Learning to Optimize Federated Queries

Liqi Xu
University of Illinois (UIUC)
liqixu2@illinois.edu

Richard L. Cole
Tableau
ricole@tableau.com

Daniel Ting
Tableau
dting@tableau.com

ABSTRACT

Query optimization is challenging for any database system, even with a clear understanding of its inner workings. Consider then, query planning for a federation of third-party data sources where little detail is known. This is exactly the challenge of orchestrating data execution and movement faced by Tableau's cross-database joins feature, where the data of a query originates from two or more data sources. In this paper, we present our work on using machine learning techniques to address one of the most fundamental challenges in federated query optimization: the dynamic designation of a federation engine. Our machine learning model learns the performance and data characteristics of a system by extracting features from query plans. We further extend the ability of our model to manipulate database settings on a per query level. Our experimental results demonstrate that we can achieve a speedup of up to 10.7× compared to an existing federated query optimizer.

ACM Reference Format:
Liqi Xu, Richard L. Cole, and Daniel Ting. 2019. Learning to Optimize Federated Queries. In *International Workshop on Exploiting Artificial Intelligence Techniques for Data Management (aiDM'19), July 5, 2019, Amsterdam, Netherlands.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3329859.3329873

1 INTRODUCTION

The proliferation of datasets generated and residing in multiple sources have increased demand for querying data managed by more than one database. In response to these requests, starting in Tableau 10, the cross-database joins feature [2] allows users to create connections to more than 60 data sources and directly join datasets across platforms. This functionality addresses a fundamental requirement for users that is even more important than fast query performance, namely the ability to easily access local and remote data spread across an organization.

At a high level, the execution of a federated query, a query with cross-database joins, shares similarities with traditional query execution: a query plan is generated and then executed. In Tableau, this query plan is composed of component query plans executed on each data source [3]. The component results are then combined, e.g., via joins or unions, by a single data source chosen to be the federation engine. In such a process, the federation engine relies on individual data sources to locally optimize their individual plans and hopes that such a solution also yields a globally performant plan. However, the selection of the federation engine in Tableau is static and does not change for different workloads. This can lead to long runtimes since a poor choice of federation engine can lead to moving large amounts of data between data sources or lead to splitting the work amongst the components inappropriately.

In traditional databases, the solution would be to create a global cost model which is evaluated over a set of enumerated query plans. However, this is a challenge for federated queries as data sources act as black boxes. Furthermore, the internal cost models of the data sources may be incorrect due to poor system configurations or not accounting for unique costs associated with data transfer in federated systems.

In this paper, we present our supervised machine learning approach on *dynamically choosing the federation engine and optimizing per query to minimize query runtime.* This involves learning a model to predict runtimes as well as partially enumerating a set of plans that the costs can be evaluated on. Doing so requires both extracting information out of the data sources in order to predict runtimes as well as being able to manipulate the system to evaluate and execute the desired plans. We achieve this using mechanisms that are readily available in most database systems, namely through `EXPLAIN PLAN` and by sending appropriate hints and settings (e.g., `enable_nestloop`) with the SQL statements themselves.

One of the biggest challenges in federation engine prediction is determining the inputs, or in other words, the features of our predictive model. Because data sources are installed in a user's secure working environment, we cannot directly access statistics, such as histograms, that are typically used in cost models. However, using `EXPLAIN PLAN` allows us to access a limited number of aggregated statistics and estimates. We examined several feature sets based on these as

aiDM'19, July 5, 2019, Amsterdam, Netherlands

ACM ISBN 978-1-4503-6802-5/19/07…$15.00
https://doi.org/10.1145/3329859.3329873

well as feature sets based on the SQL texts themselves and found the following to be useful features for the model: *(i)* estimated foreign cost, i.e., the estimated cost of completing component query plan on each data source *(ii)* estimated transfer size, i.e., the estimated amount of data that needs transfer from a data source to the federation engine, and *(iii)* estimated local cost, i.e., the estimated cost of combining all results received from each data source on the federation engine before returning the final results to users.

We evaluated the performance of our approach on both Join Order Benchmark (JOB) [22] and TPC-H [7] workloads. We compared our approach against the federation engine selected directly based on the estimates in traditional query optimizers. We observed that our approach could achieve speedups of up to 4.5× and 10.7× for TPC-H and JOB workload respectively.

Outline. The chief contribution of this paper is the machine learning-based approach to optimize federated query execution. We describe our methodology of representing query related information and our learning workflow in Section 2. We present out evaluation setups and demonstrate the performance speedups we can achieve in Section 3. Lastly, we describe related work in Section 4 and discuss future work in Section 5.

2 LEARNED FEDERATED QUERY EXECUTION

In this section, we first provide background information regarding federated query processing. We then describe the feature representation as well as the machine learning workflow we use to optimize federated queries.

2.1 Federated Query Execution

A federated query, denoted as $q \in Q$, is a query that retrieves information from datasets stored on a set of data sources $D_q \subset D$, where $|D_q| > 1$. Those data sources are heterogeneous systems[1] on local or remote servers. To execute a federated query, we need to designate one data source in D_q to control and coordinate the manipulation of these data sources. We call this data source the *federation engine* for query q and denote it as FE_q. A federation engine is in charge of *a).* generating a federated query plan, *b).* receiving and processing data transferred from other data sources, and *c).* returning final results to users. An example is shown in Figure 1.

Choosing a federation engine has a significant impact on the overall runtime of a federation query. A naive approach designates the same system as the federation engine for all queries. Currently, in Tableau, by default the local Hyper [18] instance is always the federation engine. However, this approach is not optimal in many scenarios. Suppose a query accesses two tables, one small and one large. If the small table is stored on the local instance and the large table is stored on a remote data source, then the current approach may move the large dataset to Hyper and thus increase the overall query latency. Instead, a more efficient solution would designate the data source that contains the large dataset as the *alternative federation engine*; such an approach avoids costly data movement across the network and consequently improves query runtime.

[1]In this paper, we focus on relational data sources such as Microsoft SQL Server, Oracle and PostgreSQL.

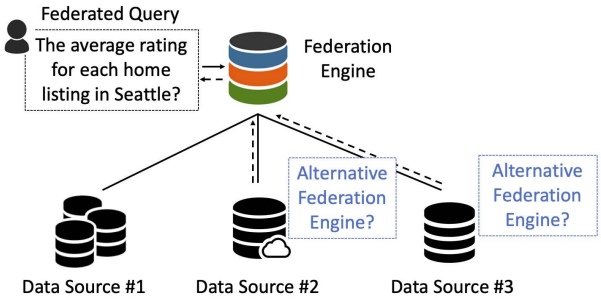


Figure 1: An Overview of a Federated Query Execution

The choice of federation engine can be seen as a query optimization problem. Given a federated query, we can enumerate a set of query plans, one (or more) for using each possible data source as the federation engine. In some cases, multiple query plans can be generated per choice of federation engine through the use of query hints or settings. We can then choose the data source with the smallest estimated runtime as the alternative federation engine for the query. The difficulty of the problem is that there is no reasonable cost model that can be assumed. While homogeneous systems can make reasonable assumptions about the relative costs of different operations, federated systems must deal with the unknown performance characteristics of each system as well as factors, such as network speed, which cannot be hard coded. Thus, one must learn the cost model.

2.2 Our Machine Learning Approach

Learning a cost model to predict query runtime is challenging since executing queries to collect training data is expensive. Thus, it is important to choose a limited number of features that are highly informative. Furthermore, these features must be computable from the limited knowledge available from the data sources.

Our approach is to generate a set of candidate queries and use the results of an `EXPLAIN PLAN` for the data sources. This allows us to indirectly access statistics in a way that is readily available in many database systems. Furthermore, this effectively allows us to perform query plan enumeration over a much larger set than the federation engine would

otherwise be able to do by itself. As each of the data sources performs query optimization for its portion of work, each enumerates a set of query plans and performs an initial pruning based on its knowledge of local costs. The federation engine takes this small set of good candidates from the set of all enumerated queries and evaluates the cost of the candidates. This accounts for global costs that data sources are not aware of, as well as the heterogeneity of the data sources. The absolute scales of component cost estimates are not directly comparable due to differences in hardware capabilities and the software. This approach can also provide a means to mitigate the effects of a poor cost model in data sources.

Feature Representation. A federated query plan, generated by a query optimizer, specifies both partial execution details on each remote data source and the final steps executed on the federation engine. Figure 2 depicts an example of a federated query plan using a query in Join Order Benchmark [22]. Here, we distribute datasets required to answer this query on three data sources and designate d_1 as the federation engine. In this example, we first join table *mk* with *k*, *cn* with *mc* respectively on d_3 and only move the partial results to d_1. On the other hand, we directly scan table *t* on d_2 and move all tuples in *t* to d_1. Note that, even for the same query, using a different data source as the alternative federation engine will result in entirely different query plans.

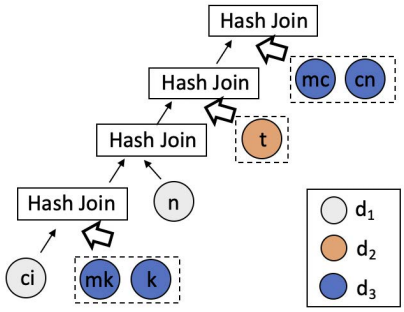


Figure 2: An Example of a Federated Query Plan

The model features we can extract from a federated system are limited. We choose a generic mechanism to obtain features by parsing the results of `EXPLAIN PLAN`s. Given a federated query plan, we extract three types of features: *(i)* the estimated foreign costs F, *(ii)* the estimated transfer sizes T, and *(iii)* the estimated local cost l. The foreign costs and transfer sizes are computed per data source and directed pair of data sources respectively. These choices target specific areas where the query optimizer's cost model is likely to have gross errors. These include network costs and foreign costs, both which involve network links and hardware that can vary significantly among machines.

We define each $f_i \in F$ as the total estimated cost of all operators processing on each foreign data source d_i. Similarly, we define each $t_i \in T$ as the total volume of data that must be moved from d_i to the federation engine via the network. Each t_i is calculated as the total number of estimated rows returned at d_i multiplied by the estimated size in bytes of the returned rows. Last but not least, we define the local cost l as the total estimated cost executed on the federation engine. Examples of this cost include the costs that result from scan and join operators that are local to the federation engine.

Workflow and Model. To train a model, we generate multiple SQL queries which add information designating the federation engine and other query settings. We then collect the query plan generated by the query optimizers for each of these and extract the machine learning features from the statistics contained in each plan. We generate training data containing the features and the actual runtimes for a set of queries that have been executed. We then fit a random forest regression model (`RF`) to predict actual runtimes. We fit separate `RF` models for different query settings as they can drastically change performance characteristics. This is equivalent to introducing a feature encoding the query setting and fully interacting it with all other feature variables.

When a new query comes in, we first extract features from the query plans using each of the data sources as the federation engine. Note that these query plans are obtained without physically running the plan[2]. We select the alternative federation engine and system setting according to the runtime predicted by the model.

We initially evaluated random forests, linear SVMs, and linear regression models while treating the problem as both a regression problem for predicting the actual runtime as well as a classification problem for predicting the best query plan. We found that random forests were more robust and the regression formulation was more useful in decreasing overall runtime as the best two plans could have very similar runtimes. We also experimented with including features based on the SQL text itself and a few other statistics such as the estimated number of rows returned by an operator. In each case, we found that they did not improve predictions given the data sizes available to us. This suggests that, locally, each data source appropriately combines raw statistics, and a machine learning model does not find gross errors that it can correct for given just these statistics.

3 PERFORMANCE EVALUATION

In this section, we evaluate the performance of our machine learning method on the ability to improve query runtime.

[2] Collecting query plans is not entirely free, primarily due to join order optimization, e.g., JOB has up to 16 joins. Also, optimizing federated query plans requires communication with remote data sources to optimize their query plan fragments. In our experiments, the median time to optimize federated queries in TPC-H and JOB was 0.5 and 1.8 seconds respectively.

We show that our approach consistently outperforms the approach that uses the query optimizer's cost model to select the federation engine and system settings on various workloads.

3.1 Experimental Setting

Federation System Setup. Our experiments were run on a federation of three PostgreSQL servers. Servers pa_1 and pa_2 were located in Palo Alto, while the third server, *sea*, was located in Seattle. Every *pa* server was a 20-core machine with 32 GB memory while *sea* server was a 16-core machine with 197 GB memory. Therefore, as shown in Table 1, network metrics vary significantly and are asymmetric with respect to source and destination. All machines were running CentOS Linux 7 and PostgreSQL v10.4 [5].

Server to Server	Bandwidth (Mbits/sec)	Ping (ms)
$pa_2 \rightarrow pa_1$	941	0.2
$pa_1 \rightarrow pa_2$	941	0.6
$sea \rightarrow pa_1$	543	41.1
$sea \rightarrow pa_2$	248	44.0
$pa_2 \rightarrow sea$	194	45.2
$pa_1 \rightarrow sea$	171	39.3

Table 1: Network Metrics by Bandwidth and Ping

PostgreSQL supports federated query execution via a Foreign Data Wrapper (FDW) [4]. When a user issues a federated query using one server as the federation engine, FDW generates a federated query plan based on its internal cost model. Each federated query plan is very similar to a query plan on a single server. It specifies the physical query operators as well as the execution order. More importantly, it orchestrates the data movement across the network by deciding which parts of the query to execute on remote and local server respectively. Even for the same query, the federated query plans are different when designating a different server as the federation engine.

Workloads. We used two workloads, the Join Order Benchmark (JOB) and the TPC-H benchmark. JOB uses a snapshot of data from the Internet Movie Database (IMDb), 3.6 GB as CSV files. There are 113 unique queries with between 3 and 16 joins. The TPC-H benchmark was run at scale factor 1, i.e., 1 GB as CSV files. The TPC-H query workload consisted of 10 streams of the 22 standard benchmark queries, for a total of 207 unique queries[3].

Our experiments used a fixed assignment of tables to servers as listed in Tables 2 and 3 respectively. This approach reflects the distributed ownership of data that is common even within a single organization. However, this framework easily supports alternative table assignments or experiments to optimize table assignment.

[3]Stream generation in TPC-H does not guarantee uniqueness across all streams, hence 207 unique queries rather than 220.

Server	Tables
pa_1	`complete_cast`, `comp_cast_type`, `link_type` `movie_info`, `movie_info_idx`, `movie_link` `title`
pa_2	`aka_title`, `company_name`, `company_type` `keyword`, `kind_type`, `movie_companies` `movie_keyword`
sea	`aka_name`, `cast_info`, `char_name`, `info_type` `name`, `person_info`, `role_type`

Table 2: JOB Table Assignment to Servers

Modeling and Metrics. To obtain experimental datasets, we ran every federated query on each of its data sources and collected query plans as well as actual query runtime. We extracted features from those query plans as described in Section 2, and used actual query runtime as our training labels. We prevented long running queries by setting a statement timeout of 120 minutes for both JOB and TPC-H. For those timeout queries, we approximated their actual time as twice the timeout value. Moreover, for each federated query, we noted the data source with the smallest actual runtime as the best federation engine (Best) of that query.

Server	Tables
pa_1	`lineitem`, `orders`
pa_2	`part`, `partsupp`
sea	`customer`, `nation`, `region`, `supplier`

Table 3: TPC-H Table Assignment to Servers

To evaluate the ability of our model to reduce runtime, we used 5-fold cross validation. Within each fold, we trained on 80% of the queries and predicted the runtime and chose the predicted best query plan for the remaining 20% of queries. (These are the original JOB and TPC-H queries without federation engine selection.) We compared our random forest (RF) method against the federated query optimizer (QO). QO made decisions directly based on cost estimates returned by the query optimizer.

We compared the performance of RF with the QO on query runtime. Specifically, we computed the relative runtime difference. We define the relative runtime difference of a method M to be the actual runtime difference between M and the best plan divided by the runtime of the best plan. This difference is zero if the selected federation engine and query settings for method M are the optimal ones.

(a) Tuning Nested Loop			(b) Default System Setting		
	TPC-H	JOB		TPC-H	JOB
Best	3.7	10.8	Best	14.6	54.5
QO	16.9	1,591.2	QO	16.9	1,591.2
RF	3.7	149.1	RF	15.2	970.0

Table 4: Average Query Runtime (in seconds) for TPC-H and JOB Workloads

3.2 Prediction Evaluation

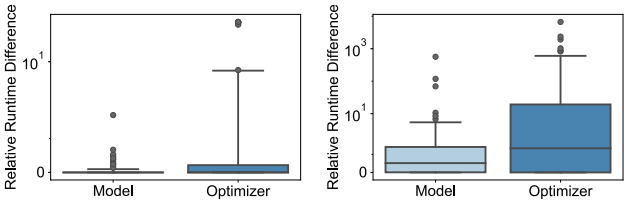
We first evaluated the effectiveness of RF on both federation engine selection and choosing query settings. We investigated specifically on query optimizer's `enable_nestloop` parameter, due to its significant impact on varying query runtime and plans.

This results in 6 possible query plans, 3 from the choice of federation engine times 2 choices on whether to enable nested loop joins. In our RF method, we collected the actual runtime for each query with nested loop on and off respectively. We then trained two separate models, one for nested loop enabled and another for nested loop disabled. At the testing phase, we predicted both federation engine and nested loop status based on the runtime estimated from all six possible cases. In this experiment, there were 152 out of 207 TPC-H queries and 66 out of 113 JOB queries that had the fastest query runtime with nested loop enabled. We reported the average runtime per query for both TPC-H and JOB workloads in Table 4(a). We also depicted the relative runtime difference in Figure 3 in log scale.

For TPC-H, RF achieved query runtime almost as fast as the best we could achieve, and had a speedup of 4.5× compared to QO. For the JOB workload, even though 6.2% of data points had an imputed training label due to timeouts, RF still significantly outperformed QO with a speedup of 10.7×. This result was because our model was able to intelligently decide the status of nested loop setting, while QO always estimated that enabling nest loop was the best choice. Moreover, we observed that the median of relative runtime difference for RF and QO is 0.5 and 1.3 respectively, while the 95th percentile of relative runtime difference for RF and QO is 6.0 and 692.0 respectively. These results demonstrate that RF can effectively learn to make better predictions from the estimates and consistently reduce the runtime compared to the federated query optimizer.

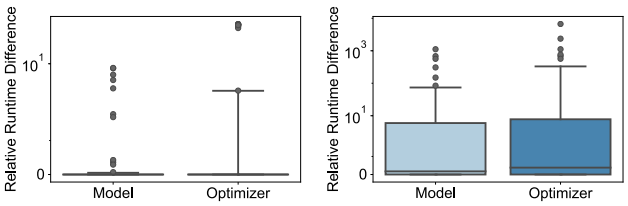
3.3 Default System Setting

We further evaluated the performance of RF over the default federated systems setting. Here, we trained our model on data points where nested loop was on and only predicted the federated engine among three data source candidates. We



(a) TPC-H **workload** (b) JOB **workload**

Figure 3: Relative Runtime Difference – The box boundary represents the 25th and 75th percentiles while the whisker represents the 95th percentiles of the distribution.



(a) TPC-H **workload** (b) JOB **workload**

Figure 4: Relative Runtime Difference with Nested Loop Enabled

show the average query runtime in Table 4(b) and relative runtime difference of this experiment in Figure 4.

For JOB queries, RF had a speedup of 1.64× compared to QO, or on average 10 minutes faster than QO per query. Moreover, the mean percentile of relative runtime difference for RF was 0.17, smaller than the mean percentile (0.5) in Section 3.2. Recall that this model was only trained on a relatively small dataset with 270 data points on average. On the other hand, RF was slightly better than QO on the TPC-H workload, achieving a speedup of 1.11× and a speedup of 3.83× with respect to Best.

Finally, we analyzed the importances of RF features for JOB queries over the default system settings. The overall importance of the three classes of RF features, estimated foreign costs, estimated transfer sizes, and estimated local cost, were 0.50, 0.25, and 0.25 respectively. All three feature classes were important to the RF model, with estimated foreign costs being the most important.

4 RELATED WORK

In this section, we discuss prior work related to this paper.

Data Federations. There are many prior examples of research into data federations that involve some measure of query optimization [11, 14, 26, 27, 29]. Notably Garlic [26] uses a wrapper architecture to provide cost and cardinality information of remote data sources, extending classic query optimization techniques to federated data. More recent work in the context of Spark SQL [10], Presto [6], and SystemPV [17] support distributed SQL processing on remote data

sources using connector architectures. Generally these system have at least basic query optimizations to push some query processing to the remote data source, with System-PV having two phases of query optimization, a global optimization phase and one specialized for the remote data. The emphasis is on mediating differences in the data models and their query languages. The same can be said for Polystores [8, 12, 28, 31].

Of these, the work that has some similarity to ours is RHEEM [8], which extends classic cost-based query optimization to multiple data sources by incorporating *a priori* detailed knowledge of their operators and using simple linear regression to calibrate cost model functions of individual query operators. The approach reimplements the functionality, including cardinality estimation and cost modeling, of remote data sources' query optimizers. We instead leverage existing database optimizers under the presumption that they can perform better local optimizations than ourselves, and we exploit mechanisms that require minimal knowledge of the inner working of data sources.

Finally, a related area is that of multi-model databases, e.g., Oracle, Drill [15], and AsterixDB [1], to name just a few. These systems do optimize queries for multiple data models; however, they are not federated systems. To sum up, except where stated otherwise, the emphasis of this body of work is on managing and specializing for the heterogeneity of the data sources rather than significantly orchestrating data execution and data movement. Additionally, none of these systems use machine learning (except RHEEM) to optimize the choice of federation engine or manipulate database settings on a per query level.

Machine Learning in Database Systems. Machine learning techniques have been applied to a number of problems and components in database systems. One fundamental problem is predicting the performance of a query. Ganapathi et al. [13] use Kernel Canonical Correlation Analysis (KCCA) model to estimate multiple metrics of interest including query runtime, records used, disk I/O and message bytes. For 85% of their test queries, the predicted runtime is within 20% of the actual query runtime. Akdere et al. [9] further improve query runtime prediction by constructing finer-grained features and training prediction models at the operator level. Examples of other machine-learning based predictions in database systems include query cardinality estimation [19, 24] and resource estimation [23]. These predictions can be used, for example, in database tuning [30] or learned index structures [20]. Machine Learning has also been used to improve query execution. Cuttlefish [16] provides a system that adaptively tunes a query plan's operators using multi-armed bandit techniques. [21] and [25] apply deep learning for better query plan enumeration.

Although tied together by the common theme of machine learning, each task faces significantly different challenges involving the inputs available to the methods, the size of the training data, and the actions that can be taken. For example, join order enumeration tasks are able to generate large training data sets since they do not run the actual query plans. They only require the relatively fast cost estimates from the query optimizer. Pure prediction tasks do not address what actions can be taken by a system. Our work examines the problem of improving query runtimes in federated query processing systems. This task involves both prediction of query runtimes and taking actions that can improve them. As these systems can be heterogeneous, a static cost model that cannot take into account the different data sources, network topology, and other factors can grossly misestimate costs. Furthermore, any machine learning system only has a limited amount of information available to it as inputs and a limited number of actions it can take. Our work shows that by exploiting common query plan features and manipulating a few database settings on a per query level, we can enumerate a small number of candidate query plans and use machine learning to choose among them to significantly improve performance.

5 CONCLUSIONS AND FUTURE WORK

In conclusion, we showed that simple machine learning techniques can significantly improve query performance, compared to the federated PostgreSQL query optimizer, when choosing alternative federation engines in a federation of data sources. We evaluated these techniques using JOB and TPC-H workloads on both federation engine selection and choosing query settings. Our machine learning model improved query runtime, in some cases up to the best possible runtime where every query was assigned to its ideal federation engine with the best possible query settings.

In the future, we plan to conduct more performance evaluations, including varying the assignment of tables to data sources, evaluating additional workloads (e.g., TPC-DS and workloads from Tableau), using a federation of data sources besides PostgreSQL, and the ability to manipulate more query settings given limited data. Additionally, there are several interesting research directions we intend to pursue. First of all, we plan to extend our methodology to broader use cases. For example, rather than limiting plans to using a single federation engine that all data sources communicate with, we consider using multiple engines to further orchestrate query plans. We also want to consider alternative machine learning goals, such as optimizing for robustness rather than best performance and accounting for uncertainty in the predictions.

REFERENCES

[1] 2019. Apache AsterixDB. (2019). Retrieved 2019-03-18 from https://asterixdb.apache.org/

[2] 2019. Integrate your data with cross-database joins in Tableau 10. (2019). Retrieved 2019-03-12 from https://www.tableau.com/about/blog/2016/7/integrate-your-data-cross-database-joins-56724

[3] 2019. Join Your Data - Tableau. (2019). Retrieved 2019-03-12 from https://onlinehelp.tableau.com/current/pro/desktop/en-us/joining_tables.htm#about-queries-and-crossdatabase-joins

[4] 2019. PostgreSQL: Documentation: 10: F.34.Âăpostgres_fdw. (2019). Retrieved 2019-03-06 from https://www.postgresql.org/docs/10/postgres-fdw.html

[5] 2019. PostgreSQL: The world's most advanced open source database. (2019). Retrieved 2019-03-06 from https://www.postgresql.org/

[6] 2019. Presto | Distributed SQL Query Engine for Big Data. (2019). Retrieved 2019-03-18 from http://prestodb.github.io/

[7] 2019. TPC-H - Homepage. (2019). Retrieved 2019-03-16 from http://www.tpc.org/tpch/

[8] Divyakant Agrawal, Sanjay Chawla, Bertty Contreras-Rojas, Ahmed K. Elmagarmid, Yasser Idris, Zoi Kaoudi, Sebastian Kruse, Ji Lucas, Essam Mansour, Mourad Ouzzani, Paolo Papotti, Jorge-Arnulfo Quiané-Ruiz, Nan Tang, Saravanan Thirumuruganathan, and Anis Troudi. 2018. RHEEM: Enabling Cross-Platform Data Processing - May The Big Data Be With You! -. *PVLDB* 11 (2018), 1414–1427.

[9] Mert Akdere, Ugur Çetintemel, Matteo Riondato, Eli Upfal, and Stanley B Zdonik. 2012. Learning-based query performance modeling and prediction. In *Data Engineering (ICDE), 2012 IEEE 28th International Conference on*. IEEE, 390–401.

[10] Michael Armbrust, Reynold S Xin, Cheng Lian, Yin Huai, Davies Liu, Joseph K Bradley, Xiangrui Meng, Tomer Kaftan, Michael J Franklin, Ali Ghodsi, et al. 2015. Spark sql: Relational data processing in spark. In *Proceedings of the 2015 ACM SIGMOD international conference on management of data*. ACM, 1383–1394.

[11] Sudarshan Chawathe, Hector Garcia-Molina, Joachim Hammer, Kelly Ireland, Yannis Papakonstantinou, Jeffrey Ullman, and Jennifer Widom. 1994. The TSIMMIS project: Integration of heterogenous information sources. (1994).

[12] Jennie Duggan, Aaron J. Elmore, Michael Stonebraker, Magdalena Balazinska, Bill Howe, Jeremy Kepner, Samuel Madden, David Maier, Timothy G. Mattson, and Stanley B. Zdonik. 2015. The BigDAWG Polystore System. *SIGMOD Record* 44 (2015), 11–16.

[13] Archana Ganapathi, Harumi Kuno, Umeshwar Dayal, Janet L Wiener, Armando Fox, Michael Jordan, and David Patterson. 2009. Predicting multiple metrics for queries: Better decisions enabled by machine learning. In *Data Engineering, 2009. ICDE'09. IEEE 25th International Conference on*. IEEE, 592–603.

[14] Laura Haas, Donald Kossmann, Edward Wimmers, and Jun Yang. 1997. Optimizing queries across diverse data sources. (1997).

[15] Michael Hausenblas and Jacques Nadeau. 2013. Apache Drill: Interactive Ad-Hoc Analysis at Scale. *Big data* 1 2 (2013), 100–4.

[16] Tomer Kaftan, Magdalena Balazinska, Alvin Cheung, and Johannes Gehrke. 2018. Cuttlefish: A lightweight primitive for adaptive query processing. *arXiv preprint arXiv:1802.09180* (2018).

[17] Manos Karpathiotakis, Avrilia Floratou, Fatma Özcan, and Anastasia Ailamaki. 2017. No data left behind: real-time insights from a complex data ecosystem. In *Proceedings of the 2017 Symposium on Cloud Computing*. ACM, 108–120.

[18] Alfons Kemper and Thomas Neumann. 2011. HyPer: A hybrid OLTP&OLAP main memory database system based on virtual memory snapshots. In *2011 IEEE 27th International Conference on Data Engineering*. IEEE, 195–206.

[19] Andreas Kipf, Thomas Kipf, Bernhard Radke, Viktor Leis, Peter A. Boncz, and Alfons Kemper. 2019. Learned Cardinalities: Estimating Correlated Joins with Deep Learning. In *CIDR*. www.cidrdb.org. http://cidrdb.org/cidr2019/papers/p101-kipf-cidr19.pdf

[20] Tim Kraska, Alex Beutel, Ed H Chi, Jeffrey Dean, and Neoklis Polyzotis. 2018. The case for learned index structures. In *Proceedings of the 2018 International Conference on Management of Data*. ACM, 489–504.

[21] Sanjay Krishnan, Zongheng Yang, Ken Goldberg, Joseph Hellerstein, and Ion Stoica. 2018. Learning to Optimize Join Queries With Deep Reinforcement Learning. (2018). arXiv:cs.DB/1808.03196

[22] Viktor Leis, Andrey Gubichev, Atanas Mirchev, Peter Boncz, Alfons Kemper, and Thomas Neumann. 2015. How good are query optimizers, really? *Proceedings of the VLDB Endowment* 9, 3 (2015), 204–215.

[23] Jiexing Li, Arnd Christian König, Vivek Narasayya, and Surajit Chaudhuri. 2012. Robust estimation of resource consumption for sql queries using statistical techniques. *Proceedings of the VLDB Endowment* 5, 11 (2012), 1555–1566.

[24] Tanu Malik, Randal C Burns, and Nitesh V Chawla. 2007. A Black-Box Approach to Query Cardinality Estimation.. In *CIDR*. 56–67.

[25] Ryan Marcus and Olga Papaemmanouil. 2018. Deep Reinforcement Learning for Join Order Enumeration. In *Proceedings of the First International Workshop on Exploiting Artificial Intelligence Techniques for Data Management, aiDM@SIGMOD 2018, Houston, TX, USA, June 10, 2018*, Rajesh Bordawekar and Oded Shmueli (Eds.). ACM, 3:1–3:4. https://doi.org/10.1145/3211954.3211957

[26] Mary Tork Roth, Laura M Haas, and Fatma Ozcan. 1999. *Cost models do matter: Providing cost information for diverse data sources in a federated system*. IBM Thomas J. Watson Research Division.

[27] Mary Tork Roth and Peter M Schwarz. 1997. Don't Scrap It, Wrap It! A Wrapper Architecture for Legacy Data Sources.. In *VLDB*, Vol. 97. 25–29.

[28] Ran Tan, Rada Chirkova, Vijay Gadepally, and Timothy G Mattson. 2017. Enabling query processing across heterogeneous data models: A survey. In *2017 IEEE International Conference on Big Data (Big Data)*. IEEE, 3211–3220.

[29] Anthony Tomasic, Louiqa Raschid, and Patrick Valduriez. 1998. Scaling access to heterogeneous data sources with DISCO. *IEEE Transactions on knowledge and Data Engineering* 10, 5 (1998), 808–823.

[30] Dana Van Aken, Andrew Pavlo, Geoffrey J Gordon, and Bohan Zhang. 2017. Automatic database management system tuning through large-scale machine learning. In *Proceedings of the 2017 ACM International Conference on Management of Data*. ACM, 1009–1024.

[31] Jingjing Wang, Tobin Baker, Magdalena Balazinska, Daniel Halperin, Brandon Haynes, Bill Howe, Dylan Hutchison, Shrainik Jain, Ryan Maas, Parmita Mehta, Dominik Moritz, Brandon Myers, Jennifer Ortiz, Dan Suciu, Andrew Whitaker, and Shengliang Xu. 2017. The Myria Big Data Management and Analytics System and Cloud Services. In *CIDR*.