Towards Predicting Query Execution Time for Concurrent and Dynamic Database Workloads

Wentao Wu\textsuperscript{1,2}, Yun Chi\textsuperscript{2}, Hakan Hacigumus\textsuperscript{2}, Jeffrey Naughton\textsuperscript{1}

\textsuperscript{1}Dept of Computer Sciences, University of Wisconsin-Madison
\textsuperscript{2}NEC Laboratories America
Background

- Database as a service (DaaS)

Other applications
  - Admission control, query scheduling, progress monitoring, system sizing, etc.

How can we predict the **execution time** of a query **before** it runs?
Motivation

- Previous work
  - Standalone workloads [ICDE’09, ICDE’12, VLDB’12, ICDE’13]
  - Concurrent but static workloads [EDBT’11, SIGMOD’11]

- Real world database workloads
  - Dynamic: queries are not known a priori.

Our goal: Workloads that are both concurrent and dynamic!
Problem Definition

At time $t_i$, predict the \textit{(remaining)} execution time for each query in the mix.

(a) At time $t_1$

(b) At time $t_2$

(c) At time $t_3$
Main Idea

- PostgreSQL’s cost model

\[ C = n_s c_s + n_r c_r + n_t c_t + n_i c_i + n_o c_o \]

<table>
<thead>
<tr>
<th>Cost Unit</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>( c_s ): seq_page_cost</td>
<td>1.0</td>
</tr>
<tr>
<td>( c_r ): rand_page_cost</td>
<td>4.0</td>
</tr>
<tr>
<td>( c_i ): cpu_tuple_cost</td>
<td>0.01</td>
</tr>
<tr>
<td>( c_i ): cpu_index_tuple_cost</td>
<td>0.005</td>
</tr>
<tr>
<td>( c_o ): cpu_operator_cost</td>
<td>0.0025</td>
</tr>
</tbody>
</table>

- The \( n \)'s \textit{won’t} change!
  - Even if the query is running together with other queries

- Only the \( c \)'s \textit{will} change!

Wentao Wu, Yun Chi, Shenghuo Zhu, Junichi Tatemura, Hakan Hacigümüs, and Jeffrey F. Naughton, \textit{Predicting query execution time: are optimizer cost models really unusable?} In ICDE, 2013.
Main Idea (Cont.)

- The c’s change at boundaries of *phases* during execution.

- What should be a *phase* of a query?
  - A phase = an *operator*?
  - *Pipelining* of operators => *interleaved* phases!

- We define a phase to be a *pipeline*. 
Progressive Predictor

- The execution of a query mix can then be thought of as multiple stages of *mixes of pipelines*

8 *mixes of pipelines* during the execution of the 3 queries

We need a predictor for a *mix of pipelines*!
Predictors for A Mix of Pipelines

- An approach based on machine learning
- An approach based on analytic models
Machine-Learning Based Approach

- CPU and I/O interactions are different
  - Separate the modeling of CPU and I/O interactions.

- Modeling CPU interactions ($m$ CPU cores, $n$ pipelines)
  - If $m \geq n$, then $c_{cpu} = \tau$ (same as the standalone case).
  - If $m < n$, then $c_{cpu} = \frac{n}{m} \cdot \tau$, assuming fair sharing.

- Modeling I/O interactions
  - Use machine learning.
Modeling I/O Interactions

• Previous work
  - Assume that *all* the queries are known beforehand.
  - Run *sample mixes* and *train* a regression model.
  - Apply to *static* workloads (e.g., report generation).

• It cannot be directly applied to *dynamic* workloads.
  - We do not know all the queries to be run.
Observation #1. Fixed DBMS => Fixed # scan operators

Observation #2. Fixed DBMS + Fixed DB schema => Fixed # scan types

scan type = scan operator + table name (e.g., \textit{index scan} over \textit{orders})

We can apply the machine-learning idea to \textit{scan types} instead of query templates!

NB: Additional I/O’s (e.g., from hash-joins) => Additional scans
Analytic-Model Based Approach

- Problem of the machine-learning based approach
  - Infinitely many *unknown* queries/query mixes

- Model the system with a queueing network.

1. Two *service centers*: Disk, CPU.
2. Pipelines *are customers*.
3. The c’s are the *residence times per visit* of a customer.
The effect of the buffer pool
- The buffer pool cannot be modeled as a service center.

We used a model [SIGMETRICS’92]
- For the “clock” algorithm used by PostgreSQL
Experimental Settings

- PostgreSQL 9.0.4, Linux 3.2.0-26
- TPC-H 10GB database
- Multiprogramming Level (MPL): 2 to 5
- Dual Intel 1.86GHz CPU, 4GB of memory
Workloads

- 2 TPC-H workloads & 3 micro-benchmarking workloads
  - TPC-H2: 12 templates (Q7, 8, 9 are more expensive)
  - MB1: *heavy index scans* with different data sharing rate.
Baseline Approach

- For each query in the mix
  - Predict its time by using the *single-query* predictor.

- *Multiply* it with the MPL as the prediction.

- Intuitively, this approach *ignores* the impact of query interactions.
Prediction Accuracy

- On TPC-H2 (with more expensive templates)
Prediction Accuracy (Cont.)

- On MB1 (mixes of heavy index scans)
Overhead

- Both approaches need to *calibrate* the optimizer’s cost model.

- The machine-learning based approach needs a *training* stage (usually *2 days*).

- The analytic-model based approach needs to *evaluate* the analytic models (usually *< 120 ms*).
Conclusion

- To the best of our knowledge, we are the first to
  - publish a technique to predict query execution times for workloads that are both concurrent and dynamic;
  - present a systematic exploration of its performance.

- We use analytic-model based approaches in addition to machine learning as used by previous work.

- We show that our analytic-model based approach can have competitive and often better prediction accuracy than a (new) machine-learning based approach.
Q & A

- Thank you😊
From A Query Plan to Pipelines

Tables:
- Students (sid, sname)
- Enroll (sid, cid, grade)

SELECT S.sname, AVG(grade) AS gpa
FROM Students S, Enroll E
WHERE S.sid = E.sid
GROUP BY S.sname

The example query plan contains 3 pipelines with the execution order: $P_1P_2P_3$. 
More Details of Queueing Network

Residence Time

Service Time

Queueing Time

\[ R_{k,m} = \tau_k + Y_k \tau_k \sum_{j \neq m} Q_{k,j} \]

\[ Q_{k,j} = \frac{V_{k,j} R_{k,j}}{\sum_{i=1}^{K} V_{i,j} R_{i,j}} \]  
(Queue Length)

\[ Y_k = \frac{1}{C_k M} \rho^{4.464(C_k^{0.676} - 1)} \]  
(Correction Factor, \( Y_k = 1 \) if \( C_k = 1 \))

\[ \rho_k = \frac{\tau_k}{C_k} \frac{\sum_{j=1}^{K} V_{k,j}}{\sum_{i=1}^{K} V_{i,j} R_{i,j}} \]  
(Utility)
More Details of Buffer-Pool Model

- Recall the “clock” algorithm
  - The buffer pages are organized in a circular queue.

- On a buffer miss, the clock pointer scans the pages and chooses the first page with count 0 for replacement.

- If a page has a count greater than 0, then the count is decreased by 1.

- On a buffer hit, the counter of the page is reset to its maximum value.
More Details of Buffer-Pool Model (Cont.)

Model the “clock” algorithm by using a Markov chain.

\[ \sum_{p=1}^{P} S_p \left( 1 - \frac{1}{\left(1 + \frac{n_0 r_p}{m S_p}\right)^{l_p+1}} \right) - B = 0 \quad \text{(steady-state condition)} \]

\[ N_p = S_p \left( 1 - \frac{1}{\left(1 + \frac{n_0 r_p}{m S_p}\right)^{l_p+1}} \right) \quad \text{(\# pages in the buffer)} \]

\[ h_p = \frac{N_p}{S_p} \quad \text{(buffer hit rate)} \]

\[ m_p = 1 - h_p = \left[ \left(1 + \frac{n_0 r_p}{m S_p}\right)^{l_p+1} \right]^{-1} \quad \text{(buffer miss rate)} \]

expected \# accesses to a page in the partition \( p \)
Workloads

- TPC-H workloads
  - TPC-H1: 9 light to moderate TPC-H query templates
  - TPC-H2: TPC-H1 + 3 more expensive templates (Q7, 8, 9)
  - Create query mixes with Latin Hypercube Sampling (LHS).
Workloads (Cont.)

- **Micro-benchmarking workloads**
  - MB₁: mixes of *heavy index scans* with different data sharing rate.
  - MB₂: mixes mingled with both *sequential scans* and *index scans*.
  - MB₃: similar to MB₂, but we replace the scans with real *TPC-H queries* that contain the corresponding scans.
Prediction Accuracy

- On TPC-H1 (light to moderate templates)
Prediction Accuracy (Cont.)

- On TPC-H2 (with more expensive templates)
Prediction Accuracy (Cont.)

- On MB1 (mixes of heavy index scans)
Prediction Accuracy (Cont.)

- On MB2 (mixes of sequential scans/index scans)
Prediction Accuracy (Cont.)

- On MB3 (similar to MB2, but with TPC-H queries)
Sensitivity to Errors in Cardinality Estimates

- On TPC-H₁, with *biased* errors
Sensitivity to Errors in Cardinality Estimates (Cont.)

• On TPC-H₁, with *unbiased* errors
Additional Overhead (Analytic-Model Based Approach)