QTune: A Query-Aware Database Tuning System with Deep Reinforcement Learning

Paper review by Victor
Structure

1. Motivations
2. How QTune works
3. Results
4. Review
Motivations

• Automatically find the best config for database
• Minimize latency
• Maximize throughput
• Individual queries or groups of queries
Existing solutions

• BestConfig, OtterTune, CDBTune
• Need large volume of high-quality training data
• Coarse-grained tuning for queries (Read-only, entire workload etc...)
• Doesn’t consider both actions and queries change the environment
QTune - Overview

• Query tuning
  • Minimizes latency

• Workload tuning
  • Maximizes throughput

• Cluster tuning
  • Group queries into different clusters
  • Optimal latency-throughput trade-off

• RL: Double-State Deep Deterministic Policy Gradient (DS-DDPG)
QTune - Overview

Query2Vector

Workload-level

Cluster-level

Vector2Pattern (DL Model)
QTune – Query Featurization

• “Query2Vector” in paper
• Turn a query into a feature vector
• Query type (Select, insert, update, delete) 4-bits of boolean flags
• Tables involved $|T|$-bits of boolean flags
• No operations (join, groupby etc...)
• Cost information from query plan (from query optimizer) $|P|$ floats
• Sum up the cost for each operation and normalize
• Unify multiple feature vectors into one:
  • Union query types (bitwise OR)
  • Sum up tables (?)
  • Sum up costs
QTune – Query Featurization
QTune – DS-DDPG

• Double state: both inner state and outer metrics
• DDPG solves the problem of infinite actions (continuous config values)
• Predictor is also DL model
• 3) Predicts change in outer metrics but not the next set of actual metrics ($S' = S + \Delta S$)
QTune – DS-DDPG – Predictor

• Training data: [(query feature vector, outer metrics, inner state, change in S)]
• Four fully connected layers
• ReLU in hidden layers
• MSE for loss function
QTune – DS-DDPG – Actor-Critic

- Training data: [(S’, Action, Reward)] from list of queries
- Update actor policy with gradient of Q-value and state
- Estimate actual state-action value with Bellman equation, reward and Q-value
- Calculate loss with squared error
QTune – DS-DDPG – Reward function

1. Define percentage changes

\[ \Delta_{0,t} = \begin{cases} 
\frac{m_t - m_0}{m_0}, & \text{the higher the better} \\
\frac{m_0 - m_t}{m_0}, & \text{the lower the better} 
\end{cases} \]

\[ \Delta_{t-1,t} = \begin{cases} 
\frac{m_t - m_{t-1}}{m_{t-1}}, & \text{the higher the better} \\
\frac{m_{t-1} - m_t}{m_{t-1}}, & \text{the lower the better} 
\end{cases} \]

2. Calculate a specific metric \( m \)

\[ r_m = \begin{cases} 
((1 + \Delta_{t-1,t})^2 - 1)|1 + \Delta_{0,t}|, & \Delta_{0,t} > 0 \\
-(((1 - \Delta_{t-1,t})^2 - 1)|1 - \Delta_{0,t}|), & \Delta_{0,t} \leq 0 
\end{cases} \]

3. Weighted mean for combining multiple metrics into reward \( R \)

\[ R = \sum w_m r_m \]

4. (Same as CDBTune)
QTune – Query clustering

- DS-DDPG has access to the configs tried for a specific query
- But expensive to use DS-DDPG to get continuous values
- Turn continuous values into discrete values like {-1, 0, 1}
- Only use the top-k frequently changed configs
- Again train a DL model to map queries to discrete config pattern vectors
- Training data: [(query vector, discrete recommended config from DS-DDPG)]
- Use DBSCAN (clustering algorithm) to group the discrete config vectors
Evaluation and results

- Evaluation on QTune techniques and internals
- Evaluation on config tuning as compared to existing methods
- Evaluation on ability to generalize with changes in a few factors

<table>
<thead>
<tr>
<th>Database</th>
<th>Knobs without restart</th>
<th>State Metrics</th>
</tr>
</thead>
<tbody>
<tr>
<td>PostgreSQL</td>
<td>64</td>
<td>19</td>
</tr>
<tr>
<td>MySQL</td>
<td>260</td>
<td>63</td>
</tr>
<tr>
<td>MongoDB</td>
<td>70</td>
<td>515</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Name</th>
<th>Sysbench</th>
<th>JOB</th>
<th>TCP-H</th>
</tr>
</thead>
<tbody>
<tr>
<td>DL</td>
<td>3792</td>
<td>8000</td>
<td>40,000</td>
</tr>
<tr>
<td>Predictor</td>
<td>3792</td>
<td>8000</td>
<td>40,000</td>
</tr>
<tr>
<td>Actor-Critic</td>
<td>1500</td>
<td>480</td>
<td>300</td>
</tr>
</tbody>
</table>

- Single machine with 128GB RAM, 5TB disk, and 4GHz CPU
- Huawei Gauss Database
Evaluation and results – QTune

• Three tuning methods

Figure 6: Performance by increasing knobs in Important First (IF) and Randomly Choosing (RC) respectively when running Sysbench (RO) on PostgreSQL.

• Throughput: Cluster > Workload > Query
• Latency: Query > Cluster > Workload (> means better)
Evaluation and results – QTune

- Running time

<table>
<thead>
<tr>
<th>Database</th>
<th>Featurization</th>
<th>Tuner</th>
<th>Vector2Pattern</th>
<th>Clustering</th>
<th>Recommendation</th>
<th>Execution</th>
<th>Overhead</th>
</tr>
</thead>
<tbody>
<tr>
<td>MySQL</td>
<td>9.37 ms</td>
<td>2.23 ms</td>
<td>0.29 ms</td>
<td>1.64 ms</td>
<td>4.36 ms</td>
<td>0.45 s - 262.9 s</td>
<td>3.8 % 0.0068 %</td>
</tr>
<tr>
<td>PostgreSQL</td>
<td>9.46 ms</td>
<td>2.38 ms</td>
<td>0.39 ms</td>
<td>2.51 ms</td>
<td>5.01 ms</td>
<td>0.46 s - 263.3 s</td>
<td>4.1 % 0.0075 %</td>
</tr>
<tr>
<td>MongoDB</td>
<td>13.48 ms</td>
<td>2.16 ms</td>
<td>0.36 ms</td>
<td>2.32 ms</td>
<td>4.31 ms</td>
<td>0.63 s - 264.5 s</td>
<td>3.5 % 0.0085 %</td>
</tr>
</tbody>
</table>

Table 5: Time distribution of queries in JOB (RO) benchmark on MySQL, PostgreSQL and MongoDB respectively. Execution is the range of time the database executes a query. Overhead is the percentage of tuning in the total time for a query.

- Low overhead during running time (and probably only running time)
Evaluation and results – Comparison with others

• BestConfig, OttorTune, CDBTune, and database admins
Evaluation and results – Comparison with others

- QTune achieves the highest throughput and lowest latency
- Better than CDBTune because CDBTune only has outer metrics in environment
Evaluation and results – Adaptability to changes

• Use models trained with one workload to tune PostgreSQL under another workload

• QTune performs the best because it takes queries into account
Evaluation and results – Adaptability to changes

• Evaluate QTune on different databases

• QTune outperforms other methods
Review

• Improvement upon CDBTune
• The lead author (Li) was involved in CDBTune
• Difference is query featurization, clustering according to pattern, and predictor model
• But makes the whole system really complicated (2 + 2 + 1 = 5 models)
• Probably takes a long time to train and tune the system itself
Discussion