ResTune

Resource Oriented Tuning Boosted by Meta-Learning for Cloud Databases

Xinyi Zhang*
Hong Wu*, Zhuo Chang, Shuowei Jin,
Jian Tan, Feifei Li, Tieying Zhang, Bin Cui
DBMS Tuning in Cloud

DBA

Tuning Hundreds of Knobs

Knobs Configurations

Apply

Various Workloads

• Database Performance
• Resource Utilization

Various Instances
Limitation of Existing Methods

#1 Not Optimizing Resource Usage

All Methods

#2 Time Consuming

Search Based

#3 High Training overhead

Reinforcement Learning Based

#4 Weak Adaptability

Bayesian Optimization Based
Our Goal

• **Goal 1:** To optimize the performance and the resource utilization simultaneously.

• **Goal 2:** To boost the tuning process with different past tuning tasks from different instance types and different workloads
Observations

The throughput and CPU usage on a real workload with 2 controlling knobs:

**Observation 1**: Throughput is not the bottleneck in most cases.

**Observation 2**: A wide range of configurations has different CPU usages but the same throughput.
Resource Oriented Tuning Problem

• We formalize the resource-oriented tuning problem as an optimization problem with SLA constraints
  • Consider a database with a continuous configuration space $\Theta$:

  $\arg\min_{\theta} f_{\text{resource}}(\theta)$
  
  s.t. $f_{\text{Throughput}} \geq SLA_{\text{Throughput}}$
  $f_{\text{Latency}} \leq SLA_{\text{Latency}}$
Solving Constrained Optimization

- Tradition Bayesian Optimization uses acquisition function (e.g., the Expected Improvement $\alpha_{EI}$) to guide the search of the optimal.

Initial Point: $x=3$
Constraint: $f_{SLA} \geq 0$

Best Feasible $f_{resource}$: None
Solving Constrained Optimization

• Tradition Bayesian Optimization uses acquisition function (e.g, the Expected Improvement $\alpha_{EI}$) to guide the search of the optimal.

- Initial Point: $x=3$
- Constraint: $f_{SLA} \geq 0$
- Best Feasible $f_{resource}$: None

![Graph showing iteration 2 with $f_{resource}$ and $f_{SLA}$ over the range $x = -10.0$ to $10.0$ with Ground Truth, GP Regressor, Acquisition Function, Uncertainty, Infeasible Point, Feasible Point, and Next Query markers.](image-url)
Solving Constrained Optimization

- Tradition Bayesian Optimization uses acquisition function (e.g., the Expected Improvement $\alpha_{EI}$) to guide the search of the optimal.

Initial Point: $x=3$

Constraint: $f_{SLA} \geq 0$

Best Feasible $f_{resource}$: None
Solving Constrained Optimization

• Tradition Bayesian Optimization uses acquisition function (e.g, the Expected Improvement $\alpha_{EI}$) to guide the search of the optimal.

Initial Point: $x=3$
Constraint: $f_{SLA} \geq 0$

Best Feasible $f_{resource}$: None
Solving Constrained Optimization

• Tradition Bayesian Optimization uses acquisition function (e.g, the Expected Improvement $\alpha_{EI}$) to guide the search of the optimal.

Initial Point: $x=3$
Constraint: $f_{SLA} \geq 0$

Best Feasible $f_{resource}$: None
Solving Constrained Optimization

- Tradition Bayesian Optimization uses acquisition function (e.g., the Expected Improvement $\alpha_{EI}$) to guide the search of the optimal.

Initial Point: $x=3$
Constraint: $f_{SLA} \geq 0$

Best Feasible $f_{resource}$: None
Solving Constrained Optimization

• Tradition Bayesian Optimization uses acquisition function (e.g., the Expected Improvement $\alpha_{EI}$) to guide the search of the optimal.

Initial Point: $x=3$
Constraint: $f_{SLA} \geq 0$

Best Feasible $f_{resource}$: None
Solving Constrained Optimization

• Tradition Bayesian Optimization uses acquisition function (e.g., the Expected Improvement $\alpha_{EI}$) to guide the search of the optimal.

Initial Point: $x=3$
Constraint: $f_{SLA} \geq 0$

Best Feasible $f_{resource}$: None
Solving Constrained Optimization

- Tradition Bayesian Optimization uses acquisition function (e.g., the Expected Improvement $\alpha_{EI}$) to guide the search of the optimal.

Initial Point: $x=3$
Constraint: $f_{SLA} \geq 0$

Best Feasible $f_{resource}$: None
Solving Constrained Optimization

• Tradition Bayesian Optimization uses acquisition function (e.g, the Expected Improvement $\alpha_{EI}$) to guide the search of the optimal.

Initial Point: $x=3$
Constraint: $f_{SLA} \geq 0$

Best Feasible $f_{resource}$: None
To solve our constrained optimization problem, we extend the acquisition function:

\[
\alpha_{CEI} = \alpha_{EI} \times \text{Prob}(\text{feasibility})
\]

We also use Gaussian Process to model \(\text{Prob}(\text{feasibility})\).
Guiding Search in Feasible Region

Initial Point: \( x = 3 \)
Constraint: \( f_{\text{SLA}} \geq 0 \)

Best Feasible \( f_{\text{resource}} \): 1.1656
Guiding Search in Feasible Region

Initial Point: $x=3$
Constraint: $f_{SLA} \geq 0$

Best Feasible $f_{resource}$: 1.1656
Guiding Search in Feasible Region

Initial Point: x=3
Constraint: $f_{SLA} \geq 0$

Best Feasible $f_{resource} = 0.9071$
Guiding Search in Feasible Region

Initial Point: $x = 3$
Constraint: $f_{SLA} \geq 0$

Best Feasible $f_{resource}: 0.9071$
Guiding Search in Feasible Region

Initial Point: \( x = 3 \)
Constraint: \( f_{SLA} \geq 0 \)

Best Feasible \( f_{resource} \): 0.9071
Guiding Search in Feasible Region

Initial Point: $x=3$
Constraint: $f_{SLA} \geq 0$

Best Feasible $f_{resource}: 0.9071$
Guiding Search in Feasible Region

Initial Point: $x=3$
Constraint: $f_{SLA} \geq 0$

Best Feasible $f_{resource}$: 0.8991
Guiding Search in Feasible Region

Initial Point: $x=3$
Constraint: $f_{SLA} \geq 0$

Best Feasible $f_{resource}$: 0.8991
Guiding Search in Feasible Region

**Iteration 9**

Initial Point: $x=3$

Constraint: $f_{SLA} \geq 0$

Best Feasible $f_{resource}: 0.8991$
Guiding Search in Feasible Region

Initial Point: \( x=3 \)
Constraint: \( f_{SLA} \geq 0 \)

Best Feasible \( f_{resource} \): 0.8991
Boosting Tuning Process

• The same workloads running on different hardware share information for tuning knobs.

• Even for different workloads, the relationship between hidden features can lead to knowledge sharing.
Boosting Tuning Process: Meta-Learning

- Human learns across tasks.
- Why? Require less trial-and-error, less data

![Diagram showing the process of meta-learning across tasks](image-url)
Knowledge Extraction

The prior knowledge is extracted from historical tuning tasks by ensemble.

\[f_1, f_2, \ldots, f_n\]

\[f = \sum_{j=1}^{30} w_j f_j\]
How to determine the weights?

Learning from Meta-Feature
- Static
- Good initialization

Learning from Model Predictions
- Dynamic
- Avoid over-fitting
Learning from Meta-Feature

• Meta-features: measurable properties of tasks
• ResTune learns the meta-feature by workload characterization.

A Workload characterization pipeline

Workload j → TF-IDF → Random Forest Model → Meta-Feature
Learning from Meta-Feature

- The static weight is calculated by the distance between meta-features.

\[
\|m_T - m_j\|
\]

Similarity

Meta-feature \(m_T\)

New Task

Meta-feature \(m_1\)

Task 1

Meta-feature \(m_2\)

Task 2

Meta-feature \(m_3\)

Task 3

\[\ldots\]

Task \(n\)

Meta-feature \(m_n\)
Learning form Model Predictions

• We define base-learners’ similarity in terms of how accuracy base-learner can predict the performance of the target task.

• Challenge: The performances can differ in scale significantly among various hardware environments in the cloud.
Learning form Model Predictions

• Our observation: the actual values of the predictions do not matter, since we only need to identify the location of the optimum!
• We calculate the ranking loss of base learners against target observations.

Target ranking: (Ground Truth)

Base-learner $j$ ranking:

\[
\text{Ranking Loss for } j = \frac{\# \text{Misranking pairs}}{\# \text{Pairs}} = \frac{6}{12}
\]
Adaptive weight schema

• Static Weight Assignment:
  • Meta-features gives a coarse-grained abstraction about task properties.
  • Suggesting knobs that are promising according to similar historical tasks.

• Dynamic Weight Assignment:
  • Ranking of model predictions measures the similarity of tasks in the optimization problem.
  • Avoiding over-fitting by shrinking historical base learners' weight.
System Architecture of ResTune
Experimental Study

• DBMS: version 5.7 of MySQL RDS
• Hardware instances:

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
<th>F</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPU</td>
<td>48 cores</td>
<td>8 cores</td>
<td>4 cores</td>
<td>16 cores</td>
<td>32 cores</td>
<td>64 cores</td>
</tr>
<tr>
<td>RAM</td>
<td>12GB</td>
<td>12GB</td>
<td>8GB</td>
<td>32GB</td>
<td>64GB</td>
<td>128GB</td>
</tr>
</tbody>
</table>

• Workloads:
  • Three Benchmark workloads: SYSBENCH，TPC-C，Twitter
  • Two real world workloads: Hotel，Sales

• Data Repository:
  • We collect workload features and observation histories of 34 past tuning tasks on instances A and B as our meta-data
Experimental Study

• Baselines:
  • **Default**: The default knobs provided by experienced DBA;
  
  • **iTuned**: We change its objective to minimizing the resource utilization;
  
  • **OtterTune-w-Con**: We replace OtterTune’s acquisition function to our designed CEI to guide search in feasible region;
  
  • **CDBTune-w-Con**: We modify its reward function to encourage the agent to minimize resource usage and satisfy the SLA;
  
  • **ResTune-w/o-ML**: ResTune without Meta-Learning;
  
  • **ResTune**: Our approach that uses the meta-learner to boost the tuning.

iTuned [VLDB 2009]; OtterTune [SIGMOD 2017] ; CDBTune [SIGMOD 2019]:
Efficiency Comparison

Takeaway:
• ResTune can reduce the default CPU usage by 50.1% on average and guarantee the SLA.
• ResTune-w/o-ML performs much better than iTuned and CDBTune-w-Con.
• With meta-learning design, ResTune achieves 18.6X speedup than OtterTune-w-Con in SYSBENCH and 7.38X speedup on average.
Evaluation on Adaptability

• Hardware Adaption
  • B to A
  • A to B
  • AB to C, D, E and F respectively

• Workload Adaption
  • holding out the target workload’s data from the data repository
Performance Adapting to Different Hardware Environments

(a) SYSBENCH (B to A)
(b) Twitter (B to A)
(c) TPC-C (B to A)
(d) Hotel (B to A)
(e) Sales (B to A)

(f) SYSBENCH (A to B)
(h) Twitter (A to B)
(g) TPC-C (A to B)
(i) Hotel (A to B)
(j) Sales (A to B)

Performance Adapting to Different Workloads

(a) SYSBENCH
(b) Twitter
(c) TPC-C
(d) Hotel
(e) Sales
Evaluation on Adaptability

<table>
<thead>
<tr>
<th>Instance</th>
<th>Improvement</th>
<th>C</th>
<th>D</th>
<th>E</th>
<th>F</th>
</tr>
</thead>
<tbody>
<tr>
<td>SYSBENCH</td>
<td>Restune</td>
<td>5.02%</td>
<td>8.13%</td>
<td>17.16%</td>
<td>20.38%</td>
</tr>
<tr>
<td></td>
<td>Restune-w/o-ML</td>
<td>3.34%</td>
<td>7.58%</td>
<td>16.76%</td>
<td>19.96%</td>
</tr>
<tr>
<td>Iteration</td>
<td>Restune</td>
<td>37</td>
<td>64</td>
<td>100</td>
<td>35</td>
</tr>
<tr>
<td></td>
<td>Restune-w/o-ML</td>
<td>57</td>
<td>80</td>
<td>115</td>
<td>53</td>
</tr>
<tr>
<td></td>
<td>Speed Up</td>
<td>35%</td>
<td>20%</td>
<td>14%</td>
<td>34%</td>
</tr>
<tr>
<td>TPC-C</td>
<td>Restune</td>
<td>4.96%</td>
<td>19.22%</td>
<td>33.26%</td>
<td>47.60%</td>
</tr>
<tr>
<td></td>
<td>Restune-w/o-ML</td>
<td>2.78%</td>
<td>18.28%</td>
<td>33.09%</td>
<td>42.62%</td>
</tr>
<tr>
<td>Iteration</td>
<td>Restune</td>
<td>12</td>
<td>25</td>
<td>45</td>
<td>18</td>
</tr>
<tr>
<td></td>
<td>Restune-w/o-ML</td>
<td>99</td>
<td>47</td>
<td>79</td>
<td>25</td>
</tr>
<tr>
<td></td>
<td>Speed Up</td>
<td>87.87%</td>
<td>46.80%</td>
<td>43.03%</td>
<td>28%</td>
</tr>
</tbody>
</table>

Hardware Adaptation on More Instances
Tuning other types of Resources

- Other types of resources
  - I/O (BPS and IOPS)
  - Memory

- Takeaway:
  - ResTune reduces 87% of I/O, and 39% of memory on average.
Thanks for Listening!

My E-mail: zhang_xinyi@pku.edu.cn
# Execution Time Breakdown

<table>
<thead>
<tr>
<th>Phase</th>
<th>ResTune</th>
<th>ResTune-w/o-ML</th>
<th>iTuned</th>
<th>CDBTune-w-Con</th>
<th>OtterTune-w-Con</th>
</tr>
</thead>
<tbody>
<tr>
<td>Meta-Data Processing</td>
<td>0.653s~1.983s</td>
<td>/</td>
<td>/</td>
<td>/</td>
<td>/</td>
</tr>
<tr>
<td>Model Update</td>
<td>0.312s~2.298s</td>
<td>0.649s</td>
<td>0.151s</td>
<td>0.586s</td>
<td>11.347s</td>
</tr>
<tr>
<td>Knob Recommendation</td>
<td>5.115s</td>
<td>1.907s</td>
<td>0.912s</td>
<td>0.005s</td>
<td>4.457s</td>
</tr>
<tr>
<td>Target Workload Replay</td>
<td>182.237s(95.1%)</td>
<td>182.237s(98.6%)</td>
<td>182.186(99.4%)</td>
<td>182.336s(99.7%)</td>
<td>182.337s(92.0%)</td>
</tr>
<tr>
<td>Total Time</td>
<td>191.630s</td>
<td>184.793s</td>
<td>183.245s</td>
<td>182.927s</td>
<td>198.141s</td>
</tr>
</tbody>
</table>