Automatic Database Management System Tuning Through Large-scale Machine Learning

Dana Van Aken
Andrew Pavlo
Geoffrey J. Gordon
Bohan Zhang
Weakness in manually DBMS tuning

• Dependencies between different knobs
• Changing of performance is irregular
• Configurations depend on specific platform
• Tuning Complexity

• -> how to tune DBMS automatically?
OtterTune: an automatic tuning system
OtterTune: an automatic tuning system

1. Select a metric to be optimized
2. Benchmark and controller measure DBMS independent external metrics
3. Collect DBMS-specific internal metrics
4. Store result into its repository (dataset)
5. Compute the next configuration
   - Try to understand the target workload, and map it to a similar DBMS platform from repository
   - Recommend a new configuration that is specially designed for current platform
Workload characterization

• To discover a model that could identify which previously seen workloads in the repository are similar to target workload.

• Use DBMS’s internal runtime metrics to characterize workload.

• Cons:
  – Accurate
  – Could be directly affected by knobs’ settings
Workload characterization

• 1. Collect DMBS runtime statistics

• Save as K/V pairs in its repository

• Limitation: only consider global knobs
Workload characterization

• **2. Remove redundant metrics** in order to reduce the search space of ML algorithms

![Diagram showing the process: High dimensional DBMS metric data → Factor Analysis → Low dimensional DBMS metric data → K-means → Clustered meaningful groups]
Workload characterization

• 2. Remove redundant metrics in order to reduce the search space of ML algorithms

• The one closet to cluster center as the representation of all metrics in this cluster

(a) MySQL (v5.6)

(b) Postgres (v9.3)

Figure 4: Metric Clustering – Grouping DBMS metrics using $k$-means based on how similar they are to each other as identified by Factor Analysis and plotted by their ($f_1$, $f_2$) coordinates. The color of each metric shows its cluster membership. The triangles represent the cluster centers.
Identify important knobs

• Identify which knobs have the strongest impact on the target objective function.

• Use Lasso, a feature selection technique for linear regression, to expose the knobs that have strongest correlation to the system’s overall performance.
Identify important knobs

• Feature selection with Lasso: a Linear regression method

• Cost function: \[ J(w) = \frac{1}{m} \sum_{i=1}^{m} (y_i - w^T x_i)^2 + \lambda ||w||_1 (\lambda > 0) \]

• \( X \): DBMS’s knobs

• \( Y \): metrics collected during observation period

• Use \textit{Lasso Path Algorithm} to determine the order of importance of the DBMS knobs (Appendix A)
Automated tuning

• **STEP1 Workload Mapping**
  - Find a workload in its repository which is most similar with the target DBMS’s workload
  - For each metric, build a matrix from the data in repository

S[num_of_metrics][num_of_workload][configuration]
S[m][i][j] == The value of metric $m$ observed when executing workload $i$ with configuration $j$

• Workload mapping: by calculating Euclidean distance(ED[m][i]) between the vector of measurements for target workload and each S[m][i]

• Score of workload $i$ = Average(ED[m][i] for each $m$), and select the workload with minimum score
Automated tuning

• **STEP2 Configuration recommendation**

• Use *Gaussian Process regression*

• Find the best configuration in each observation period by choosing the strategy with the greatest expected improvement

  – (1) *exploration*: searching an unknown region in its GP (workloads with little to no data)

  – (2) *exploration*: selecting a configuration that is near the best configuration in its GP
Experimental evaluation

• 1. Platform

• OLTP workloads:
  – DBMS: MySQL, Postgres
  – 5-minute observation periods
  – Target metric: 99%-tile latency

• OLAP workloads:
  – DBMS: Active Vector
  – Workloads: TPC-H
  – Variable-length(total exec time) observation periods
  – Target metric: total exec time of the workload
Experimental evaluation

2. Generate initial training data in repository

- Workloads: Permutations of YCSB and TPC-H
- Knobs configurations: random values within valid range
- Execute over 30k experiments on each DBMS with different workload and knob configurations.
Experimental evaluation

• 3. Analysis of OtterTune when optimizing different numbers of knobs

• To prove that OtterTune can identify the appropriate number of knobs to be tuned. (balance between DBMS performance and tuning complexity)

• 2 settings: fixed number of knobs & increase the number of knobs gradually
3. Analysis of OtterTune when optimizing different numbers of knobs

- **A:** MySQL + TPC-C
  - Incremental method is the best
  - Larger number of knobs have little improvement

- **B:** Postgres + TPC-C
  - Incremental method and 4 knobs are the best
  - Incremental method allows exploring and optimizing the configuration space for a small set of the most impactful knobs, before expanding its scope to consider the others

- **C:** Vector + TPC-H
  - Incremental method, 8 knobs, and 16 knobs are the best
  - Incremental method is the best approach
Experimental evaluation

• 4. Show how learning from data in repository (previous tuning sessions) improve OtterTune’s ability to find a good knob configuration

• Compare with another tool “iTuned”

• OtterTune: trains its GP models using the data from the most similar workload mixed with the data determined in the last workload mapping stage. Use incremental method.

• iTuned: does not train its GP models using data collected from previous tuning sessions (The initial configuration is generated by a stochastic sampling technique). It start use incremental knob only after running initial set of experiments.
Experimental evaluation

- 4. Show how learning from data in repository (previous tuning sessions) improve OtterTune’s ability to find a good knob configuration
  - A: TPC-C
    - OtterTune find better configuration in less time
    - Trained GP model in OtterTune have a better understanding of the configuration space
    - OtterTune achieved lower latency.
  - C: TPC-H
    - OtterTune achieved lower latency
Experimental evaluation

5. Analysis the amount of time that OtterTune spends in the different parts of its tuning algorithm

- **Workload Execution**: The time that it takes for the DBMS to execute the workload in order to collect new metric data.
- **Prep & Reload Config**: The time that OtterTune’s controller takes to install the next configuration and prepare the DBMS for the next observation period (e.g., restarting if necessary).
- **Workload Mapping**: The time that it takes for OtterTune’s dynamic mapping scheme to identify the most similar workload for the current target from its repository. This corresponds to Step #1 from Sect. 6.1.
- **Config Generation**: The time that OtterTune’s tuning manager takes to compute the next configuration for the target DBMS. This includes the gradient descent search and the GP model computation. This is Step #2 from Sect. 6.2.
Experimental evaluation

- 5. Analysis the amount of time that OtterTune spends in the different parts of its tuning algorithm
Experimental evaluation

6. Compare the configuration generated by OtterTune with the one provided by human DBA, open-source tuning advisor tools, and cloud DBaaS provider (Amazon RDS).
Future Work

• Hardware capabilities
• Turning component-specific knobs
• Max num of log files in Sec7.6 (optimize several metrics simultaneously)
• Online training without the type of workload specified