Dynamic and Decentralized Global Analytics via Machine Learning

Hao Wang\textsuperscript{1}, Di Niu\textsuperscript{2}, Baochun Li\textsuperscript{1}
\textsuperscript{1}University of Toronto, \textsuperscript{2}University of Alberta
Query Processing

1. CREATE VIEW MoviesOf1996 AS
2. SELECT * 
3. FROM Movies
4. WHERE year = 1996;
5. 
6. SELECT starName, studioName 
7. FROM MoviesOf1996 JOIN StarsIn;
Decentralized Global Analytics

SQL Query

QEP Candidates

parse

select

Hive / SparkSQL

QEP

DAG

Map-Reduce

Hadoop / Spark

DC1

DC3

DC2

...
Fluctuating WAN

iperf -t 10 -P 5

Google Cloud  Taiwan  Iowa
A Toy Example

01. SELECT
02. C.name, O.orderstatus,
03. L.discount, PS.availqty
04. FROM
05. customer as C,
06. order as O,
07. lineitem as L,
08. partsupp as PS
09. WHERE O.orderkey == L.orderkey,
10. AND PS.partkey == L.partkey,
11. AND PS.suppkey == L.suppkey,
12. AND C.custkey == O.custkey
Query Plan Candidates

Plan A

- The worst plan
- The baseline

Plan B

- The initial optimal plan
- Selected by Clarinet

Plan C

- ...
A Toy Example

Plan C
- The adjusted plan
- Adapt to bandwidth fluctuation
Query Completion Time

- Centralized plan
- Baseline (Plan A)
- Plan selected by Clarinet (Plan B)
- Dynamic adjusted plan (Plan C)

$\Delta t$: The data movement time
Dynamic Query Planning

Challenges:

- Accurately estimating runtime cost of query plans.
- Minimize overall completion time of queries.
Turbo

SQL Query

QEP Candidates

QEP

DAG

Map-Reduce

Hive / Spark

Hadoop / Spark

Model Training

QEP Adjustment

Cost Estimator

DC1

DC2

DC3

...
Prediction Target

(duration, output size)
Data Generation

1. Operator -> Map stage
   \[ \sigma_{\text{price}>100} \]
   \[ \text{filter}(\text{order } o=>(o.\text{price}>100)) \]

2. Operator -> MapReduce stages
   \[ \text{map}(\text{customer } c=>(c.\text{custkey}, c.\text{values})) \]
   \[ \text{map}(\text{order } o=>(o.\text{custkey}, o.\text{values})) \]
   \[ \text{reduce}(\text{custkey}, \text{values}) \]
Data Generation

We built a new dataset of 15K samples, each recording the time it took to run a (possibly complex) query from TPC-H benchmark. Finally, we focused on generated by TPC-H to be predicted. We show that the proposed models can achieve a significant reduction in query execution time.

### 4.1 Dataset Creation

To create this dataset, we ran different data-intensive jobs. Each job involved a variety of features, including bandwidth, which was obtained via

\[ \text{avail_bw} = \frac{\text{cpu_core_num} \times \text{mem_size}}{\text{total_executor_num}} \]

This indicates both the upper bound of available bandwidth, which the distributed computing engine can hardly fully saturate due to a number of reasons mentioned earlier.

### 4.2 Crafting the Nonlinear Feature Space

In this section, we describe our machine learning models based on rule-based feature crafted for predicting the time cost and output cardinality of each pairwise join between a pair of tables, which will serve as a basis for our dynamic QEP adjustment schemes. We created a dataset of 15K samples by running the realistic TPC-H benchmark.

### Table 1: The raw features.

<table>
<thead>
<tr>
<th>Raw Features</th>
<th>Range</th>
</tr>
</thead>
<tbody>
<tr>
<td>total_exec_num</td>
<td>1 – 16</td>
</tr>
<tr>
<td>cpu_core_num</td>
<td>1 – 8 per executor</td>
</tr>
<tr>
<td>mem_size</td>
<td>1 – 4 GB per executor</td>
</tr>
<tr>
<td>avail_bw</td>
<td>5 – 1000 Mbps per link</td>
</tr>
<tr>
<td>tbl1_size, tbl2_size</td>
<td>0.3 – 12 GB per table</td>
</tr>
<tr>
<td>hdfs_block_num</td>
<td>1 – 90</td>
</tr>
</tbody>
</table>

### Table 2: The handcrafted features.

<table>
<thead>
<tr>
<th>Raw Features</th>
<th>Range</th>
</tr>
</thead>
<tbody>
<tr>
<td>total_exec_num</td>
<td>1 – 16</td>
</tr>
<tr>
<td>cpu_core_num</td>
<td>1 – 8 per executor</td>
</tr>
<tr>
<td>mem_size</td>
<td>1 – 4 GB per executor</td>
</tr>
<tr>
<td>avail_bw</td>
<td>5 – 1000 Mbps per link</td>
</tr>
<tr>
<td>tbl1_size, tbl2_size</td>
<td>0.3 – 12 GB per table</td>
</tr>
<tr>
<td>hdfs_block_num</td>
<td>1 – 90</td>
</tr>
</tbody>
</table>

Since the query completion time and output cardinality may depend on input features in a nonlinear way, we further leverage handcrafted nonlinear features that are shown in Table 2. These features are related to the query shuffling during reduce stages, which is a major bottleneck in geo-distributed analytics. We focus on different types of pairwise joins under varied combinations of input features. These features are related to the query executions and collecting the corresponding statistics, which we call key (derived) features. These selected features are further input to statistical feature engineering and selection approach.

We ran different data-intensive jobs. Each job involved a variety of features, including bandwidth, which was obtained via

\[ \text{avail_bw} = \frac{\text{cpu_core_num} \times \text{mem_size}}{\text{total_executor_num}} \]

This indicates both the upper bound of available bandwidth, which the distributed computing engine can hardly fully saturate due to a number of reasons mentioned earlier.

### Table 1: The raw features.

<table>
<thead>
<tr>
<th>Raw Features</th>
<th>Range</th>
</tr>
</thead>
<tbody>
<tr>
<td>total_exec_num</td>
<td>1 – 16</td>
</tr>
<tr>
<td>cpu_core_num</td>
<td>1 – 8 per executor</td>
</tr>
<tr>
<td>mem_size</td>
<td>1 – 4 GB per executor</td>
</tr>
<tr>
<td>avail_bw</td>
<td>5 – 1000 Mbps per link</td>
</tr>
<tr>
<td>tbl1_size, tbl2_size</td>
<td>0.3 – 12 GB per table</td>
</tr>
<tr>
<td>hdfs_block_num</td>
<td>1 – 90</td>
</tr>
</tbody>
</table>

Since the query completion time and output cardinality may depend on input features in a nonlinear way, we further leverage handcrafted nonlinear features that are shown in Table 2. These features are related to the query shuffling during reduce stages, which is a major bottleneck in geo-distributed analytics. We focus on different types of pairwise joins under varied combinations of input features. These features are related to the query executions and collecting the corresponding statistics, which we call key (derived) features. These selected features are further input to statistical feature engineering and selection approach.
Data Preprocessing

1. Handcrafting features

2. Polynomial feature crossing

3. Feature selection by LASSO path

\[ [a, b, c] \rightarrow [1, a, b, c, a^2, ab, ac, b^2, bc, c^2] \]

Table 2: The handcrafted features.

<table>
<thead>
<tr>
<th>Feature</th>
<th>Expression</th>
</tr>
</thead>
<tbody>
<tr>
<td>tbl_size_sum</td>
<td>( \text{sum}(\text{tbl1}<em>\text{size}, \text{tbl2}</em>\text{size}) )</td>
</tr>
<tr>
<td>max_tbl_size</td>
<td>( \text{max}(\text{tbl1}<em>\text{size}, \text{tbl2}</em>\text{size}) )</td>
</tr>
<tr>
<td>min_tbl_size</td>
<td>( \text{min}(\text{tbl1}<em>\text{size}, \text{tbl2}</em>\text{size}) )</td>
</tr>
<tr>
<td>1/avail_bw</td>
<td>( \frac{1}{\text{avail_bw}} )</td>
</tr>
<tr>
<td>1/total_exec_num</td>
<td>( \frac{1}{\text{total_exec_num}} )</td>
</tr>
<tr>
<td>1/cpu_core_num</td>
<td>( \frac{1}{\text{cpu_core_num}} )</td>
</tr>
</tbody>
</table>
Feature Selection

1. max_tbl_size
2. tbl_size_sum
3. min_tbl_size
4. cpu_core_num
5. max_tbl_size / bw
6. 1/bw^2
7. total_exec_num
8. mem_size
9. Other features

Coefficients vs. L1 penalty (decreasing)
Feature Selection

1. max_tbl_size
2. tbl1_size
3. tbl_size_sum
4. min_tbl_size
5. Other features

Coefficients

L1 penalty (decreasing)
Training

LASSO Regression

Linear Regression with L1 penalty

GBRT

Gradient Boosting Regression Tree

500 ternary regression trees of depth 3
Model Test

Absolute Percentage Error:

\[ APE_i = \left| \frac{y_i - h(x_i)}{y_i} \right| \times 100\% . \]

![Graph showing APE for different methods](image-url)

- **APE (%)** for Duration:
  - LASSO
  - GBRT-raw
  - GBRT

- **APE (%)** for Output Size:
  - LASSO
  - GBRT-raw
  - GBRT
Model Test

![Box plots showing APE (%) for different dataset sizes in terms of duration and output size.](image)

- **APE (%)**: The percentage of absolute prediction error is visualized for various dataset sizes (3K, 5K, 7K, 9K, 11K, 13K, 15K).
- **Duration**: The box plots for duration across different dataset sizes show the variation in APE.
- **Output Size**: Similarly, the box plots for output size across different dataset sizes also display the APE distribution.
Dynamic Planning Strategies

- **Shortest Completion Time First (SCTF)**
  
  \[ \text{duration} \]

- **Maximum Data Reduction First (MDRF)**
  
  \[ \text{data_reduction} \]

- **Maximum Data Reduction Rate First (MDRRF)**
  
  \[ \frac{\text{data_reduction}}{\text{duration}} \]
Evaluation Setup

- TPC-H benchmark
- Google Cloud
  - 33 instances across 8 regions

<table>
<thead>
<tr>
<th>Table</th>
<th>Location</th>
<th>Table</th>
<th>Location</th>
</tr>
</thead>
<tbody>
<tr>
<td>lineitem</td>
<td>Taiwan</td>
<td>customer</td>
<td>Frankfurt</td>
</tr>
<tr>
<td>region</td>
<td>Singapore</td>
<td>orders</td>
<td>Sao Paulo</td>
</tr>
<tr>
<td>supplier</td>
<td>Sydney</td>
<td>nation</td>
<td>Northern Virginia</td>
</tr>
<tr>
<td>part</td>
<td>Belgium</td>
<td>partsupp</td>
<td>Oregon</td>
</tr>
</tbody>
</table>
Query

**Turbo-SCTF**
- 25.1-38.5%

**Turbo-MDRF**
- 12.6-37.1%

**Turbo-MDRRF**
- 25.2-41.4%
The completion time distributions of pairwise joins.
Case Study

The Gantt chart of the query Q21
Related Work

A number of recent studies have attempted to improve the performance of geo-distributed data analytics. Turbo adds to the rich literature on query optimization in both distributed database systems and big data analytics frameworks. Essentially, Turbo shows how to enable the query optimizer to react to runtime dynamics.

Most existing work have explored low-layer optimizations to improve GDA query performance such as data placement and task scheduling, as summarized in Table 5. Iridium [20] seeks a tradeoff between data locality and WAN bandwidth usage by data movement and task scheduling. Geode [26] jointly exploits input data movement and join algorithm selection to minimize WAN bandwidth usage. WANalytics [27] optimizes and replicates data to minimize total bandwidth usage. JetSteam [21] uses data aggregation and adaptive filtering to support data analytics. SWAG [16] coordinates job scheduling across datacenters to take advantage of data locality and improves GDA performance. Graphene [14] packs and schedules tasks to reduce job completion times and increases cluster throughput.

The closest work to us is Clarinet [25], which selects the optimal query execution plan based on the WAN condition before the query is executed. Once a plan is selected, Clarinet leaves it oblivious to the varying runtime environment. However, most of the existing solutions require the full stack of the original data processing frameworks to be re-engineered. Turbo has carefully designed a machine learning module to enable online query planning non-intrusively. A few work have applied machine learning techniques to perform resource management [12, 19], workload classification [24], cluster configuration [9] and database management system tuning [23].

8 Conclusion

In this paper, we have present our design and implementation of Turbo, a lightweight and non-intrusive system that orchestrates query planning for geo-distributed analytics. We argue that, in order to optimize query completion times, it is crucial for the query execution plan to be adaptive to runtime dynamics, especially in wide area networks. We have designed a machine learning module, based on careful choices of models and fine-tuned feature engineering, that can estimate the time cost as well as the intermediate output size of each reduce and shuffle stage (including joins) during query execution given a number of easily measurable parameters, with an accuracy of over 90%. Based on quick cost predictions made online in a pipelined fashion, Turbo dynamically and greedily alters query execution plans on-the-fly in response to bandwidth variations. Experiments performed across geo-distributed Google Cloud regions show that Turbo reduces the query completion times by up to 41% based on the TPC-H benchmark, in comparison to default Spark SQL and state-of-the-art optimal static query optimizers for geo-distributed analytics.

<table>
<thead>
<tr>
<th>Work</th>
<th>Data Placement</th>
<th>Task Scheduling</th>
<th>Plan Optimization</th>
<th>Working Mode</th>
</tr>
</thead>
<tbody>
<tr>
<td>Geode [26]</td>
<td>√</td>
<td>√</td>
<td>√</td>
<td>static</td>
</tr>
<tr>
<td>WANalytics [27]</td>
<td>√</td>
<td>√</td>
<td>√</td>
<td>static</td>
</tr>
<tr>
<td>Iridium [20]</td>
<td>√</td>
<td>√</td>
<td>√</td>
<td>static</td>
</tr>
<tr>
<td>SWAG [16]</td>
<td>√</td>
<td></td>
<td>√</td>
<td>static</td>
</tr>
<tr>
<td>JetSteam [21]</td>
<td>√</td>
<td></td>
<td></td>
<td>static</td>
</tr>
<tr>
<td>Clarinet [25]</td>
<td>√</td>
<td>√</td>
<td></td>
<td>static</td>
</tr>
<tr>
<td>Lube [15]</td>
<td>√</td>
<td></td>
<td></td>
<td>dynamic</td>
</tr>
<tr>
<td>Graphene [14]</td>
<td></td>
<td></td>
<td>√</td>
<td>static</td>
</tr>
<tr>
<td>Turbo</td>
<td></td>
<td></td>
<td>√</td>
<td>dynamic</td>
</tr>
</tbody>
</table>
Conclusion

- Turbo: dynamic query planning with awareness of WAN bandwidths
- Data-driven cost estimation of pairwise join with accuracy over 95%
- Greedy strategies that reduces the query completion times by up to 41% based on the TPC-H benchmark
The End | Thank You