Automatic Database Manage System Tuning Through Large-scale Machine Learning

Reviewed by Yinxuan Feng

Table of Content

What is this paper about?

- Goal: DBMSs need tuning to optimize
- Problem: Tuning is
  - Required
  - Hard
  - Expensive
- The paper: uses ML to
  - Find knobs
  - Find similar workloads
  - Give configurations
- OtterTune https://db.cs.cmu.edu/projects/ottertune/

http://www.vldb.org/pvldb/vol11/p1910-zhang.pdf

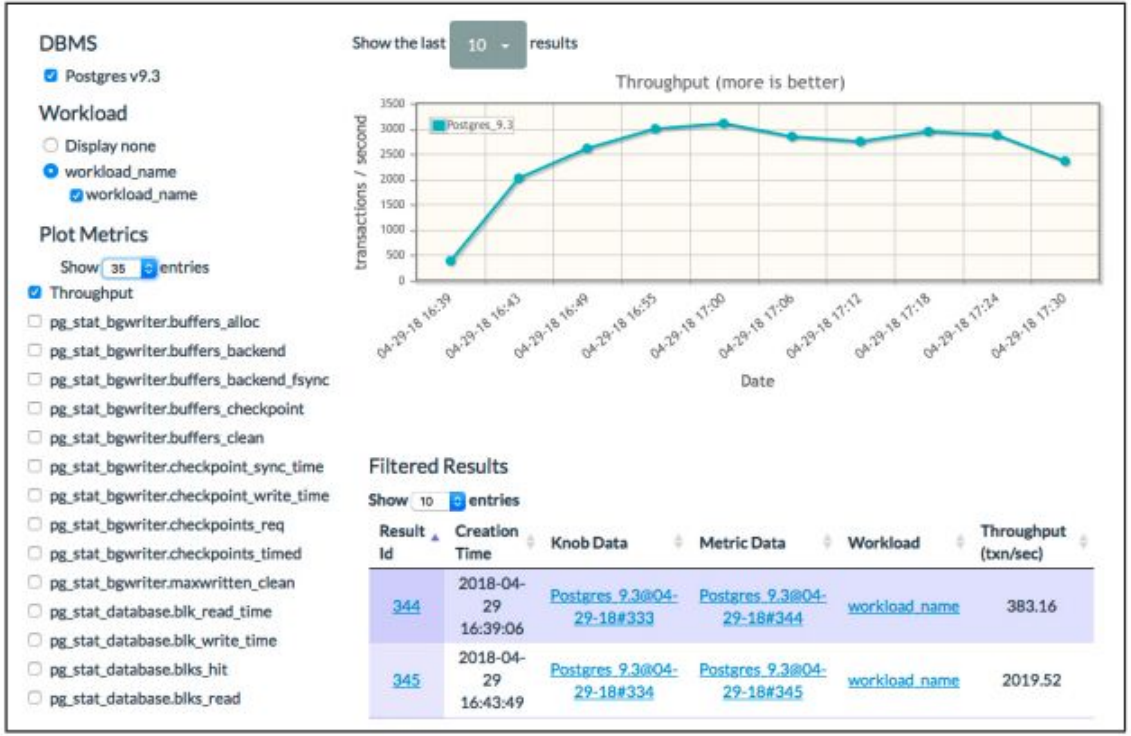
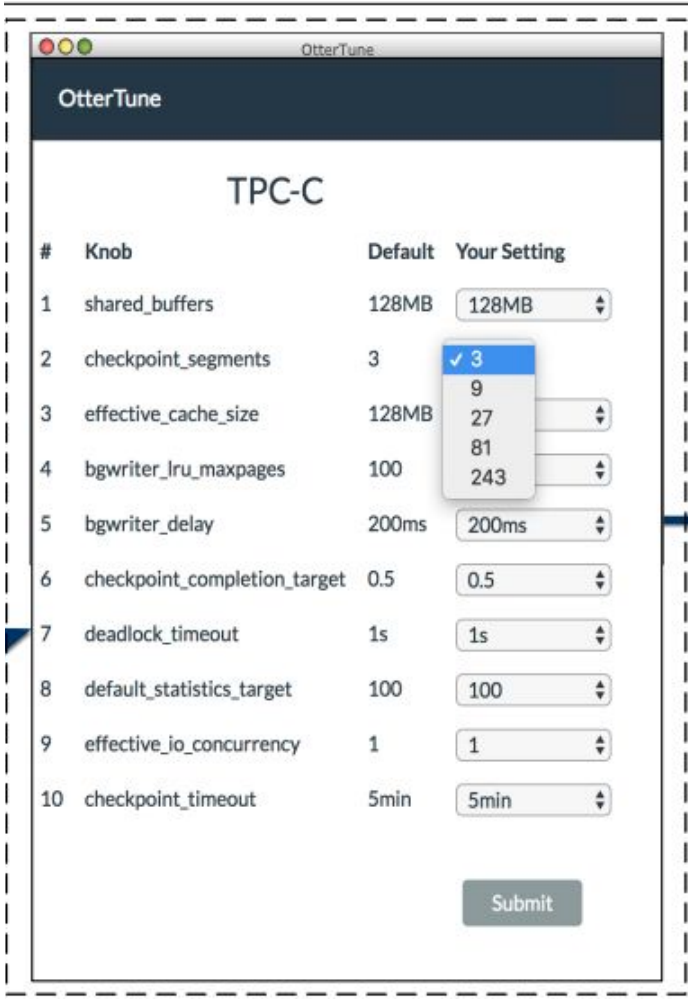
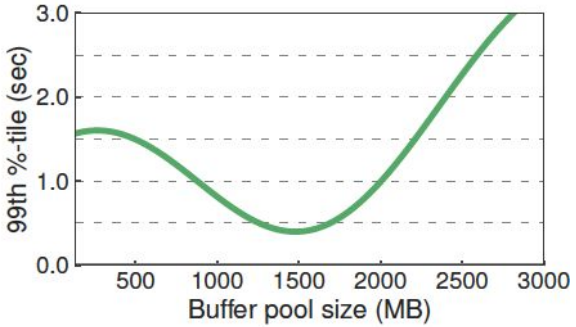


Figure 2: Tuning Session – The user can view the performance of the target DBMS as it tries the configurations recommended throughout the tuning session. Users can also view detailed information about the knob settings and metric values collected during each observation period.

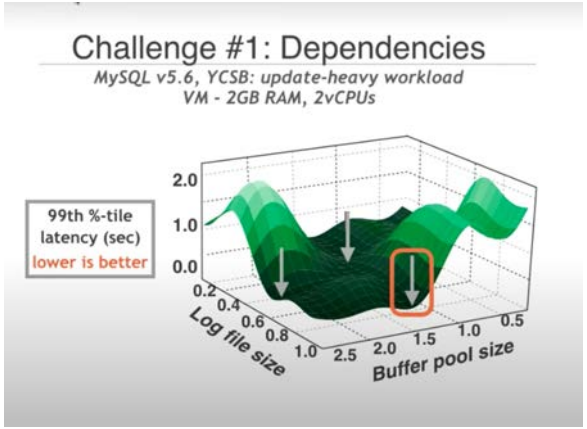


Why is it hard?

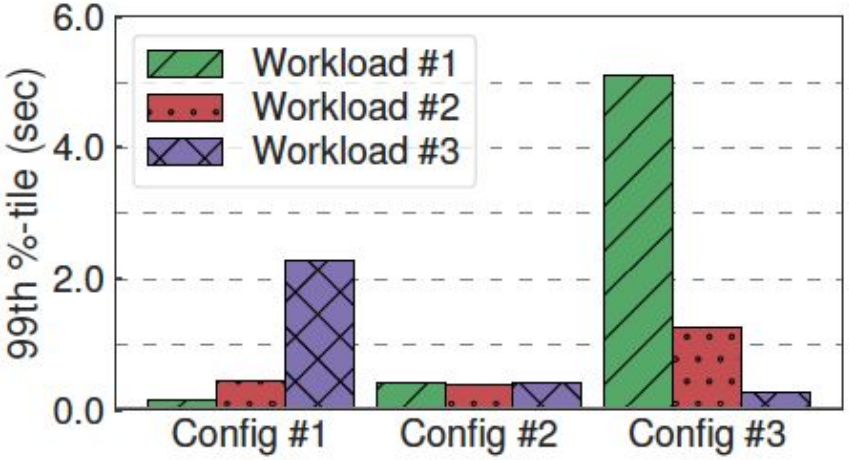
- **NP-hard**.
- Knobs are
  - **Not standardized**
  - **Not independent**
  - **Not have universal effects**
  - **Continuous**
- Their effects are
  - **not documented**
  - **expensive** to learn
- Require **expensive** human experts
- And they could **fail**
- **More** knobs over time



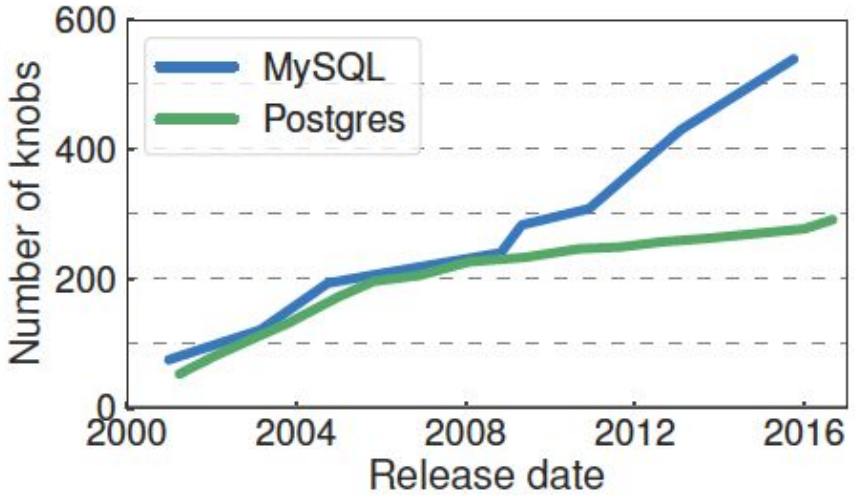
(b) Continuous Settings



More difficulties



(c) Non-Reusable Configurations



(d) Tuning Complexity

Why is it hard? Continued

- Previous attempts are **not general-purposed**.
- Some attempts still require **manual operations**
  - Deploy a second copy
  - Map dependencies between knobs
  - Guide training process
- Knowledge **cannot be transferred**.
- Old way: **trial and error**.
  - Tedious, expensive, inefficient

Contributions of this paper

1. **Automated** approach that **continuously**
    a. **Collects** data
    b. **Optimize** the DBMS
2. Uses **supervised** and **unsupervised** ML
    a. **Select** knobs
    b. **Map** workloads
    c. **Recommend** knob settings

System Overview

1. Sets target
2. Observation period
3. Stores in repository
4. Computer config

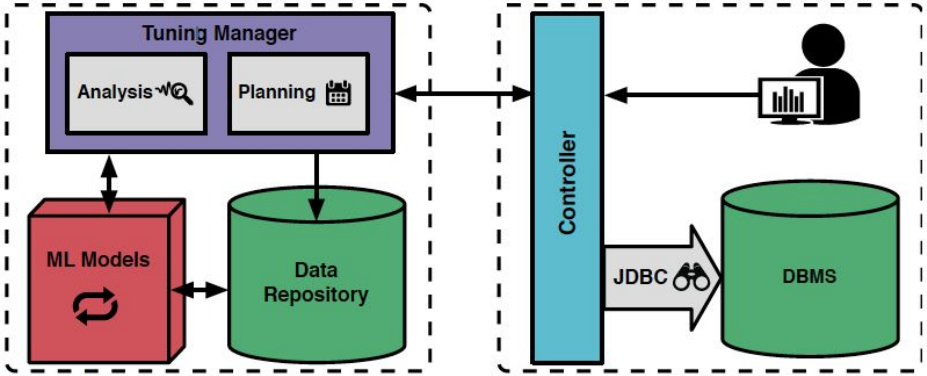


Figure 2: OtterTune Architecture – An overview of the components in the OtterTune system. The controller connects to the DBMS and collects information about the performance of the system. This information is then sent to the tuning manager where it is stored in its repository. It then builds models that are used to select an optimal configuration for the DBMS.

Assumptions and Limitations

- Priviledge
- No restart time considered
- Knows whether knobs need restart
- Database is reasonable.
  - Proper indexes
  - Materialized views
  - etc.

Workload Characterization

1. Distinguish the target workload
    a. Analyze the target workload at the logical level.
        i. Schema -> metrics
            1. Number of tables/columns accessed per query
            2. Read/write ratio of transactions
        ii. “what-if” optimizer API
    b. Internal runtime metrics.
        i. Number of pages read/written
        ii. Query cache utilization
        iii. Locking overhead.

Statistics collection

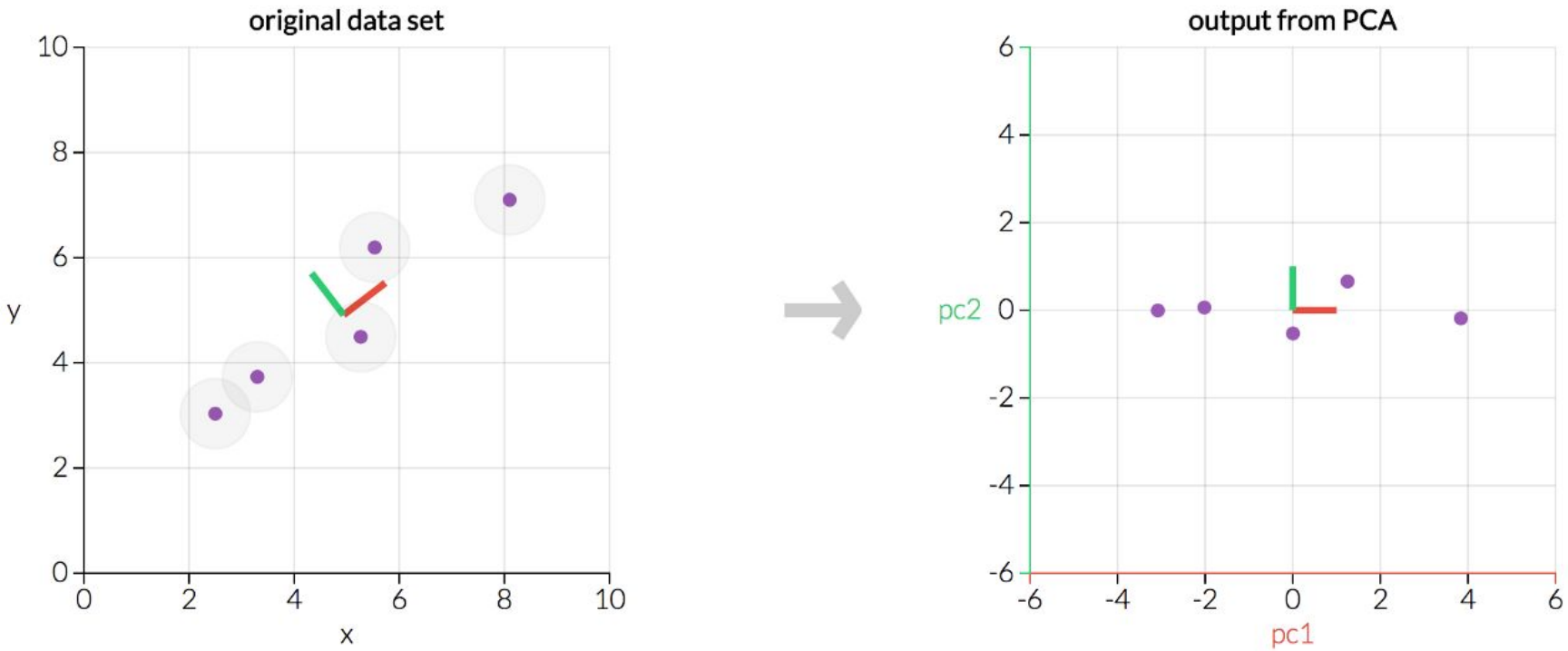
1. Reset
2. Collects all numeric metric
3. Stores the data into the repository.
4. Prefix the name of the sub-element to the metric’s name.

Pruning redundant metrics

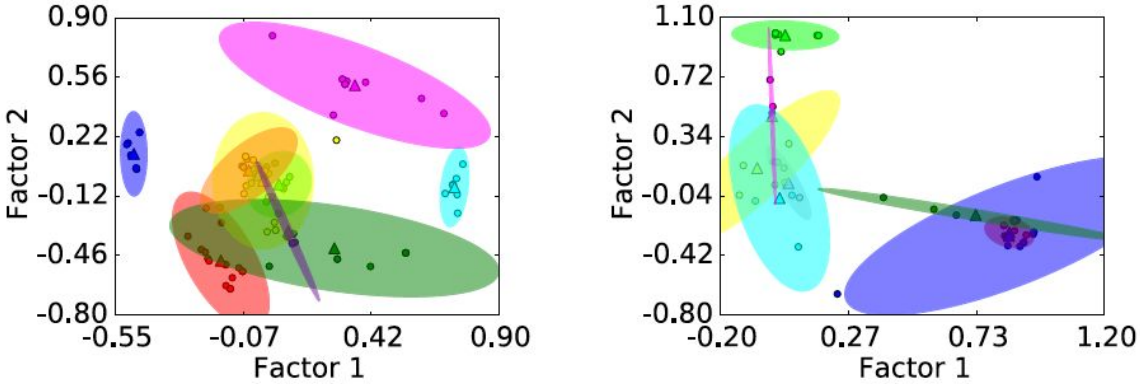
- Need
  - Smallest set of metrics
  - Most variability
  - Robust estimation
  - Full data will fit into the memory.
- Redundant metrics includes:
  - The same metric but different granularities.
  - Dependent components
- Two techniques for pruning
  - Factor analysis (FA)
  - k-means

Techniques for pruning

- FA
  - Dimensionality reduction
  - Each factor has unit variance
  - Uncorrelated with all other factors
- K-means
  - One metric per cluster
    - Drawback: requires the optimal number of clusters
    - use heuristic to automate K selection
  - Interesting finding:
    - OtterTune clusters useless metrics



https://towardsdatascience.com/a-one-stop-shop-for-principal-component-analysis-5582fb7e0a9c



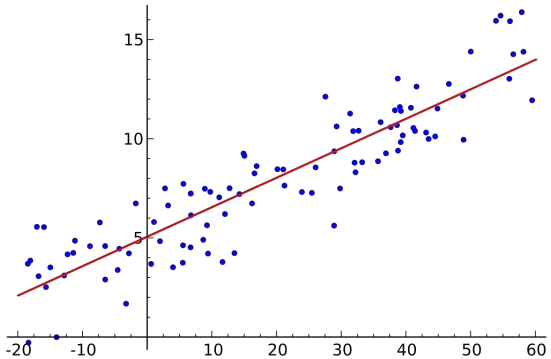
(a) MySQL (v5.6) **(b)** Postgres (v9.3)

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 (`f1`, `f2`) coordinates. The color of each metric shows its cluster membership. The triangles represent the cluster centers.

Interesting finding: OtterTune tends to group together useless metrics (e.g., SSL connection data)

Selecting knobs

- Identifies the most impactful knobs
- Find correlation to performance.
- The number of knobs selected increases dynamically.
  - Ordinary least squares (OLS)
- Instead, Lasso is a regularized version of least squares
- https://www.google.com/url?sa=i&url=https%3A%2F%2Fen.wikipedia.org%2Fwiki%2FLinear_regression&psig=AOvVaw1_phiJyscdvHstV32Hj27b&ust=1587005062164000&source=images&cd=vfe&ved=0CAIQjRxqFwoTCPCzttu06egCFQAAAAAdAAAAABAD

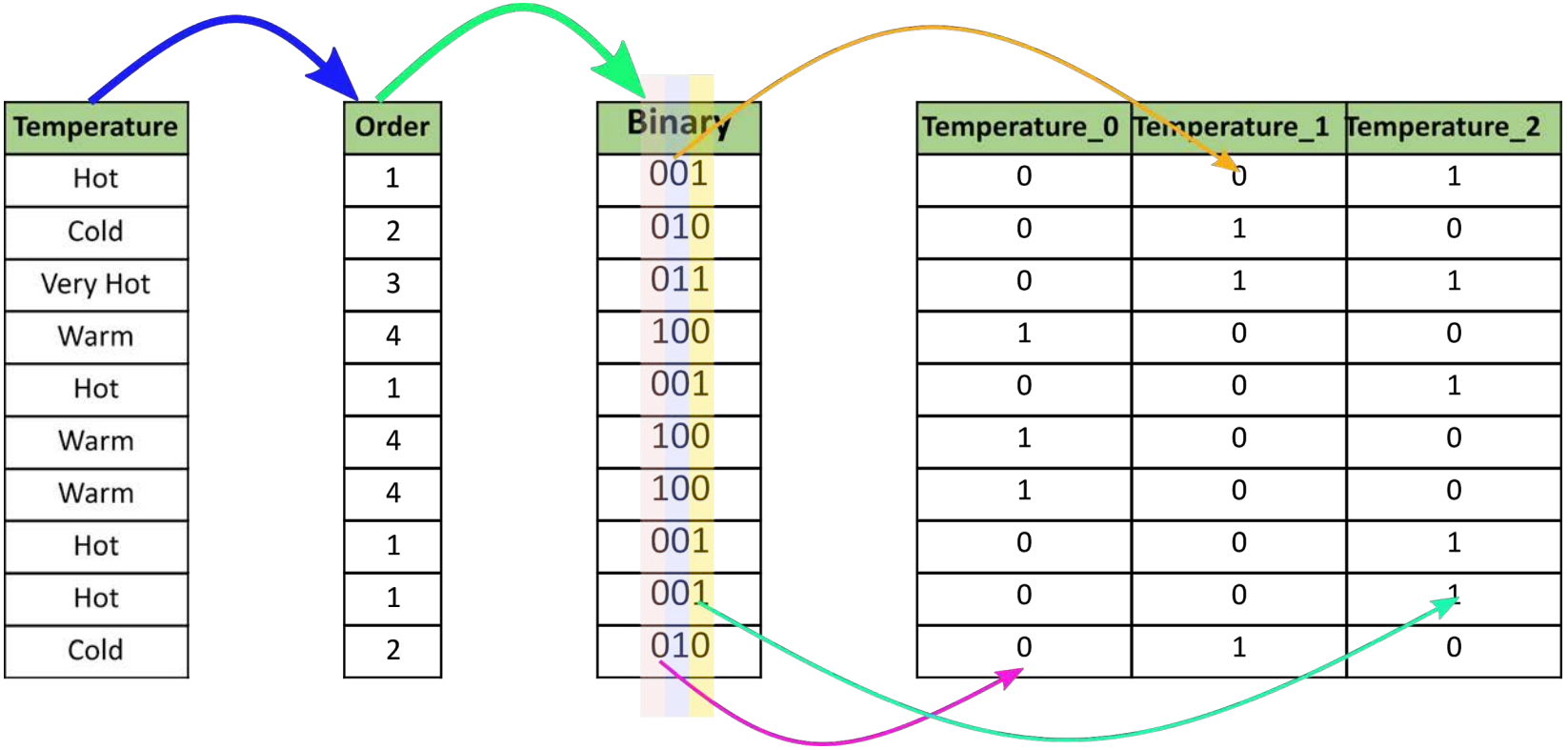


Lasso

- More interpretable, stable and computationally efficient
- L1 penalty
- Get order of importance of the knobs.
- Impact on the target metric

Data preprocessing

- Want:
  - Continuous features
  - Same order of magnitude
  - Similar variances
- Do:
  - Categorical data into binary “dummy” variable
  - Standardize the data



Temperature
Hot
Cold
Very Hot
Warm
Hot
Warm
Warm
Hot
Hot
Cold

Order
1
2
3
4
1
4
4
1
1
2

Binary
001
010
011
100
001
100
100
001
001
010

Temperature_0	Temperature_1	Temperature_2
0	0	1
0	1	0
0	1	1
1	0	0
0	0	1
1	0	0
1	0	0
0	0	1
0	0	1
0	1	0

https://www.google.com/url?sa=i&url=https%3A%2F%2Ftowardsdatascience.com%2Fall-about-categorical-variable-encoding-305f3361fd02&psig=AOvVaw24JZ1afCs1ysftqgjI3YK0&ust=1587005407431000&source=images&cd=vfe&ved=0CAIQjRxqFwoTCOiy2IC26egCFQAAAAAdAAAAABAD

Automated Tuning

- What is available now:
  - non-redundant matrics
  - most impactful knobs
  - data from previous tuning sessions.
- Executes a two-step analysis:
  - Workload mapping
  - Configuration Recommendation

Workload Mapping

- Dynamic mapping scheme
- Build a set S of N matrices
  - Rows: workloads
  - Columns: DBMS configuration
- Calculates the Euclidean distance
- Computes the score for each workload

Configuration recommendation

- Uses Gaussian Process (GP) regression
  - Good trade off between
    - exploration
    - exploitation
- Provide confidence intervals.
- Gradient descent

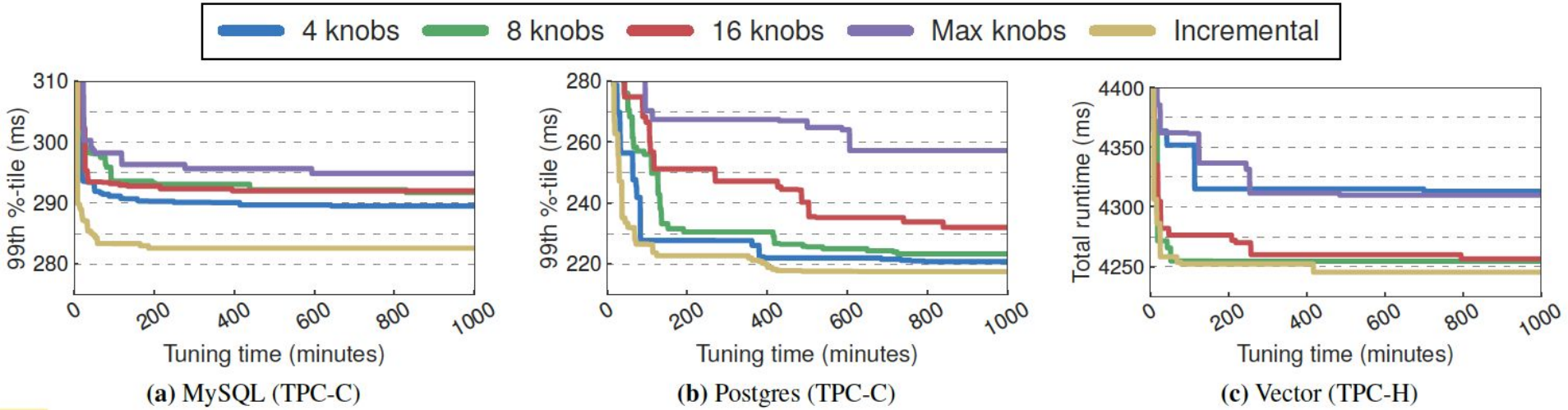
Experimental Evaluation

- Used TensorFlow and scikit-learn
- Three different DBMSs used:
  - MySQL, Postgres, and Actian Vector (OLAP)
- Deployment on Amazon EC2
- Each experiment consists of two instances
  - Controller with OLTP-Bench framework.
  - Deployed on m4.large instances
    - 4 vCPUs and 16 GB RAM
  - M3.xlarge instances
    - 4 vCPUs and 15 GB RAM
- OtterTune's tuning manager and repository
  - 20 cores and 128 GB RAM

Workloads

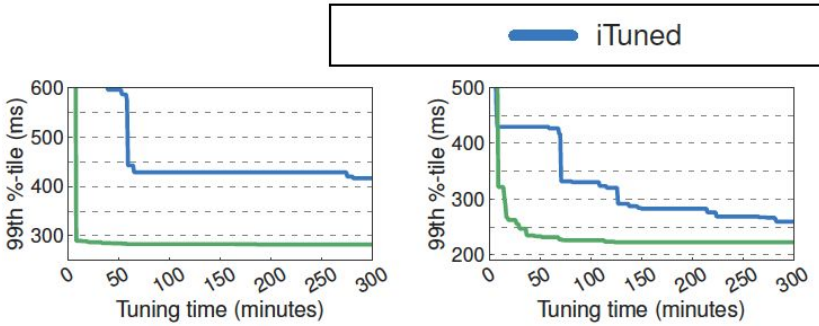
- YCSB
- TPC-C
- Wikipedia
- TPC-H (Only OLAP)
- For OLTP workloads, OtterTune uses five-minute observation periods and assign the target metric to be the 99%-tile latency

Number of Knobs



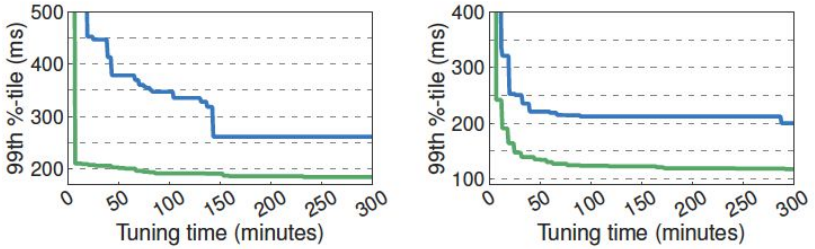
- Best for Postgres: incremental, 4, 8
- Best for Vector: incremental, 8, 16

Tuning Evaluation



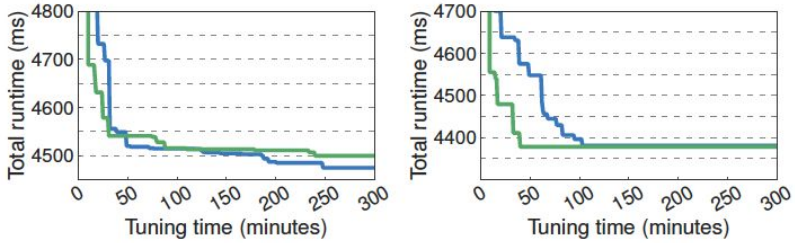
(a) MySQL (b) Postgres

Figure 6: Tuning Evaluation (TPC-C) – A comparison of the OLTP DBMSs for the TPC-C workload when using configurations generated by OtterTune and iTuned.



(a) MySQL (b) Postgres

Figure 7: Tuning Evaluation (Wikipedia) – A comparison of the OLTP DBMSs for the Wikipedia workload when using configurations generated by OtterTune and iTuned.



(a) Vector (TPC-H #1) (b) Vector (TPC-H #2)

Figure 8: Tuning Evaluation (TPC-H) – Performance measurements for Vector running two sub-sets of the TPC-H workload using configurations generated by OtterTune and iTuned.

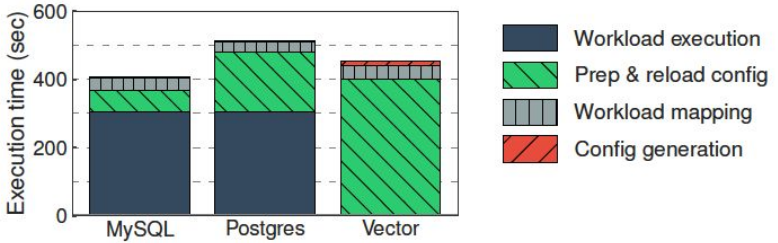


Figure 9: Execution Time Breakdown – The average amount of time that OtterTune spends in the parts of the system during an observation period.

load for the current target from its repository. This corre-

Execution Time Breakdown

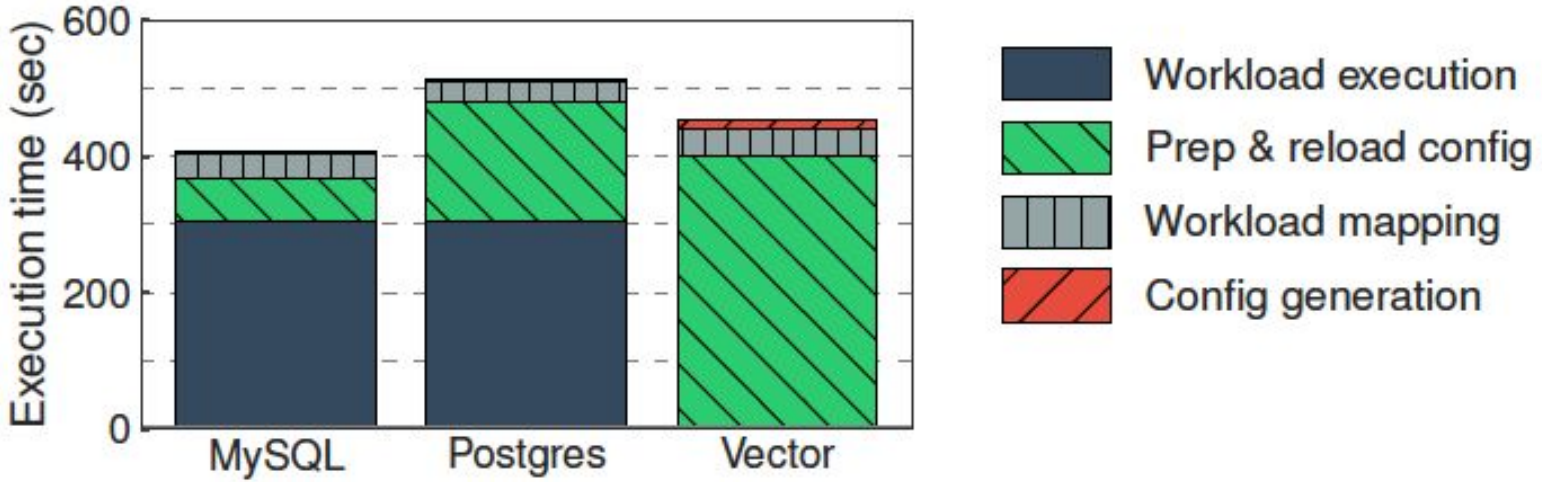
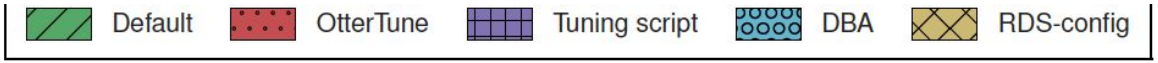
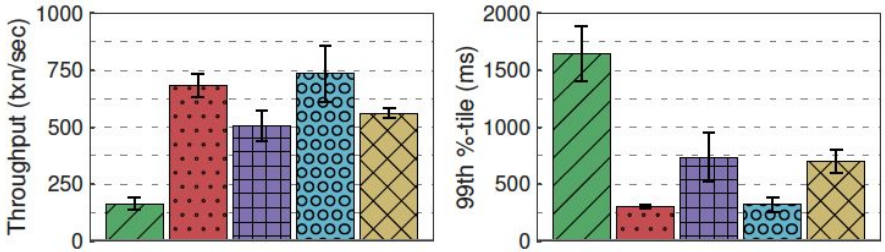


Figure 9: Execution Time Breakdown – The average amount of time that OtterTune spends in the parts of the system during an observation period.

Efficacy Comparison

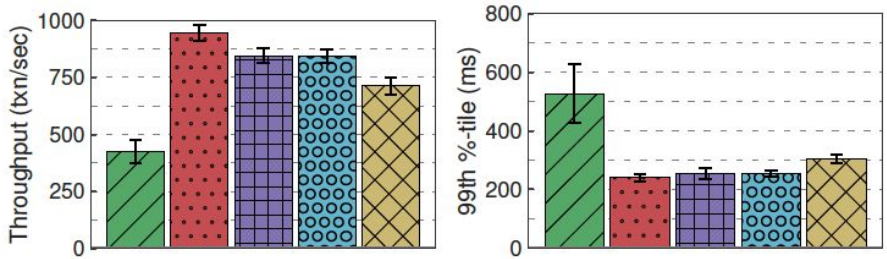
DBA: human expert
RDS-config: config given by Amazon Relational Database Service
Tuning script: tools that recommend knob configurations





(a) TPC-C (Throughput) **(b)** TPC-C (99%-tile Latency)

Figure 10: Efficacy Comparison (MySQL) – Throughput and latency measurements for the TPC-C benchmark using the (1) default configuration, (2) OtterTune configuration, (3) tuning script configuration, (4) Lithuanian DBA configuration, and (5) Amazon RDS configuration.



(a) TPC-C (Throughput) **(b)** TPC-C (99%-tile Latency)

Figure 11: Efficacy Comparison (Postgres) – Throughput and latency measurements for the TPC-C benchmark using the (1) default configuration, (2) OtterTune configuration, (3) tuning script configuration, (4) expert DBA configuration, and (5) Amazon RDS configuration.