SLA-Aware Cloud Query Processing with Reinforcement Learning-Based Multi-objective Re-optimization

Chenxiao Wang[1(✉)], Le Gruenwald[1], and Laurent d'Orazio[2]

[1] School of Computer Science, University of Oklahoma, Norman, OK, USA
{chenxiao,ggruenwald}@ou.edu
[2] CNRS IRISA, Rennes 1 University, Lannion, France
laurent.dorazio@univ-rennes1.fr

Abstract. Query processing on cloud database systems is a challenging problem due to the dynamic cloud environment. In cloud database systems, besides query execution time, users also consider the monetary cost to be paid to the cloud provider for executing queries. Moreover, a Service Level Agreement (SLA) is signed between users and cloud providers before any service is provided. Thus, from the profit-oriented perspective for the cloud providers, query re-optimization is multi-objective optimization that minimizes not only query execution time and monetary cost but also SLA violations. In this paper, we introduce ReOptRL and SLAReOptRL, two novel query re-optimization algorithms based on deep reinforcement learning. Experiments show that both algorithms improve query execution time and query execution monetary cost by 50% over existing algorithms, and SLAReOptRL has the lowest SLA violation rate among all the algorithms.

Keywords: Query optimization · Cloud databases · Reinforcement learning · Query re-optimization

1 Introduction

In a traditional database management system (DBMS), finding the query execution plan (QEP) with the best query execution time among those QEPs generated by a query optimizer is the key to the performance of a query. However, in a cloud database system, minimizing query response time is not the only goal of query optimization. As hardware usage is charged on-demand and scalability is available to users, query execution monetary cost also needs to be considered as one of the objectives for optimizing QEPs. Meanwhile, the cloud providers need to minimize SLA violation rate in addition to fulfilling the users' requirements of query execution time and monetary cost for query execution. Traditionally, the query optimizer evaluates the time and monetary costs of different QEPs to derive the best QEP for a query before execution. These time and monetary costs 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

R. Wrembel et al. (Eds.): DaWaK 2022, LNCS 13428, pp. 249–255, 2022.
https://doi.org/10.1007/978-3-031-12670-3_22

needed to execute the query. Thus, the QEP generated before query execution may not be the best one.

To deal with this issue, there exist methods proposed to re-optimize queries during their execution [1–3]. Ortiz and et al. [1] apply deep reinforcement learning (RL) to learn a representation of queries, which can then be used in downstream query optimization tasks. Marcus and et al. present work of a deep RL-based join optimizer, ReJOIN [2], which orders a preliminary view of the potential for deep RL in this context. In these techniques, QEPs are re-optimized multiple times by a deep RL model. Kipf [3] uses Deep Neural Network (DNN) to learn cardinality estimates. Wu and et al. [6] have proposed Sample, a query re-optimization algorithm that updates data statistics estimated from a sample of tuples collected during runtime. However, none of them addresses monetary costs and SLA requirements for cloud databases at the same time. In this paper, we present two algorithms, ReOptRL and SLAReOptRL, that use reinforcement learning to perform multi-objective query re-optimization for query processing in an end-to-end cloud database system. The algorithms employ a reward function designed specifically for query re-optimization considering query execution time, money cost and SLA requirements.

2 The Reinforcement Learning-Based Multi-objective Query Re-optimization Algorithm (ReOptRL)

We choose RL instead of supervised learning methods because RL does not require training data, which is a labeled dataset of past actions, to be available in advance to train the learning model before the model can be used to predict future actions. There are various kinds of RL algorithms that have been proposed. Q-Learning is one of the popular value-based RL algorithms and using the Bellman equation [4].

$$Q(S_t, a_t) \leftarrow Q(S_t, a_t) + \alpha(R_t + \Upsilon Q(S_{t+1}, a_{t+1}) - Q(S_t, a_t)) \quad (1)$$

In Q-Learning, a table (called Q-table) is used to store all the potential state-action pairs (S_n, a_n) and an evaluated Q-value associated with this pair. In Eq. (1), Q (S_t, a_t) is an evaluated value (called Q-value) for executing Action a_t at State S_t. This value is used to select the best Action to perform under the current state. In our scenario, there are many available containers on which a single query operator can be executed. Thus, many state-action pairs are in the Q-table potentially. Iterating a large Q-table incurs extra time overhead which delays the query execution. To solve this issue, we apply Deep Q Network (DQN) [4] as our reinforcement learning for query re-optimization. DQN works similarly to Q-Learning. The major difference is that, given a state, instead of using the Q-table, it uses a neural network to estimate the Q-values for all the potential actions. The input of the neural network is the current state. For the current QEP to represent the current state, we use a one-hot vector adapted from the recent work [2] to represent a QEP. The ReOptRL algorithm is given in Fig. 1. First, a query is submitted to a query optimizer which generates the QEP (logical plan) for the query (Line 4). Then the QEP is converted into a one-hot vector representation (Line 7). This vector is sent to the RL model, which is a neural network. The RL model will evaluate the Q-values

for all the potential actions to execute the next available query operator (Line 8). Each of these actions consists of two parts, a physical operator and a container (machine) to execute the physical operator. Then the action with the best Q-value will be selected and performed by the DBMS (Line 9). After that, the executed query operator is discarded from the QEP (Line 10). The reward is updated with the time and monetary cost needed to execute the operator and then the expected Q-value is updated by the Bellman Eq. (2) with the updated reward (Lines 11–13). The weights of the neural network are updated accordingly by the back-propagation method (Line 14). This process repeats for each operator in the QEP and terminates when all the operators in the QEP are executed. The query results are then sent to the user (Line 17).

Algorithm: Reinforcement Learning Based Multi-Objective Query Re-Optimization (ReOptRL)

INPUT: SQL query, Weight Profile wp, Reward Function R (), Learning rate α, Discount rate ϒ
OUTPUT: The query result set of the input query

1. t=0
2. Result = Ø
3. Q_t= 0
4. QEP = QueryOptimizer(query)
5. **while** QEP≠ Ø
6. Op=next available operator in QEP
7. State S_t= convert QEP to a state vector
8. $Action_t$=RunLearningModel (S_t, wp)
9. Result=Result ∪ execute (Op, $Action_t$)
10. QEP=QEP-Op
11. Update R_N=R (wp, $Action_t$.time, $Action_t$.money))
12. Obtain Q-value of next state Q_{t+1} from the neural network
13. Update Q-value of current state Q_t = Bellman (Q_t, Q_{t+1}, R_t, α, ϒ)
14. Update Weights in the neural network
15. t=t+1
16. **end while**
17. **return** Result

Fig. 1. The ReOptRL algorithm

In ReOptRL, after an action is performed, the reward function is used to evaluate the action. This gives feedback on how the selected action performs to the learning model. The performed action with a high reward will be more likely to be selected again under the same state. The reward function plays a key role in the entire algorithm. According to the Bellman equation, if the reward of performing a previous action A_{t-1} is high on state S_{t-1}, the Q-value will also be high. This means that, given the same state, the action with the good previous performance will have a higher chance to be selected. In our algorithm, we would like the actions with low query execution time and monetary cost to be the ones that will be more likely to be chosen. To reflect this feature, we define the reward function as follows:

$$Reward\ R = \frac{1}{1 + \left(W_t * T_{op}^q\right) + \left(W_m * M_{op}^q\right)} \tag{2}$$

where W_t and W_m are the time and monetary weights provided by the user, and T^q_{op} and M^q_{op} are the time and monetary costs for executing the current operator op in query q.

According to this reward function, the query is executed based on the user's preference.

which is either the user wanting to spend more money for a better query execution time or vice versa. We call these two preferences Weights. These two weights defined by the user are called Weight Profile (wp), which is a two-dimensional vector, and each dimension is a number between 0.0 to 1.0. Notice that the user only needs to specify one dimension of the weight profile, the other dimension is computed as 1-Weight automatically. The detail can be found in our previous work [5].

3 The SLA-Aware Reinforcement Learning-Based Multi-objective Query Re-optimization Algorithm (SLAReOptRL)

An SLA is a contract between cloud service providers and consumers, mandating specific numerical target values which the service needs to achieve. Considering an SLA in query processing is important for cloud databases. If an SLA violation happens, the cloud service providers need to pay a penalty to their users in a form such as money or CPU credits. From a profit-oriented perspective, cloud service providers would want to keep the number of SLA violations as low as possible. Different cloud service providers implement different SLAs with their users. Using time and monetary costs to execute a query as the SLA requirements has been studied in [1]. We find them practical and more specific to users and thus adopt the same SLA requirements in our work.

In particular, the reward function shown in Eq. (4) is extended from Eq. (2) to make it possible to select the best action according to the SLA requirements:

$$Reward\ R = \frac{1}{1 + \left(W_t * \left(T^q_{op} + P_t\right)\right) + \left(W_m * \left(M^q_{op} + P_m\right)\right)} \tag{3}$$

where T^q_{op} and M^q_{op} are the time and monetary costs for executing the current operator *op* in query *q*

$$P_t = \alpha_{op} * delay\_time, \quad P_m = \alpha_{op} * exceeded\_money$$

where α_{op} is the operator impact rate of the operator type *op*

$$delay\_time = \begin{cases} 0 \\ T^q_{op} - SLA.T^q_{op} \ \ if\ T^q_{op} > SLA.T^q_{op} \end{cases} \tag{4}$$

$$exceeded\_money = \begin{cases} 0 \\ M^q_{op} - SLA.M^q_{op} \ \ if\ M^q_{op} > SLA.M^q_{op} \end{cases} \tag{5}$$

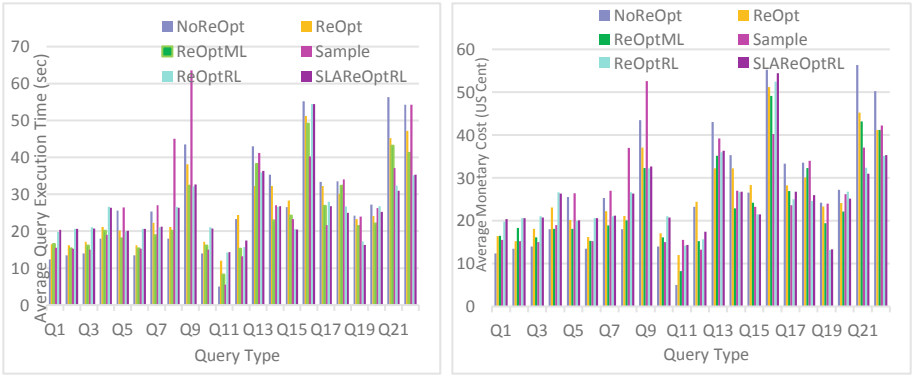
In this reward function (Eq. (4)), P_t and P_m reflect the extra costs for executing a query operator if the SLA is violated. If the SLA is not violated for executing every operator, then this equation is the same as the reward function used in ReOptRL (Eq. (2)). In Eqs. (4) and (5), *delay_time* is the amount of difference between the actual time to

execute a query operator and the maximum time allowed to execute this query operator as specified in the SLA. The same idea applies to *exceeded_money* for monetary costs. We use these two values to quantify the SLA violation on query execution time and monetary cost. In Eq. (4), these two values are used to compute P_t and P_m. It shows that the larger the number of SLA violations, the smaller the reward becomes. We build the reward function this way so that the reward is related to SLA violations. Also, we use the query operator impact rate α_{op} to scale up the impact of SLA violations on different types of operators.

4 Performance Evaluation

There are two sets of machines used in our experiments. 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 16 GB DRAM. The second set consists of 10 dedicated Virtual Private Servers (VPSs) that are used for the deployment of the query execution engine. The query optimizer and the query engine used in the experiments are modified from the open-source database management system, PostgreSQL 8.4. The data are distributed among these VPSs. The queries and database tables are generated using the TPC-H database benchmark. The database tables are populated with 1,000 GB data using the default data generator. We run 50,000 queries and these queries are generated by the query templates randomly selected from the 22 query templates from the benchmark.

We compare the performance results obtained when the following query re-optimization algorithms are incorporated into query processing: 1) our two proposed algorithms, **ReOptRL** and **SLAReOptRL**; 2) the algorithm where a query re-optimization is conducted automatically after the execution of each operator in the query is completed (denoted as **ReOpt**), which we developed based on the work presented in [5]; 3) the algorithm where a query re-optimization is conducted by a supervised machine



(a) (b)

Fig. 2. Time (a) and monetary cost (b) performance for executing queries using different algorithms

learning model decision (denoted as **ReOptML**). 4) the algorithm proposed in [6] where query optimization uses sampling-based query estimation (denoted as **Sample**), and 5) the algorithm that uses no re-optimization (denoted as **NoReOpt**).

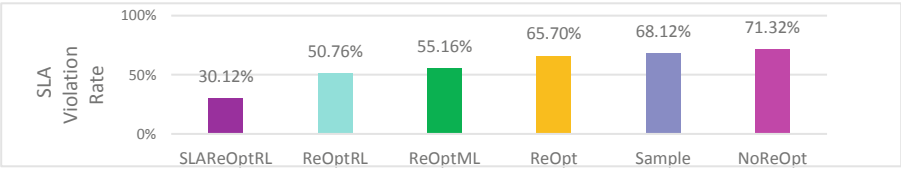


Fig. 3. Average SLA violation rates when executing queries using different algorithms

From Fig. 2 (a) and (b), we can see that, for both the query execution time and monetary costs, on average SLAReOptRL performs the best and ReOptRL performs the second best among all the algorithms. Specifically, comparing with the baseline NoReOpt where no re-optimization is conducted, the query execution time improvement using SLAReOptRL is 45%, ReOptRL 39%, ReOptML 27%, ReOpt 13%, and Sample 10%, while the monetary cost improvement using SLAReOptRL is 62%, ReOptRL 52%, ReOptML 27%, ReOpt 17%, and Sample 5%. Especially, the monetary cost has a significant improvement (SLAReOptRL and ReOptRl are 62% and 52% better than NoReOpt, repsectively). Moreover, from Fig. 3, we can also find that by using SLAReOptRL, the SLA violation rate is the lowest one among the SLA violation rates caused by all the algorithms. This shows the positive effect of considering SLA requirements in re-optimization.

5 Conclusion

This paper presents two query re-optimization algorithms called ReOptRL and SLAReOptRL. Both use a reinforcement learning-based model to decide the physical query operator and machines to execute an operator from a query execution plan (QEP) for a query in a cloud database system. The experiments conducted using the TPC-H database benchmark show that both SLAReOptRL and ReOptRL improve query response time (from 12% to 45%) and monetary cost (from 17% to 62%) over the existing algorithms In addition, we also find that when there are SLA requirements, SLAReOptRL performs 20% better than ReOptRL on SLA violation rate.

References

1. Ortiz, J., Almeida, V.T., Balazinska, M.: Changing the face of database cloud services with personalized service level agreements. In: CIDR 2015 (2015)
2. Marcus, R., Papaemmanouil, O.: Deep reinforcement learning for join order enumeration. In: aiDM 2018, pp. 1–4 (2018)
3. Kandi, M.M., Yin, S., Hameurlain, A.: An integer linear-programming based resource allocation method for SQL-like queries in the cloud. In: SAC 2018, pp. 161–166 (2018)

4. Wiering, M., Otterlo, M.V.: Reinforcement Learning: State-of-the-Art. Springer Publish-ing Company, Incorporated, Berlin, Heidelberg (2014). https://doi.org/10.1007/978-3-642-27645-3
5. Wang, C., Arrani, Z., Gruenwald, L., Laurent, D.: Adaptive time- monetary cost aware query optimization on cloud DataBase. In: Big Data 2018, pp. 3374–3382 (2018)
6. Wu, W., Naughton, J.F., Singh, H.: Sampling-based query re-optimization. In: SIGMOD 2016, pp. 1721–1736 (2016)