Robust Query Execution Time Prediction for Concurrent Workloads on Massive Parallel Processing Databases

Zhihao Zheng[1], Yuanzhe Bei[2], Hongyan Sun[3], and Pengyu Hong[1](✉)

[1] Brandeis University, Waltham, MA 02453, USA
{zhihaozh,hongpeng}@brandeis.edu
[2] MicroFocus Vertica, Cambridge, MA 02140, USA
yuanzhe.bei@microfocus.com
[3] YinTech Innovation Labs, Waltham, MA 02451, USA
hongyan.sun@yintechlabs.com

Abstract. Reliable query execution time prediction is a desirable feature for modern databases because it can greatly help ease the database administration work and is the foundation of various database management/automation tools. Most exiting studies on modeling query execution time assume that each individual query is executed as serialized steps. However, with the increasing data volume and the demand for low query latency, large-scale databases have been adopting the massive parallel processing (MPP) architecture. In this paper, we present a novel machine learning based approach for building a robust model to estimate query execution time by considering both query-based statistics and real-time system attributes. The experiment results demonstrate our approach is able to reliably predict query execution time in both idle and noisy environments at random levels of concurrency. In addition, we found that both query and system factors are crucial in making stable predictions.

Keywords: Query execution · Machine learning · Concurrent

1 Introduction

Commercial databases hold companies' most critical information and need to be maintained at high-availability with stable-latency at all times. They need to be installed and tuned very carefully (e.g., fault tolerance, knob setting, resource pool setting, etc.). Nevertheless, no matter how comprehensive a database has been tuned, it is still challenging to maintain stable-latency [2]. In real world scenarios, databases receive various queries with a wide range of complexities at any given time. Some of those queries are sub-optimal and even do not make much sense, and may cause a database to execute with unexpected long latency and fail to guarantee service quality [4]. In those cases, database administrators

F. Wotawa et al. (Eds.): IEA/AIE 2019, LNAI 11606, pp. 63–70, 2019.
https://doi.org/10.1007/978-3-030-22999-3_6

usually do not have much control and have no clue how to adjust databases in time to deal with the problem.

The situation becomes worse when a database is loaded with concurrent workloads [11], in which we can frequently observe significant degradation of query performance resulting in serious consequences. For example, a sub-second dashboard query may be slowed down to dozens of seconds or even minutes resulting in poor user experience [16]; a system with periodical ETL process can break if the report generation query failed to finish within the interval of two consecutive ETL jobs [9]; with Software as a Service (SaaS) providers the service-level agreement (SLA) can break and result in severe revenue loss [13]; etc. The degradation can be due to resource contention, as concurrent queries may compete on the same resources - disk I/O, network, memory, threads, etc [17]; the degradation can also be caused by less available resources that lead to lower execution parallelism or spills [6]. The following is a classic disastrous case. When a poorly written query (usually generated by dashboard/business intelligence (BI) tool) is sent to the database unintentionally, the query can consume most resources in the system, especially modern analytical databases with extensive distributed/parallelized query execution engines [18]. The whole database performance can be significantly impacted. Even worse. The query may take very long time to run or "never" finish. A typical solution is to allocate dedicated/cascaded resource pools, where short queries and long-run queries have different dedicated pools [10], while short queries can be evicted to the long-run pool if they does not finish within a predefined time threshold. Nevertheless, such solution can only work well if the workload is to some extent known or predictable. In addition, significant resources can be wasted on those queries that need to be evicted and cascaded to a different pool.

The above problem can be much better solved if query execution time can be somewhat predicted. Many researches have been done in estimating area [1,5,15,20,21]. Existing works on estimating the running time of concurrent queries either limit the concurrent query workload to a given set of queries or assume that the execution of each query is serialized. However, in modern analytical database platforms built on massively parallel processing (MPP) architecture.

This study takes a first step to explore the possibility to estimate dynamic workload on Vertica analytic platform [12], a column-oriented relational database system built on the MPP architecture, and is commercialized from the C-Store project [19]. Our goal is to achieve reliable estimations of the running time of any arbitrary query at various levels of concurrency. We adopted a data driven approach and applied machine learning to automatically construct a query execution time estimator.

2 Method

In this section, we explain our machine learning approach for building a model to infer query execution time under mixing workloads with high concurrency. We used Random Forest [7,8,14] to build an ensemble regressor whose inputs include the features describing the states of the operating system and the features extracted from the query plan generated by the Vertica Database for each query.

Constructing appropriate features is essential to building a good prediction model. The Vertica database uses a data flow based cost model, due to its simplicity and robustness. A query is decomposed into a set of operators, which are arranged into a operation tree. Each operator is responsible for running a certain algorithm to perform a sub-task. The Vertica database can estimate the amount resources needed by an operator to process a data flow, The resources are classified into four categories: CPU, Memory, Disk, Network. The Vertica database chooses the best query plan based on the previous mentioned cost model. The execution engine of Vertica is multi-threaded and streamed into a pipeline based on the chosen query plan. More than one operator can run simultaneously at any given time. By using the information from the cost model, we are able to learn the relationship between the required resources and its corresponding execution time. However, this approach only works in single-thread settings, since such features only describes the query itself. When multiple queries are run concurrently, the execution of each query will be significantly affected by others. Thus, we introduced the following system level features to characterize the system wise workload:

1. **CPU circles:** the number of CPU circles requested by an operator.
2. **Memory Data:** the memory size required by an operate.
3. **Network Data:** the amount of data need to be transferred via network by an operator.
4. **Disk Data:** the amount of data to be split out to disk by an operator.
5. **Optimizer Cost:** the cost estimated by the Vertica Optimizer.
6. **Input Rows:** the row count of the related tables to be scanned.
7. **Estimated Output Rows:** the estimated row count of output rows, which is calculated based on Cardinality.
8. **CPU utilization:** the percentage of CPU utilization. If there are multiple CPUs, we take the average.
9. **Query Concurrency:** the number of queries running concurrently.

3 Experiments

The experiments were carried out using the Vertica database. The Vertica database ran on a single cluster of three independent nodes. We used all 22 TPC-H query templates [3] to randomly generate 22000 queries (1000 for each query template) with 10 GB standard TPC-H data. The data was collected by running those 22000 queries at each of the 20 concurrency levels. For a given concurrency level of K $(1 \leq K \leq 20)$, we concurrently run K sessions and each session continuously executes queries randomly sampled from the 22000 queries generated above. We conducted the experiments in two environments. One of them is a "stand-alone" environment where only our experiments are allowed to run, and the other is a "noisy" environment where there are other unknown applications running simultaneously with our experiments. The running time variations are high for queries generated across all query templates and are also high within queries generated from the same query template. Hence, it is challenging to accurately and robustly estimate query execution time. For all the experiments reported below, we split the data into two parts: 70 % of queries as the training data and the rest as the test data. Below we first discuss the results of using individual features in making predictions (Sect. 3.1), which give us some intuitions about their contributions, and then report the results of using all features (Sect. 3.2).

3.1 Query Execution Time Prediction Using Individual Features

This experiment was carried out in a "stand-alone" environment. Table 1 lists the mean relative errors of using individual features in query execution time prediction. The relative error of a prediction is calculated as $|T_{predict} - T_{truth}|/T_{truth}$ where $T_{predict}$ is the predicted time and T_{truth} is the ground truth. The log relative error is calculated as $log(T_{predict}/T_{truth})$. A zero log relative error means $T_{predict} = T_{truth}$. In Table 1, we divide the test data into five tiers based on their real execution time (e.g., the first tier is the fastest 20% queries, and the last tier is the slowest 20% queries). More specifically, the execution time ranges of the queries are (0, 3.4 s) in Tier 1, [3.4 s, 7.1 s] in Tier 2, [7.1 s, 12.4 s] in Tier 3, [12.4 s, 23.2 s] in Tier 4, and (23.2 s, $+\infty$) in Tier 5. Table 1 show that individual features perform poorly under mixing workloads. The overall relative error across tiers ranges from 0.86 to 2.84, which means the predicted execution time of a 10-minute query can range from 1 min to 38 min. Since we used the Mean Square Error as the cost function to optimize the parameters of the query execution prediction model, the trained model focuses more on slow queries, which contribute more significantly to the overall error than short queries. This leads to extremely unstable predictions on short-run queries. We also observe that the model actually performs best for middle-run-time queries and its performance drops more for queries that need longer time to run.

3.2 Results of Integrating Multiple Features

To deal with the unstable problem with using individual features to make predictions, we explore the approach of integrating multiple features, which takes slightly more computational resources. This experiment was carried out in a "stand-alone" environment. Shown in Table 2, the mean relative error of stand-alone is 0.156, which is about five times better than the best results achievable by using individual features. The variation between different tiers is also significantly smaller, which indicating the model performs robustly for a variety of queries. Figure 1(a) shows that most of the log relative errors are close to zero (zero means a prediction is identical to the ground truth), and few percentages of them are larger than 0.5. Higher concurrency level also makes it more challenging to accurately predict query execution time (see Table 3). To investigate the benefits of using the "system-level" information, we performed an experiment that used only the intrinsic features (i.e., features derived from a

Table 1. The rest results of the models using individual feature in a "stand-alone" environment. The queries are grouped into 5 tiers based on their execution time. See the main text for detailed explanations.

Tier	Optimizer cost	CPU cycles	Memory data	Network data	CPU utilization
1	2.02	2.16	2.07	10.23	5.74
2	1.06	1.05	1.07	2.37	1.47
3	0.36	0.38	0.39	0.80	1.00
4	0.41	0.39	0.41	0.16	0.42
5	0.43	0.43	0.42	0.57	0.49
Overall	0.86	0.88	0.87	2.82	1.82
Tier	Disk data	Input rows	Output rows	Query concurrency	
1	10.35	3.03	3.42	5.35	
2	2.34	1.00	0.99	1.39	
3	0.77	0.37	0.34	0.97	
4	0.15	0.42	0.48	0.43	
5	0.57	0.43	0.45	0.47	
Overall	2.84	1.05	1.13	1.72	

query, excluding Query Concurrency and CPU Utilization). As shown in Table 2, the results are a little better than those of using individual features, however, are significantly worse those those of integrating all features. Thus, including the "system-level" features is essential to the accurate estimation. The above results clearly show that integrating multiple features, as we designed, can significantly improve robustness and accuracy of predictions across a wide spectrum of queries.

Table 2. The test results of the model using all features in a "stand-alone" environment. The queries were grouped into tiers in the same way to Table 1.

Tier	Mean relative error all features	Mean relative error intrinsic only
1	0.169	1.911
2	0.158	0.994
3	0.161	0.379
4	0.156	0.438
5	0.137	0.439
Overall	0.156	0.832

3.3 Noisy Environment

Most previous works conducted their experiments in "stand-alone" environments. Neglecting the effects of system status can lead to inferior performances.

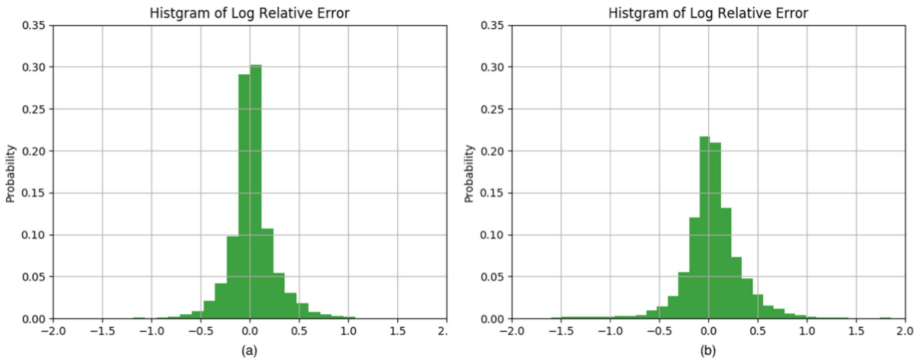

Fig. 1. Histogram of the log relative errors of the model integrating all features in a (a) "stand-alone" and (b) "noisy"environment.

Table 3. The test results of the model integrating all features in a "stand-alone" environment. The results are grouped into their corresponding concurrency levels.

Concurrency	Mean relative error	Concurrency level	Mean relative error level
1	0.024	11	0.181
3	0.119	13	0.212
5	0.094	15	0.560
7	0.229	17	0.331
9	0.203	19	0.260
10	0.186	20	0.307

When a system is busy due to whatever reasons, it will slow down the execution of database. By considering the "system-level" features, our approach can perform more robust than others. To test the robustness of our approach, we did another experiment on a public server where other users may run unknown applications on it. We call this a "noisy environment" because the data is more noisy. Shown in Fig. 1(b), the distribution of the relative error in a "noisy" environment spreads out more than that in a "stand-alone" environment, but still concentrating around 0. One encouraging observation is that the mean relative errors of short-run queries is under 0.4 even in a "noisy" condition, and the performance of our approach on long-run queries is only slightly affected by unknown "noise" in the environment.

4 Conclusion

In this paper, we present a data-driven method that applies Random Forest to build model for predicting query execution time under mixing workloads. Tested on a commercial database Vertica, we demonstrate that our approach is able to robustly and accurately estimate the running time under various levels of workloads and concurrency. We designed a set of features include the intrinsic ones, which can be extracted from each query (or the execution plan of each query), and the system-level ones (e.g., CPU Utilization and Query Concurrency). Our experiments show that each individual feature is not enough for accurately estimation the query execution time because each feature alone does not provide enough information about actual multi-thread query execution. Our experiments also show that the system-level features can be used to significantly stabilize and improve the performance of our approach. Combining the intrinsic and system-level features together in a Random Forest fashion, we are able to model the query execution characteristics well even in a noisy environment. In future, we plan to incorporate our approach into real database systems to help allocate queries into different resource pools based on the estimated query execution time, and detect poorly written queries before execution.

References

1. Agarwal, S., Mozafari, B., Panda, A., Milner, H., Madden, S., Stoica, I.: Blinkdb: queries with bounded errors and bounded response times on very large data. In: Proceedings of the 8th ACM European Conference on Computer Systems, EuroSys 2013, pp. 29–42. ACM, New York (2013). https://doi.org/10.1145/2465351.2465355
2. Chaudhuri, S., Weikum, G.: Rethinking database system architecture: towards a self-tuning RISC-style database system. In: Proceedings of the 26th International Conference on Very Large Data Bases, VLDB 2000, pp. 1–10. Morgan Kaufmann Publishers Inc., San Francisco (2000). http://dl.acm.org/citation.cfm?id=645926.671696
3. Council, T.P.P.: TPC-H benchmark specification, **21**, 592–603 (2008). http://www.tcp.org/hspec.html
4. Dageville, B., Das, D., Dias, K., Yagoub, K., Zait, M., Ziauddin, M.: Automatic SQL tuning in oracle 10G. In: Proceedings of the Thirtieth International Conference on Very Large Data Bases, VLDB 2004, vol. 30, pp. 1098–1109. VLDB Endowment (2004). http://dl.acm.org/citation.cfm?id=1316689.1316784
5. Duggan, J., Cetintemel, U., Papaemmanouil, O., Upfal, E.: Performance prediction for concurrent database workloads. In: Proceedings of the 2011 ACM SIGMOD International Conference on Management of Data, SIGMOD 2011, pp. 337–348. ACM, New York (2011). https://doi.org/10.1145/1989323.1989359
6. Golov, N., Rönnbäck, L.: Big data normalization for massively parallel processing databases. In: Jeusfeld, M.A., Karlapalem, K. (eds.) ER 2015. LNCS, vol. 9382, pp. 154–163. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-25747-1_16
7. Ho, T.K.: The random subspace method for constructing decision forests. IEEE Trans. Pattern Anal. Mach. Intell. **20**(8), 832–844 (1998)
8. Ho, T.K.: Random decision forests. In: Proceedings of the Third International Conference on Document Analysis and Recognition, vol. 1, pp. 278–282. IEEE (1995)
9. Jörg, T., Deßloch, S.: Towards generating ETL processes for incremental loading. In: Proceedings of the 2008 International Symposium on Database Engineering & Applications, pp. 101–110. ACM (2008)

10. Krompass, S., Kuno, H., Wiener, J.L., Wilkinson, K., Dayal, U., Kemper, A.: Managing long-running queries. In: Proceedings of the 12th International Conference on Extending Database Technology: Advances in Database Technology. EDBT 2009, pp. 132–143. ACM, New York (2009). https://doi.org/10.1145/1516360.1516377
11. Kuno, H., Dayal, U., Wiener, J.L., Wilkinson, K., Ganapathi, A., Krompass, S.: Managing dynamic mixed workloads for operational business intelligence. In: Kikuchi, S., Sachdeva, S., Bhalla, S. (eds.) DNIS 2010. LNCS, vol. 5999, pp. 11–26. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-12038-1_2
12. Lamb, A., et al.: The vertica analytic database: C-store 7 years later. Proc. VLDB Endow. **5**(12), 1790–1801 (2012). https://doi.org/10.14778/2367502.2367518
13. Lehner, W., Sattler, K.: Database as a service (DBaaS). In: 2010 IEEE 26th International Conference on Data Engineering (ICDE 2010), pp. 1216–1217 (2010). https://doi.org/10.1109/ICDE.2010.5447723
14. Liaw, A., Wiener, M., et al.: Classification and regression by randomforest. R News **2**(3), 18–22 (2002)
15. Macdonald, C., Tonellotto, N., Ounis, I.: Learning to predict response times for online query scheduling. In: Proceedings of the 35th International ACM SIGIR Conference on Research and Development in Information Retrieval, SIGIR 2012, pp. 621–630. ACM, New York (2012). https://doi.org/10.1145/2348283.2348367
16. Pelkonen, T., et al.: Gorilla: a fast, scalable, in-memory time series database. Proc. VLDB Endow. **8**(12), 1816–1827 (2015). https://doi.org/10.14778/2824032.2824078
17. Rahm, E., Marek, R.: Dynamic multi-resource load balancing in parallel database systems. In: Proceedings of the 21st International Conference on Very Large Data Bases, VLDB 1995, pp. 395–406. Morgan Kaufmann Publishers Inc., San Francisco (1995). http://dl.acm.org/citation.cfm?id=645921.673163
18. Stonebraker, M., et al.: MapReduce and parallel DBMSs: friends or foes? Commun. ACM **53**(1), 64–71 (2010). https://doi.org/10.1145/1629175.1629197
19. Stonebraker, M., et al.: C-store: a column-oriented DBMS. In: Proceedings of the 31st International Conference on Very Large Data Bases. VLDB 2005, pp. 553–564. VLDB Endowment (2005). http://dl.acm.org/citation.cfm?id=1083592.1083658
20. Wu, W., Chi, Y., Zhu, S., Tatemura, J., Hacigümüs, H., Naughton, J.F.: Predicting query execution time: are optimizer cost models really unusable? In: 2013 IEEE 29th International Conference on Data Engineering (ICDE), pp. 1081–1092, April 2013. https://doi.org/10.1109/ICDE.2013.6544899
21. Wu, W., Chi, Y., Hacígümüş, H., Naughton, J.F.: Towards predicting query execution time for concurrent and dynamic database workloads. Proc. VLDB Endow. **6**(10), 925–936 (2013). https://doi.org/10.14778/2536206.2536219