Adaptive query optimization in PostgreSQL

Oleg Ivanov
Postgres Professional
What is query optimization?
How does PostgreSQL optimize queries?
What is adaptive query optimization?
Machine learning and kNN method.
How to use machine learning for adaptive query optimization?
How much can it improve PostgreSQL performance?
Implementation details: AQO.
What is query optimization?

```sql
SELECT *
FROM users AS u1, messages AS m, users AS u2
WHERE u1.id = m.sender_id AND m.receiver_id = u2.id;
```
What is query optimization?

```
SELECT * 
FROM users AS u1, messages AS m, users AS u2 
WHERE u1.id = m.sender_id AND m.receiver_id = u2.id;
```
What is query optimization?

```
SELECT * 
FROM users AS u1, messages AS m, users AS u2 
WHERE u1.id = m.sender_id AND m.receiver_id = u2.id;
```
What is query optimization?

EXPLAIN SELECT *
FROM users AS u1, messages AS m, users AS u2
WHERE u1.id = m.sender_id AND m.receiver_id = u2.id;

QUERY PLAN

Hash Join  (cost=540.00..439429.44 rows=10003825 width=27)
    Hash Cond: (m.receiver_id = u2.id)
        ->  Hash Join  (cost=270.00..301606.84 rows=10003825 width=23)
            Hash Cond: (m.sender_id = u1.id)
                ->  Seq Scan on messages m  (cost=0.00..163784.25 rows=10003825 width=19)
                ->  Hash  (cost=145.00..145.00 rows=10000 width=4)
                    ->  Seq Scan on users u1  (cost=0.00..145.00 rows=10000 width=4)
                ->  Hash  (cost=145.00..145.00 rows=10000 width=4)
                    ->  Seq Scan on users u2  (cost=0.00..145.00 rows=10000 width=4)
(9 rows)
EXPLAIN SELECT *
FROM users AS u1, messages AS m, users AS u2
WHERE u1.id = m.sender_id AND m.receiver_id = u2.id;

---
Hash Join (cost=540.00..439429.44 rows=10003825 width=27)
  Hash Cond: (m.receiver_id = u2.id)
  -> Hash Join (cost=270.00..301606.84 rows=10003825 width=23)
    Hash Cond: (m.sender_id = u1.id)
    -> Seq Scan on messages m (cost=0.00..163784.25 rows=10003825 width=19)
    -> Hash (cost=145.00..145.00 rows=10000 width=4)
      -> Seq Scan on users u1 (cost=0.00..145.00 rows=10000 width=4)
  -> Hash (cost=145.00..145.00 rows=10000 width=4)
    -> Seq Scan on users u2 (cost=0.00..145.00 rows=10000 width=4)
(9 rows)

What is query optimization?
How does PostgreSQL optimize queries?

Cost-based query optimization

System R (1974)

Choose the cheapest plan among all the possible plans
How does PostgreSQL optimize queries?

\[ \text{Cost} = n_s c_s + n_r c_r + n_t c_t + n_i c_i + n_o c_o \]

<table>
<thead>
<tr>
<th>( c_i )</th>
<th>cpu_Index_tuple_cost</th>
<th>0.005</th>
</tr>
</thead>
<tbody>
<tr>
<td>( c_o )</td>
<td>cpu_Operator_cost</td>
<td>0.0025</td>
</tr>
<tr>
<td>( c_t )</td>
<td>cpu_Tuple_cost</td>
<td>0.01</td>
</tr>
<tr>
<td>( c_r )</td>
<td>random_page_cost</td>
<td>4.0</td>
</tr>
<tr>
<td>( c_s )</td>
<td>seq_page_cost</td>
<td>1.0</td>
</tr>
</tbody>
</table>
How does PostgreSQL optimize queries?

```
SELECT * FROM users
WHERE age < 25;
```

Cost = $n_s c_s + n_o \cdot c_o$

- $n_s = N_{pages}$
- $n_o = N_{tuples}$
How does PostgreSQL optimize queries?

```
SELECT * FROM users
WHERE age < 25;
```

**SeqScan**

- **Data**
- **Cost** = \( n_s c_s + n_o \cdot c_o \)
  - \( n_s = N \text{ pages} \)
  - \( n_o = N \text{ tuples} \)

**IndexScan**

- **Index**
- **Data**
- **Cost** = \( n_r \cdot c_r \)
  - \( n_r = \text{Cardinality} \)
How does PostgreSQL optimize queries?

```sql
SELECT * FROM users WHERE age < 25;
```
How does PostgreSQL optimize queries?

```
SELECT *
FROM users AS u1, messages AS m, users AS u2
WHERE u1.id = m.sender_id AND m.receiver_id = u2.id;
```
How does PostgreSQL optimize queries?

```
SELECT * 
FROM users AS u1, messages AS m, users AS u2 
WHERE u1.id = m.sender_id AND m.receiver_id = u2.id;
```
How does PostgreSQL optimize queries?

```
SELECT *
FROM users AS u1, messages AS m, users AS u2
WHERE u1.id = m.sender_id AND m.receiver_id = u2.id;
```
How does PostgreSQL optimize queries?

Optimization method

Full search

Plan's cost estimation

Cost = 439429

Cost = 304528
How does PostgreSQL optimize queries?

Optimization method

- Full Search

HashJoin

SeqScan
- users

SeqScan
- messages

SeqScan
- pictures

Cost = 439429

Plan's cost estimation

Cost = 304528
How does PostgreSQL optimize queries?

**Optimization method**
- Dynamic programming
- Genetic algorithm

**Plan's cost estimation**
- HashJoin
  - SeqScan
    - users
  - SeqScan
    - messages
  - SeqScan
    - pictures
  Cost = 439429

- MergeJoin
  - SeqScan
    - messages
  - SeqScan
    - pictures
  Cost = 304528
Dynamic programming over subsets

- System R
- Time complexity: $3^n$
- Memory consumption: $2^n$
- Always finds the cheapest plan
Genetic algorithm

- PostgreSQL
- Common and flexible method
- Can be stopped on every iteration
- No guarantees
How does PostgreSQL optimize queries?

**Optimization method**
- Dynamic programming
- Genetic algorithm

**Plan's cost estimation**
- HashJoin
- Cost = 439429
- MergeJoin
- Cost = 304528
Query clauses

Cardinality estimation

Information about stored data

PostgreSQL state

Cost estimation
Dataset:
The TPC Benchmark™DS (TPC-DS)
http://www.tpc.org/tpcds/
Error: 300 times

Error: 4 times

Dataset:
The TPC Benchmark™DS (TPC-DS)
http://www.tpc.org/tpcds/
Query clauses

Cardinality estimation

Information about stored data

PostgreSQL state

Cost estimation

How good are query optimizers, really? V. Leis, A. Gubichev, A. Mirchev et al.
SELECT * FROM users
WHERE age < 25;

Selectivity ≈ 0.3
Cardinality = $N_{tuples} \cdot \text{Selectivity}$
Only selectivities of individual clauses (i.e. *marginal* selectivities) are known

\[ \text{Selectivity}_{\text{age}} = \frac{1}{3} \]

\[ \text{Selectivity}_{\text{city}} = \frac{1}{7} \]

\[ \text{Selectivity}_{\text{age,city}} = ? \]
age < 25

city = 'Moscow'

1/7

1/21

1/3
SELECT * FROM users
WHERE age < 25 AND city = 'Moscow';

Only selectivities of individual clauses are known

The clauses are considered to be independent:

$$Selectivity_{age, city} = Selectivity_{age} \cdot Selectivity_{city}$$

With the exception of

$$Selectivity_{25 < age \ AND \ age < 57} = Selectivity_{25 < age < 57}$$
age < 25

city = 'Moscow'
SELECT * FROM users
WHERE age < 12 AND married = true;
SELECT * FROM users
WHERE age < 12 AND married = false;
SELECT * FROM users
WHERE age > 25 AND married = true
AND position = 'CTO';
Multidimensional histograms
What is adaptive query optimization?

- SQL query
- Query optimization
  - A priori cost estimation
  - Histograms
  - Cost models
  - Execution statistics
- Feedback
- Query execution
- Result
What is adaptive query optimization?

SQL query → Query optimization → Query execution → Result

A priori cost estimation → Feedback

Histograms → Cost models → Execution statistics

37
### K Nearest Neighbours method

<table>
<thead>
<tr>
<th>Age</th>
<th>Salary</th>
</tr>
</thead>
<tbody>
<tr>
<td>25</td>
<td>50</td>
</tr>
<tr>
<td>47</td>
<td>120</td>
</tr>
<tr>
<td>55</td>
<td>100</td>
</tr>
<tr>
<td>32</td>
<td>80</td>
</tr>
<tr>
<td>22</td>
<td>30</td>
</tr>
<tr>
<td>45</td>
<td>90</td>
</tr>
<tr>
<td>28</td>
<td>?</td>
</tr>
</tbody>
</table>
### K Nearest Neighbours method

<table>
<thead>
<tr>
<th>Age</th>
<th>25</th>
<th>47</th>
<th>55</th>
<th>32</th>
<th>22</th>
<th>45</th>
<th>28</th>
</tr>
</thead>
<tbody>
<tr>
<td>Salary</td>
<td>50</td>
<td>120</td>
<td>100</td>
<td>80</td>
<td>30</td>
<td>90</td>
<td>?</td>
</tr>
</tbody>
</table>
Gradient approach to kNN

<table>
<thead>
<tr>
<th>Age</th>
<th>27</th>
<th>47</th>
<th>28</th>
</tr>
</thead>
<tbody>
<tr>
<td>Salary</td>
<td>53</td>
<td>103</td>
<td>?</td>
</tr>
</tbody>
</table>
How to use machine learning for adaptive query optimization?
The object is a node with its subtree

NestedLoopJoin

u1.id = messages.sender_id

SeqScan

users u1

IndexScan

messages m

MergeJoin

u2.id = messages.receiver_id

u2.married = true
AND
u2.age < 25

IndexScan

users u2
Node cardinality is 105 tuples!

Clauses list:
users.id = messages.receiver_id
AND
users.married = true
AND
users.age < 25

Histograms

Information about the data

PostgreSQL estimator
Clause selectivities
- 0.0001
- 0.73
- 0.23

Clauses list
users.id = messages.receiver_id
AND
users.married = const
AND
users.age < const

Machine learning

Node cardinality is 1017 tuples!
Machine learning problem statement

<table>
<thead>
<tr>
<th>Features</th>
<th>Node cardinality</th>
<th>Hidden value</th>
</tr>
</thead>
<tbody>
<tr>
<td>users.id = messages.receiver_id</td>
<td>0.0001</td>
<td></td>
</tr>
<tr>
<td>users.married = const</td>
<td>0.73</td>
<td></td>
</tr>
<tr>
<td>users.age &lt; const</td>
<td>0.23</td>
<td></td>
</tr>
</tbody>
</table>

Object is a plan node

Node cardinality

?
Workflow

Query parsing → Query optimization → Query execution

Machine learning

Cardinality estimation

Learning

Query execution statistics

Machine learning data
Theoretical properties

- Will it converge?
  Yes, in the finite number of steps

- How fast will it converge?
  Don't know (in practice in a few steps)

- What guarantees on obtained plans or regressor do we have?
  Predictions are correct for all executed paths
  With perfect cost model obtained plans are not worse
How much can it improve PostgreSQL performance?

Experimental evaluation
Estimation error

- Original PostgreSQL
- Linear regression
- Neural network
- K nearest neighbours
- Gradient boosting

Number of queries
Estimation error

Number of queries

Original PostgreSQL

k nearest neighbours limited
Gradient k nearest neighbours
K nearest neighbours
Estimation error

![Graph showing estimation error over number of queries for different methods.](image-url)
Performance improvement

TPC-H fast

TPC-H slow

Original
Adaptive

+1.3%
Performance improvement

TPC-H slow

-4.4%

TPC-H fast
Performance improvement

- TPC-DS very fast: +12%
- TPC-DS fast
- TPC-DS normal
- TPC-DS slow
- TPC-DS very slow
Performance improvement

TPC-DS very fast
TPC-DS fast
TPC-DS normal
TPC-DS slow
TPC-DS very slow

Original \hspace{2cm} Adaptive

\begin{itemize}
  \item TPC-DS very fast: +24%
  \item TPC-DS fast
  \item TPC-DS normal
  \item TPC-DS slow
  \item TPC-DS very slow
\end{itemize}
Performance improvement

![Graph showing performance improvement for different TPC-DS categories: TPC-DS very fast, TPC-DS fast, TPC-DS normal, TPC-DS slow, TPC-DS very slow. The graph compares the original performance (orange) with the adaptive performance (blue). The adaptive performance shows a +41% improvement in the TPC-DS normal category.]
Performance improvement

TPC-DS very fast
TPC-DS fast
TPC-DS normal
TPC-DS slow
TPC-DS very slow

Original
Adaptive

Performance improvement +285%
Performance improvement

TPC-DS very fast
TPC-DS fast
TPC-DS normal
TPC-DS slow
TPC-DS very slow

Original  Adaptive

0  5000  10000  15000  20000

+115%
Maximum acceleration
Overheads

Experimental evaluation

Slowdown for genetic algorithm is not more than 2 seconds

Slowdown for dynamic programming is not more than 30 ms
Applicability

Complex analytical queries with a repeating pattern.
Learning progress
Learning progress

TPC-DS 69

- Execution time, s
- # iter

Graph showing learning progress with a peak at iteration 1.
Learning progress

![Graph showing the learning progress for TPC-DS 26](image)

- **Execution time, s**
- **# iter**
Current code for vanilla PostgreSQL (extension + patch): https://github.com/tigvarts/aqo

Available in Postgres Pro Enterprise
AQA: adaptive query optimization

For some queries we don't need AQA.

So we need a mechanism to determine whether the query needs AQA.
Query type is the set of queries, which differ only in their constants.

Query type:
```
SELECT * FROM users WHERE age > const AND city = const;
```

Queries:
```
SELECT * FROM users WHERE age > 18 AND city = 'Moscow';
SELECT * FROM users WHERE age > 65 AND city = 'Kostroma';
...
AQO: adaptive query optimization

aqo.mode:

• Disabled for all query types
• Enabled for all query types
• Use manual settings for known query types, ignore others
• Use manual settings for known query types, tries to tune others automatically
What is next?

SQL query → Query optimization → Query execution → Result

- A priori cost estimation
- Feedback

- Histograms
- Cost models
- Execution statistics
Questions

Contacts:
- o.ivanov@postgrespro.ru
- +7 (916) 377-55-63
Postgres Professional

http://postgrespro.ru/

+7 495 150 06 91

info@postgrespro.ru