVITY(C1)
DIFF_SELECTIVITY(C2)
DIFF_SELECTIVITY(Cn)
DIFF_NDV(C_1)
DIFF_NDV(C_2)
DIFF_NDV(C_n)
DIFF_HISTOGRAM(C_1)
DIFF_HISTOGRAM(C_2)
DIFF_HISTOGRAM(C_n)

If the re-optimization is predicted to be beneficial, the QEP is then re-optimized using the updated data statistics. Following this, the next stage (Stage 2) is executed based on the new QEP. The process is then repeated for the rest of the stages. In this example, Stage 2 is possibly changed. At this point, Stage 2 after the re-optimization is compared to the Stage 2 before the re-optimization to observe any potential changes.

4.4. Machine Learning Model Selection

There exist a lot of machine learning models, but we need to choose a model that has a high accuracy in predicting if a re-optimization is beneficial, and incurs smaller overheads than the amounts of query execution time and monetary cost that it can save by avoiding unnecessary reoptimizations. The overheads incurred by a prediction model include the time to train the model (training time) and the time to apply the trained model for prediction (prediction time). In our case, as the model is trained offline, we are only concerned about the prediction time overhead. Applying different models trained by different learning algorithms may have different prediction time overheads. For example, applying a model created by a Neural Network learning algorithm [13] may have a different prediction time overhead comparing with the prediction time overhead when applying a model trained by a Random Forest algorithm [12]. This overhead may be different even when applying different models that are trained by the same learning algorithm.

Algorithm: Query Processing with Machine Learning Based Re-Optimization (ReOptML)

INPUT: SQL query
OUTPUT: The query result set of the input query

```
1.  Old_Statistics = get current data statistics
2.  Result = Φ
3.  Old_QEP = generate_QEP (old_statistics, result)
4.  result = execute the first stage in Old_QEP and remove it
5.  Update the data statistics
6.  New_Statistics = get current data statistics
7.  Diff_Statistics = compute difference between Old_Statistics and New_Statistics
8.  while Old_QEP≠ Φ
9.     decision = RunPredictiveModel(Diff_Statistics)
10.    if decision = 'YES'
11.       New_QEP = generate_QEP(new_Statistics, result)
12.     result = execute the first stage in New_QEP and remove it.
13.     Old_QEP = New_QEP
14.    else if decision = 'NO''
15.       result = execute the first stage in Old_QEP and remove it
16.    end if
17.    if QEP ≠ Φ
18.      update data statistics
19.      Old_statistics = New_statistics
20.      New_statistics = get current data statistics
21.      Diff_ftatistics<-compute difference of Old_statistics and New_statistics
22.    end If
23. end while
24. return result
```

Figure 7. Query processing algorithm with machine learning based re-optimization

For example, checking a Neural Network with 50 layers to derive a prediction is different from checking a Neural Network with 100 layers.

In this paper, we study three major supervised machine learning algorithms representing three different families of machine learning models: Neural Network [13], Random Forest [26], and Support Vector Machine (SVM) [14]. These algorithms have also been used in the recent works on applying machine learning to database research [21]. We compare these three algorithms based on their prediction accuracy and prediction time and monetary overheads. The comparison results are reported in Section 5.2 and Section 5.3.

4.5. Query Processing Using the Proposed Machine Learning-based Query ReOptimization Model (ReOptML)

In this section, we illustrate how the trained model is applied during the query execution and the details are provided in Algorithm 1 shown in Figure 7. In Lines 1 to 3 of Algorithm 1, the query is first optimized and converted to a QEP. This QEP is denoted as Old_QEP. At this time, the current data statistics are collected and stored as Old_Statistics. In Line 4, Stage 1 of the QEP is executed and removed from the QEP, and the intermediate results are saved. In Lines 5 and 6, the data statistics are updated and recorded as New_Statistics after the Stage 1's execution. In Line 7, the New_Statistics and Old_Statistics are compared and denoted as Diff_Statistics. From Line 8, while there are unfinished stages remaining in the Old_QEP, the Diff_Statistics is sent to the

machine learning model to predict whether the new re-optimization will be beneficial or not and the corresponding decision of "YES" or "NO" is returned. If the decision is "YES", then in Lines 10 to 13, the Old_QEP is re-optimized into the New_QEP by the query optimizer using the new data statistics and the intermediate results. The New_QEP replaces the Old_QEP and the first stage in the New_QEP is then executed. If the decision is "NO" in Line 14, then the re-optimization is skipped and the current first stage of the Old_QEP is executed without re-optimization. From Line 16 to Line 22, regardless of whether the QEP has been re-optimized or not, the data statistics are always updated and compared as the model requires the Diff_Statistics to make a decision before executing the next stage. This procedure repeats until all the stages have been executed. Then the query results are sent to the user.

5. PERFORMANCE EVALUATION

In this section, we first describe the hardware configuration and the procedure used to collect the training data. We then evaluate the machine learning models that predict whether the reoptimization is necessary to identify the best one to process queries. Finally, we compare the performance of processing queries using ReOptML with the performances of three competitive algorithms: query processing without re-optimization, query processing with re-optimization after each stage is executed and query processing with re-optimization after each check point.

5.1. Hardware Configuration

There are two sets of machines used in our experiments. The first set consists of a single local machine used to train the machine learning model and to perform the query optimization. This local machine has an Intel i5 2500K Dual-Core processor running at 3 GHz with 16GB DRAM. The second set consists of 10 dedicated Virtual Private Servers (VPSs) that were used for the deployment of the query execution engine. Five of these VPSs, called small containers, have one Intel Xeon E5-2682 processor running at 2.5GHz with 1 GB of DRAM. The other 5 VPSs, called large containers, each has two Intel Xeon E5-2682 processors running at 2.5GHz with 2 GB of DRAM. The query optimizer and the query engine used in this experiment were modified from PostgreSQL 8.4 [28]. The data were distributed among these VPSs.

5.2. Performance of Different Machine Learning Models for Query Re-Optimization

Generating training queries and databases: In order to study the model accuracy of different models on different sizes of training queries, batches of different sizes of training queries are created. Each batch contains a different number of queries that have been executed and monitored on the same system with the algorithm implemented based on work [5]. The batch sizes are from 10,000 queries to 60,000 queries with an interval of 5,000 queries between them. Also, we prepare two databases for training: one database where all the tables are populated with uniform distributed data and the other database where all the tables are populated with zpfi distributed skew data. To simulate the real usage of a database management system, the tuples of all the database tables are randomly changed constantly. This means we continuously insert, delete, or update tuples and index columns of all the tables. The table structures remain unchanged, no new tables are created, and no current tables are dropped. After each query execution, multiple observations are gathered as discussed in Section 4.3. After executing each query, multiple reoptimizations are conducted and each observation is for one of the query re-optimizations. Then these observations are labeled manually according to whether the QEP has been changed after the re-optimization. An observation is labelled "YES" if the QEP has been changed after the reoptimization and "NO" otherwise.

Tuning model parameters: If the learning algorithm constructing the model has parameters, one important thing is to choose the best values for the parameters before training the model. For the Random Forest algorithm, we found three parameters influencing the model accuracy which are the number of trees, number of nodes in the tree, and random split option. In order to select the best values for the parameters for running this model, the Nested Cross Validation method [28] is used. This method first uses one round of cross validation to search for the best parameters' values and then applies another round of cross validation to test the true accuracy of the model. In our experiments we found that using the number of hidden layers as 600, the number of nodes in the tree as 200, and the random split option as YES provide the best accuracy of our Random Forest model.

Model Accuracy: Model accuracy reflects the overall success rate of predicting useful reoptimizations. We use 10-fold cross validation to test the accuracy of three models, Neural Network, Random Forest, and SVM. We also study the impact of different data distributions on the accuracy of the learning models. We populate the database tables with both the uniformly distributed data and skew data and the same queries are executed on both of them. Many traditional query optimizers, like PostgreSQL [29], assume that data is uniformly distributed, so if only uniformly distributed data is used, there are more chances that re-optimization has no effect at all. Skew data may cause wrong cost estimations and thus the QEP selected by the traditional query optimizer is far from optimal, thus re-optimization may be more useful when data is skew. We use skew data on purpose to see how model the accuracy and query execution performance are impacted.

As shown in Figure 8, as the number of queries increases, the accuracy increases as well. This is because as more observations were learned by the model, it is more capable of predicting beneficial re-optimizations. We find the accuracy among these three models are slightly different. Averagely, the Neural Network is near 70% accurate, while Random Forest and SVM are close to 75%. From the data distribution perspective, the models on the uniform data and on the skew data have slightly different accuracies with the average accuracy being within 5% difference of each other.

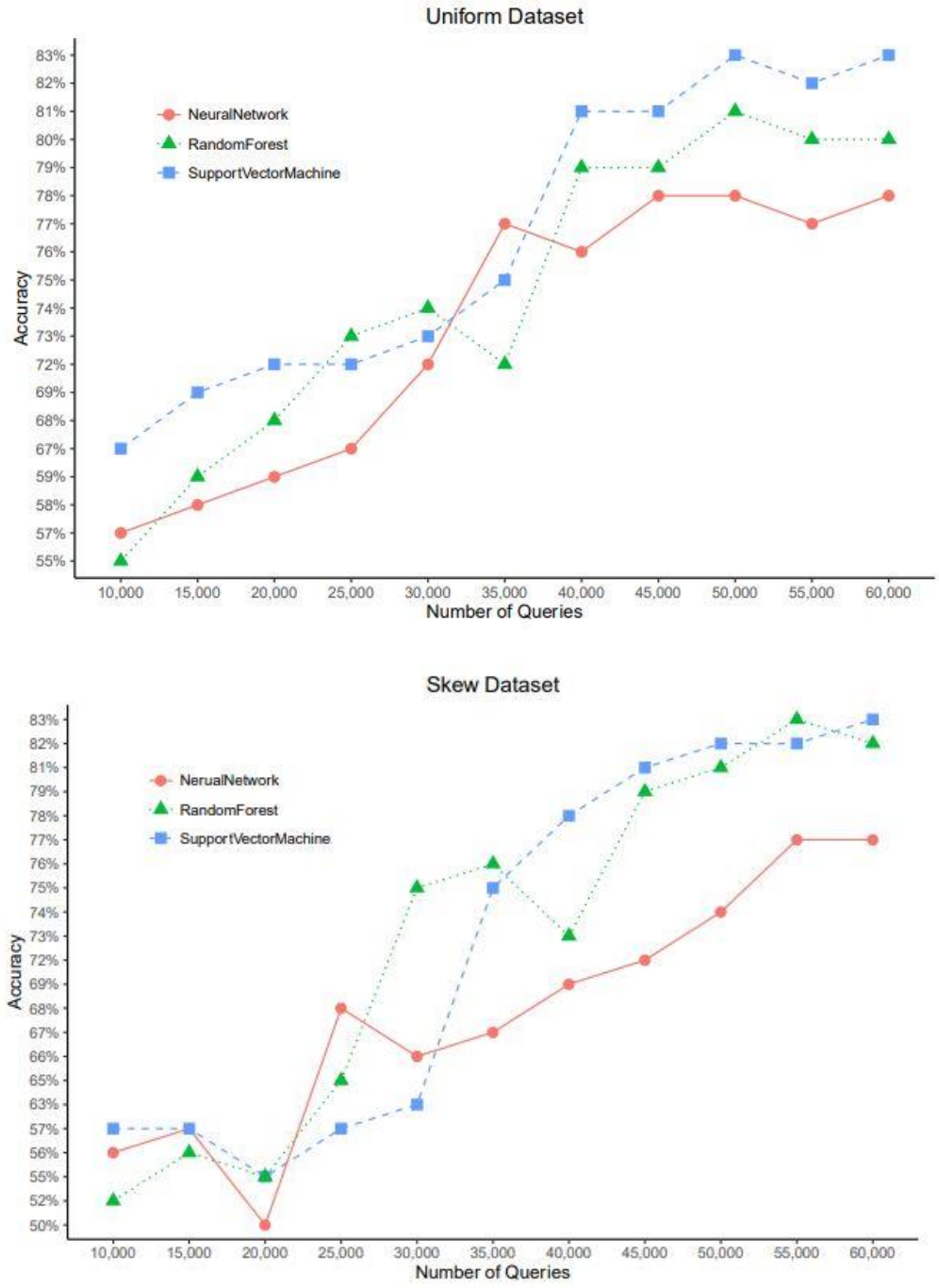

Figure 8. Model accuracy of three different machine learning algorithms that learn from queries executed on (a) uniform data and (b) skewed data

5.3. Performance Obtained When Applying Different Machine Learning Models for Query Re-Optimization to Query Processing

The model accuracy is close to each other as reported above; so to select which model should be used eventually, in this section, we evaluate these models in terms of performance on query execution when incorporating them into query processing as shown in Algorithm 1 in Figure 7 in

Section 4.5. We generate 100 query instances from each of the 22 TPC-H benchmark query types, totaling 2200 queries. On average, each query has 13 stages. These queries are executed and reoptimized based on the decisions made by these three models. Each QEP is evaluated with the same weight on time and monetary costs when the query optimizer selects the best QEP. This means we assume the users have no preference on time or monetary costs themselves. The actual time and monetary costs resulted from applying these three models are compared. To be fair, these queries are newly generated and not seen by any of these models during the model training process. Figure 9 shows the end-to-end query response time and monetary cost on executing the queries generated from all 22 query types of the TPC-H benchmark and these costs are summarized in Table 2. These results are averaged on running queries on both uniform and skew dataset.

From Table 2, we can see that SVM gives the best query response time. As shown in Figure 8, the three models have a very similar model accuracy. This means that the optimizer has a similar chance to perform useful re-optimizations by using any of these models. However, it takes different amounts of time to apply these models as we have discussed in Section 4.4. As these models are applied online during query execution, the overheads caused by using these models are added to the query response time. Thus, a small difference in this overhead may cause a significant difference in query response time, and thus is crucial to the users. From the monetary

Table 2. Average and Cumulative Query Response Time and Monetary Cost Using Three Different Machine Learning Models

	Neural Network	***Random Forest***	***SVM***
Average Query Response Time	36.2 sec	35.4 sec	31.5 sec
Cumulative Query Response Time of 2,200 queries of 22 query types	79,200 sec	77,880 sec	69,300 sec
Average Query Monetary Cost	0.070 ¢	0.071 ¢	0.069 ¢
Cumulative Query Monetary Cost of 2,200 queries of 22 query types	154.6¢	156.2 ¢	151.8 ¢

cost perspective, the amount of money to execute each query seems negligible when using any of the three models as shown in Figure 9 (b). However, this amount shown in this figure is just for one query execution, but in practice, tens of thousands of queries are executed for enterprise applications. This results in a large difference in cumulative monetary costs. Also, for each query type, the monetary cost has a larger variation than the query response time. This is because in our hardware configuration, a large container is charged 4 times of money more than a small container according to our price model. If an operator is assigned to a large container, it costs way more money to be executed but the time cost may be just a little bit less. Thus, the accumulative monetary cost varies a lot. Overall, SVM has the best prediction accuracy and query response time, and the second best monetary cost. Thus, in the following experiments, we select SVM as the machine learning model to be used in our proposed machine learning-based query reoptimization, ReOptML, and compare this algorithm against other query re-optimization algorithms. We select this model for comparison purposes only; we do not intent to suggest which model should be selected automatically as some QEPs may be executed faster but costs more money and vice versa, depending on the selected model.

5.4. Performance of Different Query Re-Optimization Algorithms

In this section, we compare the end-to-end query processing performances obtained when the following query re-optimization algorithms are incorporated into query processing: 1) our proposed algorithm (ReOptML); 2) the algorithm presented in [5] (denoted as ReOpt in our figures) where a query re-optimization is conducted automatically after the execution of each stage in the query is completed. We developed this algorithm based on the state of art works [1,31]; the algorithm proposed by Tukwila [18] (denoted as Tukwila), a well-known adaptive query re-optimization algorithm that triggers a re-optimization after an operator is executed if the difference between the estimated query cost and the actual query cost exceeds some threshold; and the baseline algorithm where queries are processed without any query re-optimization (denoted as NoReOpt).

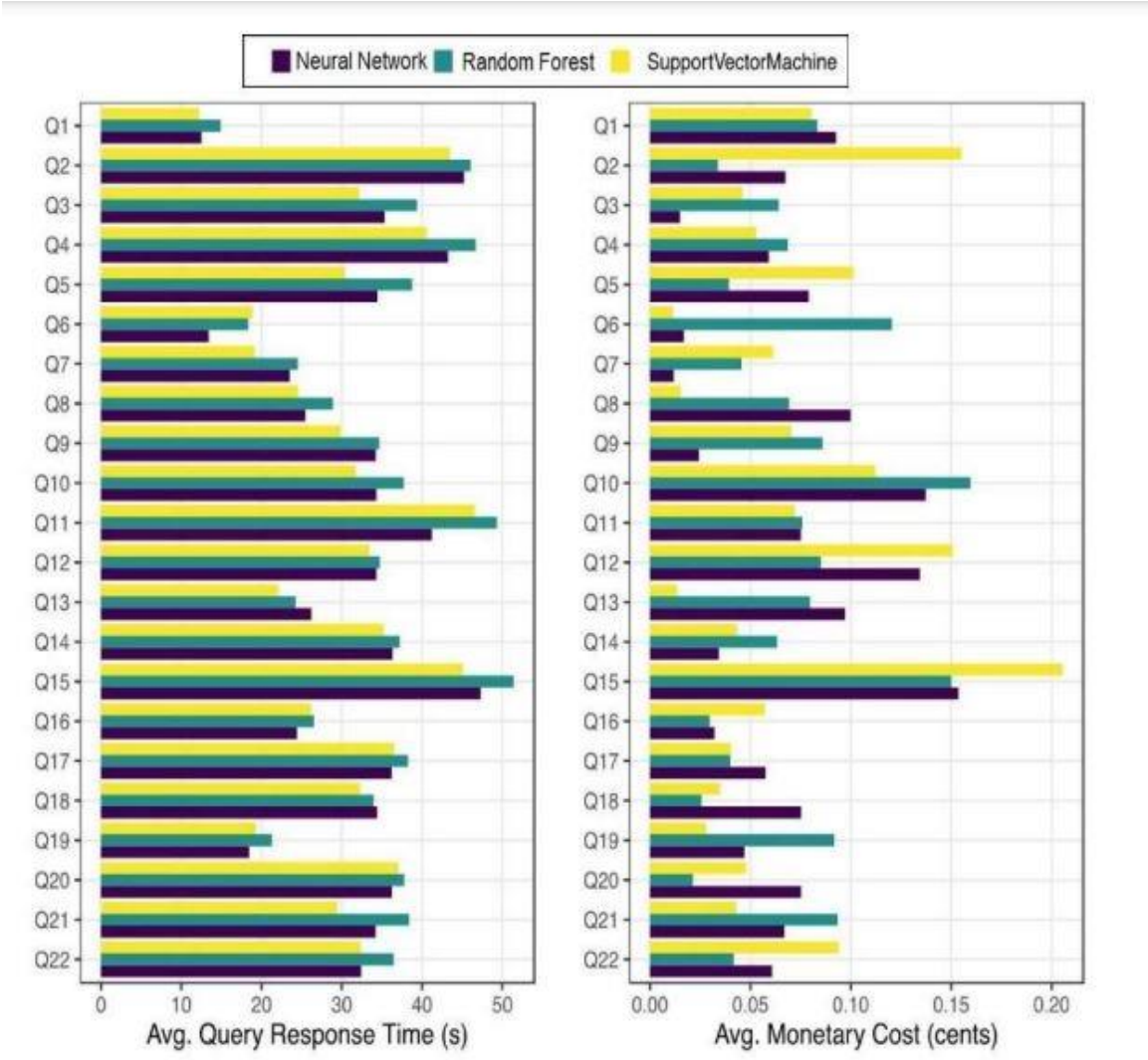

Figure 9. (a) and (b). Average response time and Average monetary cost of executing queries using three different machine learning models for query re-optimization

We launch 2200 queries with 100 queries being generated from each of the 22 TPC-H query types both on uniform and skew data. We compare the average query response time and

monetary cost. We report the query types that have large differences between ReOpt and ReOptML on average so that we can see with the help of machine learning, how much improvement can be obtained with re-optimization.

Skew Data: we compare our algorithm with Tukwila, NoReOpt, and ReOpt on skew data. The experimental results show that our algorithm performs the best both in terms of query response time and monetary costs. From Figure 10 (a), on average we see that ReOptML yields 13%, 22%, and 35% less query response time than ReOpt, Tukwila, and NoReOpt, respectively. From Figure 10 (b), on average we see that ReOptML spends 17%, 34%, and 35% less money than NoReOpt, ReOpt, and Tukwila, respectively.

The above results show that ReOptML save more time and monetary cost than the other three algorithms, ReOpt, Tukwila, and NoReOpt. In this experiment, re-optimization contributes to these savings and it is beneficial on two aspects. First, after a re-optimization, the optimizer implements different types of physical opertors. Different types of physical operators, such as NestedLoopJoin or HashJoin, used to execute these JOINs can result in a large difference in query response time. Second, re-optimizations help decide the degree of parallelism of each operator so that a lot of money is saved as fewer containers are used for executing these operators. However, not all re-optimizations are useful as discussed in Sections 1 and 3, conducting more useful reoptimizations and avoiding unnecessary re-optimizations can further improve performance. We compare the QEP before and after re-optimizations in each algorithm to find out whether each re-optimization is actually necessary or not. In this experiment, nearly 70% of the re-optimizations are necesscary in ReOptML, while only 35% in ReOpt and 28% in Tukwila are necessary. From this, we conclude that using machine learning further helps improve both time and monetary costs of query execution by avoiding unnecessary re-optimizations.

Uniform Data: In addition to the results obtained from executing queries on skew data, Figure 10 (c) and (d) also show the results of executing the same queries on uniform data. These two figures report only the query types that have the large differences in query response time and monetary cost. From Figure 10 (c), on average we see that ReOptML yields 13%, 13% and 21% less query response time than ReOpt, Tukwila, and NoReOpt, respectively. The total savings of query response times resulted from ReOptML, ReOpt, Tukwila and NoReOpt on uniform data are less than those on skew data because the optimizer assumes the data is uniformly distributed by default. Thus, the error of cost estimation on uniform data is less than that on skew data. This shows that query re-optimization in general is more helpful on executing queries on skew data. In term of monetary cost, from Figure 10 (d), on average we see that ReOptML spends the same amount of money as ReOpt, 7% less money than Tukwila, but 10% more money than NoReOpt. From these results, we find that when queries are executed on uniform data, re-optimization saves time, but does not improve monetary cost.

In summary, we conclude that using machine learning to predict when a re-optimization is beneficial does improve query response time no matter queries are executed on uniform or skew data. In terms of monetary cost, this algorithm also saves a significant amount of monetary cost when queries are executed on skew data, but gives no improvement when queries are executed on uniform data.

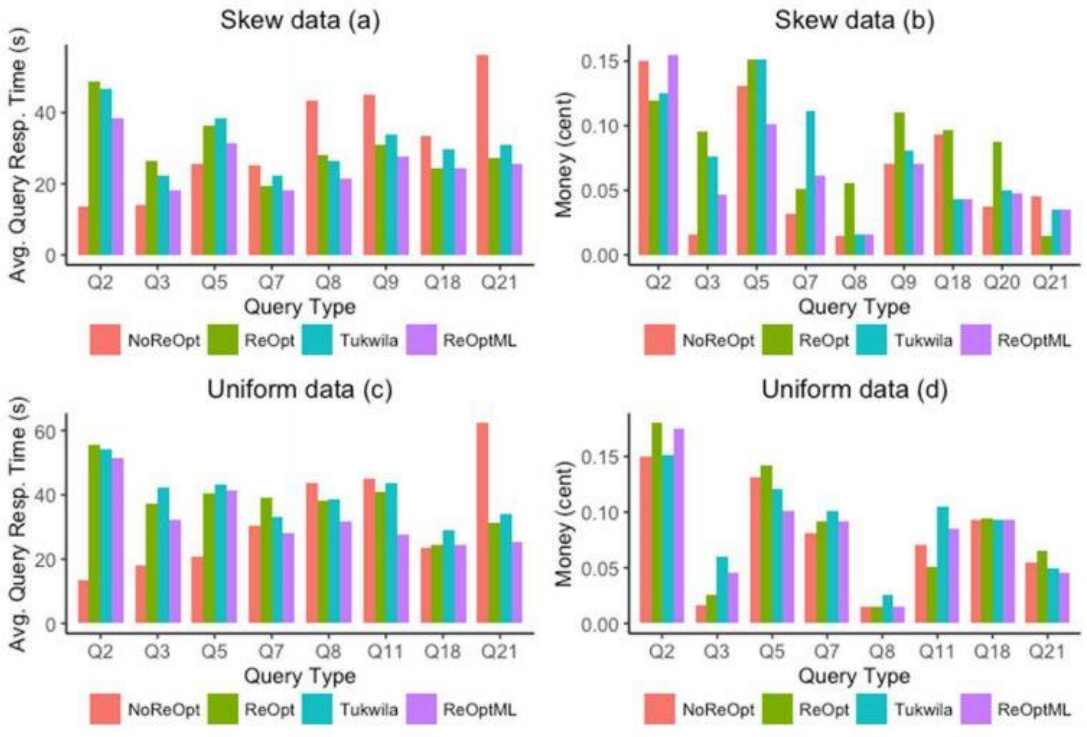

Figure 10. (a)-(d). Average query response time and monetary cost of executing one query from different query types on skew data (a-b) and on uniform data (c-d)

6. CONCLUSION AND FUTURE WORK

This paper presents an algorithm called ReOptML that uses a machine learning-based model to decide whether or not a query should be re-optimized. The experiments conducted show that for skew data, ReOptML improves the query response time (from 13% to 35%) and monetary cost (from 17% to 35%) over the existing algorithms that use either no re-optimization, optimization after each stage in the query execution plan (QEP) is executed, or re-optimization when a check point is reached and the difference between the actual query cost and estimated query cost exceeds some threshold. For uniform data, the proposed algorithm also improves query response time (13% to 21%) over the existing algorithms, but does not improve monetary cost. While our studies have shown that machine learning has positive impacts on deciding whether a re-optimization should be conducted, the machine learning model proposed in this work provides only a binary decision of whether or not a re-optimization should be carried out, and the model relies on the data statistics (features) which may not be available in all DBMSs. For future work, we will investigate techniques that do not rely on data statistics. In addition, we will also extend our approach to predicting, independently of query stages, when a query re-optimization should be carried out, and predicting how many times such query re-optimization should occur.

REFERENCES

[1] N. Bruno, S. Jain and J. Zhou, "Continuous cloud-scale query optimization and processing",VLDB Endow. 6, 11 pp.961–972, 2013.

[2] F. Wolf, N. May, R. Willems and K.-U. Sattler, "On the Calculation of Optimality Ranges forRelational Query Execution Plans", 2018 International Conference on Management of Data (SIGMOD '18), pp.663–675, 2018.

[3] A. Deshpande, Z. Ives and V. Raman, "Adaptive Query Processing", Foundations and Trends® inDatabases: Vol. 1: No. 1, pp.1-140, 2017.
[4] V. Markl, V. Raman, D. Simmen, G. Lohman, H. Pirahesh and M. Cilimdzic, "Robust queryprocessing through progressive optimization", 2004 ACM SIGMOD international conference onManagement of data (SIGMOD '04), pp.659–670, 2004.
[5] Details omitted for double-blind reviewing.
[6] A. A. Bankole and S. A. Ajila, "Predicting cloud resource provisioning using machine learningtechniques", 26th IEEE Canadian Conference on Electrical and Computer Engineering (CCECE),pp.1-4, 2013.
[7] A. Jindal, K. Karanasos, S. Rao and H. Patel, "Thou Shall Not Recompute: SelectingSubexpressions to Materialize at Datacenter Scale", VLDB Endow. 11, 7,2018.
[8] H. Liu, M. Xu, Z. Yu, V. Corvinelli and C. Zuzarte, "Cardinality Estimation Using NeuralNetworks", 25th Annual International Conference on Computer Science and Software Engineering(CASCON '15), pp.53-59, 2015.
[9] R. Marcus and O. Papaemmanouil, "Deep Reinforcement Learning for Join Order Enumeration" in1st International Workshop on Exploiting Artificial Intelligence Techniques for Data Management(aiDM'18), pp.1-4, 2018.
[10] P. Yongjoo, T. Ahmad, C. Michael and M. Barzan, "Database Learning: Toward a Database thatBecomes Smarter Every Time", 2017 ACM International Conference on Management of Data(SIGMOD '17), pp.587-602, 2017.
[11] J. Ortiz, M. Balazinska, J. Gehrke and S. S. Keerthi, "Learning State Representations for QueryOptimization with Deep Reinforcement Learning" the Second Workshop on Data Management forEnd-To-End Machine Learning (DEEM'18), pp.1-4, 2018.
[12] L. Breiman, "Random Forests", Machine Learning, 45 (5), 2001.
[13] J. Schmidhuber, "Deep Learning in Neural Networks: An Overview." Neural Networks. , vol. 61,no. 10, pp.85-117, 2014.
[14] C. Corinna and V. Vladimir N., "Support-vector networks", Machine Learning. 20 (3), 1995.
[15] A. Kipf, T. Kipf, B. Radke, V. Leis, P. Boncz and A. Kemper, "Learned Cardinalities: EstimatingCorrelated Joins with Deep Learning", 9th Biennial Conference on Innovative Data SystemsResearch (CIDR'19), 2019.
[16] Thirumuruganathan, Saravanan, H. Shohedul, K. Nick and D. Gautam, "Approximate QueryProcessing for Data Exploration using Deep Generative Models", 36th International Conference onData Engineering (ICDE), pp.1309-1320, 2020.
[17] W. Wu, J. F. Naughton and H. Singh, "Sampling-Based Query Re-Optimization", 2016International Conference on Management of Data (SIGMOD '16), pp.1721–1736, 2016.
[18] Z. G. Ives, D. Florescu, M. Friedman, A. Levy and D. S. Weld, "An adaptive query executionsystem for data integration", 1999 ACM SIGMOD international conference on Management ofdata (SIGMOD '99), pp.299-310, 1999.
[19] K. Navin and D. David, "Efficient mid-query re-optimization of sub-optimal query executionplans", SIGMOD Rec. 27, 2, pp.106–117, 1998.
[20] W. Lang, R. Nehme and I. Rae, "Database optimization for the cloud: Where costs, partial results,and consumer choice meet", 5th Biennial Conference on Innovative Data Systems Research(CIDR'15), 2015.
[21] W. Wang, M. Zhang, G. Chen, H. V. Jagadish, B. C. Ooi and K.-L. Tan, "Database Meets DeepLearning: Challenges and Opportunities", SIGMOD Rec., 45(2), pp.17–22, 2016.
[22] I. Trummer, S. Moseley, D. Maram, S. Jo and J. Antonakakis, "SkinnerDB: Regret-bounded QueryEvaluation via Reinforcement Learning", VLDB Endow. 11, 12, pp.2074–2077, 2018.
[23] M. Stillger, G. M. Lohman, V. Markl and M. Kandil, "LEO - DB2's LEarning Optimizer", 27thInternational Conference on Very Large Data Bases (VLDB '01), pp.19–28, 2001.
[24] W. Chenggang, A. Jindal, S. Amizadeh, H. Patel, W. Le, S. Qiao and S. Rao, "Towards a learningoptimizer for shared clouds", VLDB Endow. 12, 3, pp.210–222, 2018.
[25] F. Helff, L. Gruenwald and L. d'Orazio, "Weighted Sum Model for Multi-Objective QueryOptimization for Mobile-Cloud Database Environments", EDBT/ICDE Worshops, 2016.
[26] D. Meignan, "A heuristic approach to schedule reoptimization in the context of interactiveoptimization", 2014 Annual Conference on Genetic and Evolutionary Computation (GECCO '14),pp.461-468, 2014.

[27] M. Barata, J. Bernardino and P. Furtado, "An Overview of Decision Support Benchmarks: TPCDS, TPC-H and SSB" Advances in Intelligent Systems and Computing. , vol. 353. , pp.619-628,2015.
[28] E. Alpaydin, Introduction to Machine Learning 3rd Edition, 2014.
[29] PostgreSQL, "https://www.postgresql.org/," [Online].
[30] R. Marcus, P. Negi, H. Mao, C. Zhang, M. Alizadeh, T. Kraska, O. Papaemmanouil and N. Tatbul,"Neo: a learned query optimizer" in VLDB Endow. 12, 11, pp.1705–1718, 2019.
[31] H. Kllapi, E. Sitaridi, M. M. Tsangaris, and Yannis Ioannidis, "Schedule optimization for dataprocessing flows on the cloud", 2011 ACM SIGMOD International Conference on Management ofdata (SIGMOD '11), pp.289–300, 2011 .
[32] C. H. Papadimitriou and M. Yannakakis, "Multiobjective query optimization", the 20th ACMSIGMOD-SIGACT-SIGART symposium on Principles of database systems (PODS '01), pp.52–59, 2001.

AUTHORS

Chenxiao Wang is currently PhD student in Computer Science Department, University of Oklahoma. His research interest is Cloud Query Optimization

Zach Arani is undergraduate student and research assistant in Computer Science Department, University of Oklahoma. His research interest is Mobile-Cloud Environment

Dr. Le Gruenwald is currently the professor of School of Computer Science, University of Oklahoma. She received PhD degree at Southern Methodist University. Her research interest are Cloud Databases, Mobile Databases.

Dr. Laurent d'Orazio is currently the professor of computer science, Univ Rennes, CNRS, IRISA. His research topics include Distributed Systems: Cloud Computing and Big Data.

Dr. Eleazar Leal is currently the assistant professor of college of science and engineering, University of Minnesota, Duluth. He received his Phd degree in University of Oklahoma. His research interests are spatial Databases, StreamDatabases, Multicore and GPU Algorithms for Databases