A Vision of a Decisional Model for Re-optimizing Query Execution Plans Based on Machine Learning Techniques

Chenxiao Wang
University of Oklahoma
Norman, Oklahoma, United States of America
chenxiao@ou.edu

Zachary Arani
University of Oklahoma
Norman, Oklahoma, United States of America
myrrhman@ou.edu

Le Gruenwald
University of Oklahoma
Norman, Oklahoma, United States of America
ggruenwald@ou.edu

Laurent d'Orazio
Rennes 1 University
Lannion, France
laurent.dorazio@univ-rennes1.edu

ABSTRACT

Many of the existing cloud database query optimization algorithms target reducing the monetary cost paid to cloud service providers in addition to query response time. These query optimization algorithms rely on an accurate cost estimation so that the optimal query execution plan (QEP) is selected. The cloud environment is dynamic, meaning the hardware configuration, data usage, and workload allocations are continuously changing. These dynamic changes make an accurate query cost estimation difficult to obtain. Concurrently, the query execution plan must be adjusted automatically to address these changes. In order to optimize the QEP with a more accurate cost estimation, the query needs to be optimized multiple times during execution. On top of this, the most updated estimation should be used for each optimization. However, issues arise when deciding to pause the execution for minimum overhead. In this paper, we present our vision of a method that uses machine learning techniques to predict the best timings for optimization during execution.

KEYWORDS

Query Optimization, Cloud Computing, Cloud databases

1 INTRODUCTION

Many of the existing cloud database query optimization algorithms target reducing the monetary cost paid to cloud service providers in addition to query response time. The time and monetary costs needed to execute a query are estimated based on the data statistics that the query optimizer has available when the query optimization is performed. These statistics are often not accurate, which may result in inaccurate estimates for the time and monetary costs needed to execute the query [3, 12]. Thus, the QEP generated before the query is executed may not be the best one. One approach can be applied to solve this issue. Adaptively optimizing the QEP during the query execution to employ more accurate statistics will yield better QEP selection, and thus will improve query performance [3, 6] . QEP is not executed as whole at one time but is divided and executed part by part. After completion of one part of the QEP, data statistics are updated so that the rest part of the QEP is re-optimized adopting the new data statistics. The new QEP is expected to be changed as it either contains different operators or is scheduled to be executed on different machines. Such changes result in a different way of executing a QEP which brings different response time and monetary cost. However, re-optimizing a QEP costs time overhead which in turn produces extra monetary cost as well. To avoid unnecessary re-optimization and decide whether or not a QEP should be re-optimized is not an easy task. In the work [6], the authors manually set check points between different operators of a QEP. Re-optimizations at these check points are necessary but still not accurate. The work [3] proposed a query optimization method where the query is re-optimized multiple times during its execution based on stages. Stages are formed automatically by the query optimizer and operators that do not rely on the completion of others are grouped together. Every time one stage is finished, the query is re-optimized by force. We implemented this algorithm in our previous work [11]. However, our experimental studies show that after applying many re-optimizations, the QEP remains the same compared to the original one. This is because the stages are not aligned with the best timing to apply the re-optimization. This wastes time as unnecessary optimization increases overhead. In this paper, we provide our vision of a method using machine learning techniques to predict whether a QEP should be optimized or not during the query execution, so that the overall overhead of re-optimization is further reduced as unnecessary re-optimization is avoided more accurately.

The rest of the paper is organized as follows. Section 2 discusses the related works. Section 3 discusses the effects of query re-optimization. Section 4 presents how to predict query re-optimization by machine learning. Section 5 discusses the feature selection and machine learning model selection issues according to the status of our current work. Section 6 provides conclusion and future work.

2 RELATED WORK

There are several works [4–9] that study when a query should be re-optimized. Some of them are interactive, which means human input is required in the re-optimization process. [6] presents a mid-query re-optimization algorithm. In this work, a few pointers are put over different locations of the QEP and whenever the query operators before the pointers finish, the query will be re-optimized. These locations are chosen based on a set of rules built by the authors. The locations that satisfy these rules indicate that re-optimization is worthwhile. For example, the re-optimization will take place before a Join operator. [7] introduces an algorithm for re-optimizing the schedule to run different queries. In this work, the algorithm compares the distance between an initial schedule and the ideal schedule which is defined by multiple human defined rules. Whenever the distance exceeds

some threshold, the schedule is re-optimized. [4] also presents another mid-query re-optimization algorithm where a statistic-collect operator is introduced to be placed at key points and used to collect the updated data statistics during the query execution. These updated statistics are used to re-optimize the QEP itself and the memory allocation to execute the query. The estimated execution time after re-optimization is compared to the estimated execution time before the same QEP is re-optimized. If the difference in execution time exceeds a manually set threshold, the re-optimization is conducted. However, as presented in our previous work [11], taking human input into the decision-making process greatly increases time overhead and introduces a source of unreliable accuracy. In [8], instead of using human input, reinforced learning is used for the optimizer to decide which physical operators the optimizer should select (the optimizer's actions) based on the current data statistics (the optimizer's states). But still, inaccurate data statistics will result in sub-optimal selection of physical operators. [9] presents a query re-optimization algorithm to estimate the cost of the current query by looking at the similar queries answered in the past. This method uses previous known column statistics to predict unknown column statistics by using the joint probability density function. However, in this work, a matrix inversion is required to calculate the cost of the unknown column statistics. Applying such an operation online would cost a lot of time overhead. [5] presents an algorithm to adaptively optimize the QEP on cloud databases. This work assumes that users are willing to accept incomplete query results if the cost is under users' budget. However, the optimization in this work considers either time cost or monetary cost, not both. All these existing works, while using query re-optimization to improve query response time, are not concerned about monetary costs at the same time or are not designed for cloud databases.

To fill the gaps in the existing works, the approach we envision in this paper emphasizes on addressing a number of important issues. First, the approach utilizes query re-optimization not only for query response time but also monetary costs at the same time for cloud databases. Second, to achieve a greater accuracy in terms of the timing for re-optimization to occur and to do so autonomously, the approach uses a machine learning model trained by historical queries to predict when to do re-optimization instead of manually deciding. Third, to reduce the time overhead, the approach consists of the offline and online processes. The offline process takes the majority of the time to build the machine learning model, while the online process is only applying the model to do the prediction, which limits the time overhead. Fourth, the approach always uses the actual data statistics so that the optimizer is always able to select the correct QEP.

3 QUERY RE-OPTIMIZATIONS

In a traditional DBMS, queries are first converted to multiple QEPs. Following this, all the QEPs are then evaluated by the query optimizer to obtain the time costs. In some systems, additional costs like monetary cost, network bandwidth, and hardware utilization are also calculated. Finally, the optimizer chooses the best QEP and sends it to the execution engine. However, unlike traditional DBMS, we apply mid-query re-optimization (Figure 1). This means a query plan can be optimized for multiple times during execution. In a traditional DBMS, the QEPs are optimized by the cost estimation(s) which are evaluated based on data statistics such as the cardinality of a column, number of rows in a table, etc. However, such data statistics are often not accurate

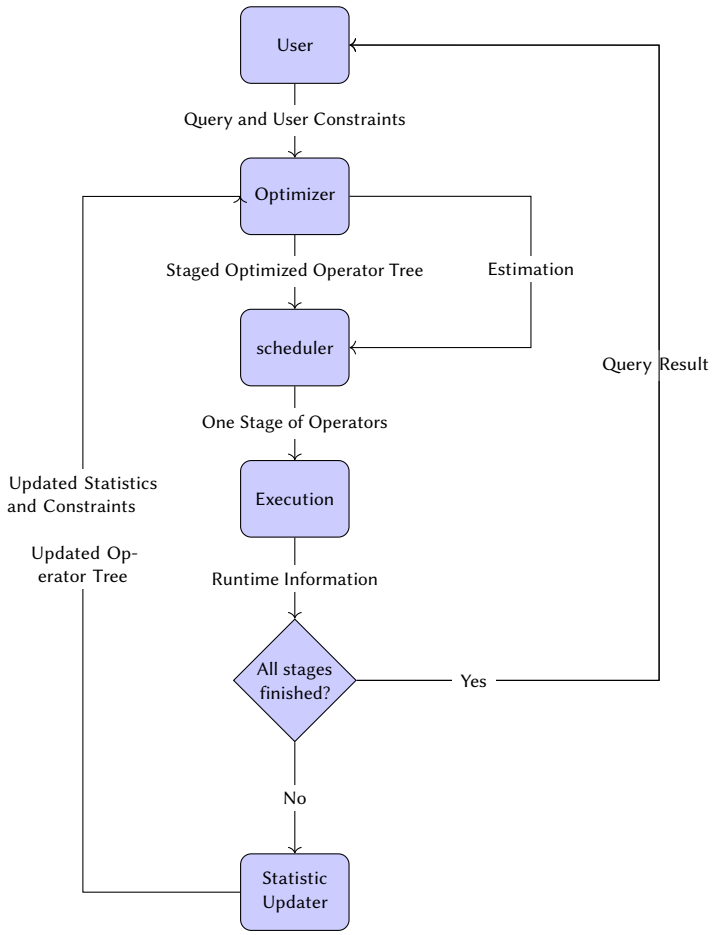

Figure 1: The query re-optimization procedure.

when the estimation is evaluated. Thus, the QEP generated may not be the most efficient. Re-optimizing the QEP during query execution to employ more accurate statistics will yield better QEP selection, and thus will improve query performance. In our previous work [10], we discover that query re-optimization will enable the optimizer to select better physical operators to execute the QEP and select better hardware configurations to execute the QEP (such as the number of containers and the type of containers). These optimizations are beneficial for improving either the overall query execution time or the monetary cost or both. Figure 2 shows the result of executing the query from our previous work [10]. In the experiment query, there is a Join of two subqueries. The data size of each subquery is unknown. We want to see how the physical operator of this Join will change depending on the data size of the subquery. So, we purposely made the data size of the right side of the Join operator small enough to fit in the cache. As a consequence, the Shuffle Join operator will be changed to the Broadcast Join operator only after re-optimization. Broadcast Join is executed around 40% faster than Shuffle Join in our environment. The results show the overall time cost using re-optimization has approximately 20% improvement on average over no re-optimization, while the monetary costs of the two approaches were close, with only a 4% difference. This increase of monetary cost is due to the fact that the more powerful containers being selected are the containers which charge more hourly.

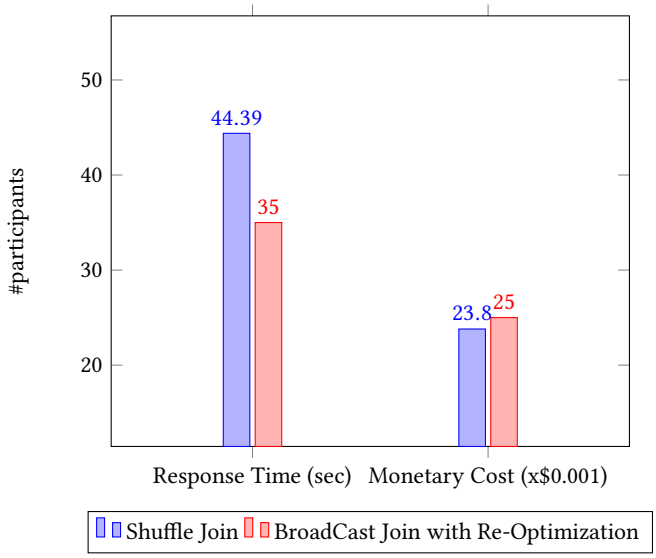

Figure 2: Impact of physical operators on time and monetary cost for the execution of Query 2.

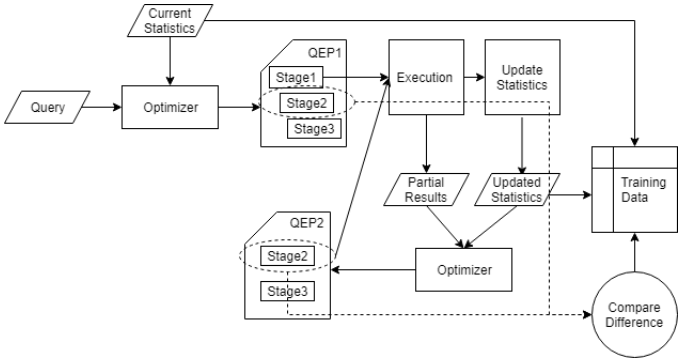

Figure 3: The procedure of collecting training data.

4 MACHINE LEARNING BASED QUERY RE-OPTIMIZATION DECISION PROCESS

4.1 Overview

Though we find that re-optimization improves the query response time, the re-optimization itself increases the overhead. Besides that, not all the re-optimizations are effective for the QEP, that is, some re-optimizations may yield no changes. These re-optimizations are unnecessary and increase overhead. In order to avoid this issue, we envision a method using a machine learning model built on past data (training data) to predict whether or not a future re-optimization would be useful. If the prediction indicates that the re-optimization will have no effect on the QEP, then the query will not be re-optimized.

4.2 Architecture

Figure 3 shows the architecture of how the training data is collected. First, we collect the training data by running random queries on our current system and observing the data statistics. This way the prediction model can be applied to all queries. If re-optimization is only for most costly/most representative queries, then in this first step, the training data should be collected from

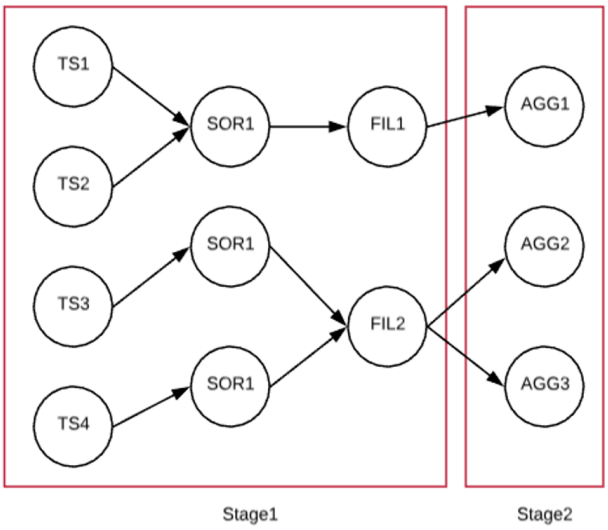

Figure 4: A QEP is divided into different stages after being compiled from the query.

running only random but most costly/representative queries. After a query is submitted, we record the current data statistics in the system. These current statistics are called Statcurr. Then, the query is sent to the optimizer and an optimal QEP is generated by using the optimizer in a traditional DBMS. This QEP includes the stage information and on which nodes these stages will be executed. Figure 4 shows an example of a QEP generated by the query optimizer for the following query:

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

In Figure 4, TS, SOR, FIL and AGG stand for TableScan, Sort, Filter and Aggregate operators, respectively. The subscripts distinguish the same operators that are executed in parallel on different data. After that, Stage 1 is sent to the query execution engine. During the execution, we update the data statistics using the method mentioned in the work [3] and we call these updated statistics Statupdate. Since these statistics are collected from the actual running query, Statupdate is more accurate than Statcurr which is obtained from the estimation. The difference between the Statupdate and Statcurr, called Statdiff, is then used in the machine learning model to predict whether the re-optimization is effective. 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 one feature of the training dataset. This process is applied to all the features. The selected features are shown in Table 1.

If the re-optimization is predicted to be effective, 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. We define that a QEP is changed if one of the following three aspects occurs: 1) changes in the physical operator types; 2) changes in the number of containers; or 3) changes in the types of containers.

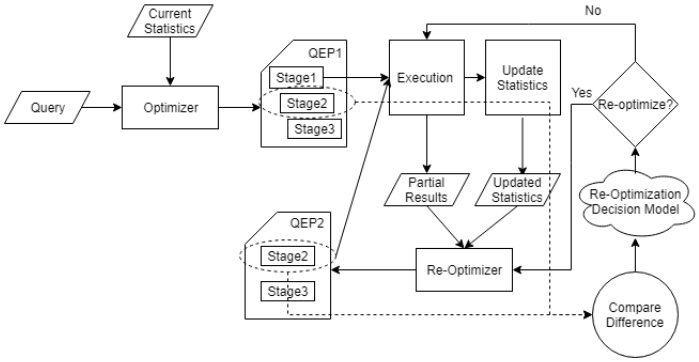

Figure 5: The procedure of applying the prediction model to decide re-optimization.

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 experiment, 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 QEP after a re-optimization, this re-optimization is probably effective, and the re-optimization will be applied if the similar situation is encountered.

A change in the number or types of containers means 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 containers are distributed into four containers before the re-optimization and use three containers after the re-optimization. This change highly influences the monetary cost of query execution. Thus, such re-optimization becomes useful if such changes are detected. Similarly, a change in the types of containers means the QEP after the re-optimization is executed on different types of containers, either more powerful ones or weaker ones. Detecting such changes may influence the monetary cost as well.

4.3 Feature Selection

The three changes discussed in Section 4.2 usually occur whenever the estimated data size has also changed. This is because the 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 is effective, we use data features that are highly relevant to the changes in data size estimation.

Assume in the current DBMS, there exist the C={C_1,C_2,...,C_n} columns in all the tables. Selectivity differences (*DIFF_SELECTIVITY*), distinct values (*DIFF_NVD*), and histograms (*DIFF_HISTOGRAM*) of each column are used as the data features in the training data used for prediction as shown in Table 1, and the binary value YES/NO is used as the predicted class, where YES means 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 [6, 12, 13]. Thus, the differences in these three features result in changes to data size estimation of intermediate results. Hence, they become relevant in deciding the effectiveness of re-optimization

In our preliminary work to test our vision, we have collected the training data from running 5000 random queries on a DBMS for 24 hours and collected the data for the above features. Then a prediction model was trained using these data and applied to a new query to predict whether or not the re-optimization should be conducted after each stage of this query is executed. Figure 5 illustrates how this model is applied during the query execution. The query is first converted to a QEP and Stage 1 is submitted for execution. The prediction model is used to check whether or not the re-optimization should be conducted. Only a 'YES' prediction will trigger re-optimization. By doing this, the unnecessary re-optimization discussed previously is eliminated.

Table 1: List of Features and their types

DIFF_SELECTIVITY(C1)
DIFF_SELECTIVITY(C2)
DIFF_SELECTIVITY(Cn)
DIFF_NDV(C1)
DIFF_NDV(C2)
DIFF_NDV(Cn)
DIFF_HISTOGRAM(C1)
DIFF_HISTOGRAM(C2)
DIFF_HISTOGRAM(Cn)

5 DISCUSSION

5.1 Additional Feature Selection

Some additional features can also be added to the training model, although the utility of new features may not be immediately clear. For instance, we considered adding the total available CPU usage to the feature list. However, the metric of the total available CPU appears to not be relevant in predicting re-optimization. For instance, take the statement "If the total available CPU is low, then the re-optimization will not be conducted." The answer to this statement is false as when we collect the training data, re-optimizations are conducted anyway no matter how low the available CPU usage is. Re-optimization is only influenced by the estimation of data size. The CPU usage only determines which containers should be assigned. Thus, this attribute is not relevant.

5.2 Machine Learning Model Selection

We postulate that several machine learning techniques can be applied to predict whether or not re-optimization should be conducted. Pre-processing data to reduce the number of features before processing with a machine learning model is necessary. As in our model, the number of features is linearly related to the accumulative number of columns in all tables. If there is a high number of columns, there will be too many predictors. Principle Component Analysis (PCA) can be a useful option to find out what features are important.

Unsupervised learning such as clustering has been used in query optimization [14]. In our case, clustering can be used to determine if re-optimization is useful by identifying a specific pattern of data statistics for the DBMS. There exist serval popular clustering algorithms such as K-means and DBSCAN [1], which are used on a situational basis. In our study, we think K-means is suitable as the value of K is fixed. In order to predict whether or not re-optimization should be conducted, we can set the K value to two. Two clusters are formed, one is for the "YES" cases and another one is for the "NO" cases. Similar data statistics collected after re-optimizations are measured by the normalized Euclidean distance function and are grouped together to form these two clusters. When new data statistics arrive, they are measured to

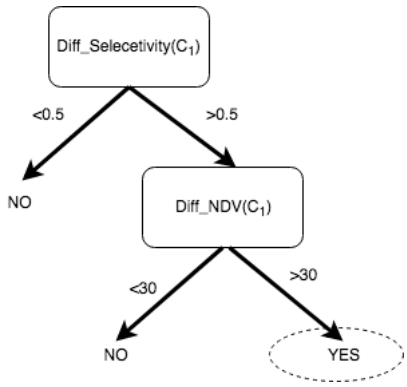

Figure 6: An Example of a decision tree showing that re-optimization should take place

determine to which cluster they belong. If it belongs to the "YES" cluster, then a re-optimization is necessary. Besides unsupervised learning such as the K-means clustering, supervised learning can also be used to predict whether or not re-optimization would be necessary. Supervised learning is not without issue, however. For instance, it is usually hard to get labeled data for training a model. In our case, labels can be easily obtained by observing the effect of re-optimization on past QEPs. As there are a fair amount of supervised learning algorithms, several possible options exist for prediction. For instance, a binary classifier decision tree can be examined to classify whether or not re-optimization should be conducted. Each feature will be represented by a node split into either "YES" or "NO". Figure 6 shows an example of this partial decision tree. When the final classification is "YES", then a re-optimization is necessary. Also, the performance of several other supervised learning algorithms like Neural Networks, Support Vector Machines, and Linear Regression were studied in [2] for cloud provisioning, but not for query re-optimization. We are currently investigating which machine learning models are most suitable for our study. An appropriate machine learning model should be highly accurate in prediction (i.e. having low error rates when applied to the training data and test data). The training data must be selected carefully to avoid the cases of model overfitting, i.e., the cases where the model provides a low training error rate, but a high testing error rate. In addition, an appropriate model should incur a low overhead while being applied online for re-optimization prediction and should perform effectively in reducing the overall query response time and monetary cost.

6 CONCLUSION AND FUTURE WORK

In this paper, we provide our vision of a model that utilizes machine learning techniques to study previously observed statistics data from a running system in order to build a prediction model. This model is used to predict whether or not a query should be re-optimized in order to avoid unnecessary re-optimizations. By doing this, a query's execution time and/or monetary cost can be further reduced. In future work, we plan to fully implement the approach we envisioned and use it to predict additional behaviors of a DBMS. For example, we would like to study methods for increasing or decreasing the number of executing containers based on current data statistics. We believe that predicting additional useful behaviors will make the query re-optimization process more efficient. In addition, we will also extend our approach to predict, independently of query stages, when a query re-optimization should be done, and to predict how many times such query re-optimizations should occur.

ACKNOWLEDGMENTS

This work is partially supported by the National Science Foundation Award No. 1349285.

REFERENCES

[1] G. Ahalya and H. M. Pandey. 2015. Data clustering approaches survey and analysis. In *2015 International Conference on Futuristic Trends on Computational Analysis and Knowledge Management (ABLAZE)*. 532–537. https://doi.org/10.1109/ABLAZE.2015.7154919

[2] A. A. Bankole and S. A. Ajila. 2013. Predicting cloud resource provisioning using machine learning techniques. In *2013 26th IEEE Canadian Conference on Electrical and Computer Engineering (CCECE)*. 1–4. https://doi.org/10.1109/CCECE.2013.6567848

[3] Nicolas Bruno, Sapna Jain, and Jingren Zhou. 2013. Continuous Cloud-scale Query Optimization and Processing. *Proc. VLDB Endow.* 6, 11 (Aug. 2013), 961–972. https://doi.org/10.14778/2536222.2536223

[4] Navin Kabra and David J. DeWitt. 1998. Efficient Mid-query Re-optimization of Sub-optimal Query Execution Plans. *SIGMOD Rec.* 27, 2 (June 1998), 106–117. https://doi.org/10.1145/276305.276315

[5] Willis Lang, Rimma V. Nehme, and Ian Rae. 2015. Database Optimization in the Cloud: Where Costs, Partial Results, and Consumer Choice Meet. In *CIDR*.

[6] Volker Markl, Vijayshankar Raman, David Simmen, Guy Lohman, Hamid Pirahesh, and Miso Cilimdzic. 2004. Robust Query Processing Through Progressive Optimization. In *Proceedings of the 2004 ACM SIGMOD International Conference on Management of Data (SIGMOD '04)*. ACM, New York, NY, USA, 659–670. https://doi.org/10.1145/1007568.1007642

[7] David Meignan. 2014. A Heuristic Approach to Schedule Reoptimization in the Context of Interactive Optimization. In *Proceedings of the 2014 Annual Conference on Genetic and Evolutionary Computation (GECCO '14)*. ACM, New York, NY, USA, 461–468. https://doi.org/10.1145/2576768.2598213

[8] Jennifer Ortiz, Magdalena Balazinska, Johannes Gehrke, and S. Sathiya Keerthi. 2018. Learning State Representations for Query Optimization with Deep Reinforcement Learning. In *Proceedings of the Second Workshop on Data Management for End-To-End Machine Learning (DEEM'18)*. ACM, New York, NY, USA, Article 4, 4 pages. https://doi.org/10.1145/3209889.3209890

[9] Yongjoo Park, Ahmad Shahab Tajik, Michael Cafarella, and Barzan Mozafari. 2017. Database Learning: Toward a Database That Becomes Smarter Every Time. In *Proceedings of the 2017 ACM International Conference on Management of Data (SIGMOD '17)*. ACM, New York, NY, USA, 587–602. https://doi.org/10.1145/3035918.3064013

[10] Chenxiao Wang, Zach Arani, Le Gruenwald, and Laurent d'Orazio. 2018. Adaptive Time, Monetary Cost Aware Query Optimization on Cloud Database Systems. 3374–3382. https://doi.org/10.1109/BigData.2018.8622401

[11] Chenxiao Wang, Jason Arenson, Florian Helff, Le Gruenwald, and Laurent d'Orazio. 2017. Improving user interaction in mobile-cloud database query processing. In *Workshop on Scalable Cloud Data Management*. Boston, United States. https://hal.archives-ouvertes.fr/hal-01640072

[12] Florian Wolf, Norman May, Paul R. Willems, and Kai-Uwe Sattler. 2018. On the Calculation of Optimality Ranges for Relational Query Execution Plans. In *Proceedings of the 2018 International Conference on Management of Data (SIGMOD '18)*. ACM, New York, NY, USA, 663–675. https://doi.org/10.1145/3183713.3183742

[13] Wentao Wu, Jeffrey F. Naughton, and Harneet Singh. 2016. Sampling-Based Query Re-Optimization. In *Proceedings of the 2016 International Conference on Management of Data (SIGMOD '16)*. ACM, New York, NY, USA, 1721–1736. https://doi.org/10.1145/2882903.2882914

[14] J. Zahir, A. El Qadi, and S. Mouline. 2014. Access plan recommendation: A clustering based approach using queries similarity. In *2014 Second World Conference on Complex Systems (WCCS)*. 55–60. https://doi.org/10.1109/ICoCS.2014.7060936