Georgia Tech

DATA ANALYTICS USING DEEP LEARNING

AUTOMATIC DATABASE MANAGEMENT SYSTEM TUNING THROUGH LARGE-SCALE MACHINE LEARNING

SIDDHARTH BISWAL

CREATING THE NEXT®

TODAY'S PAPER

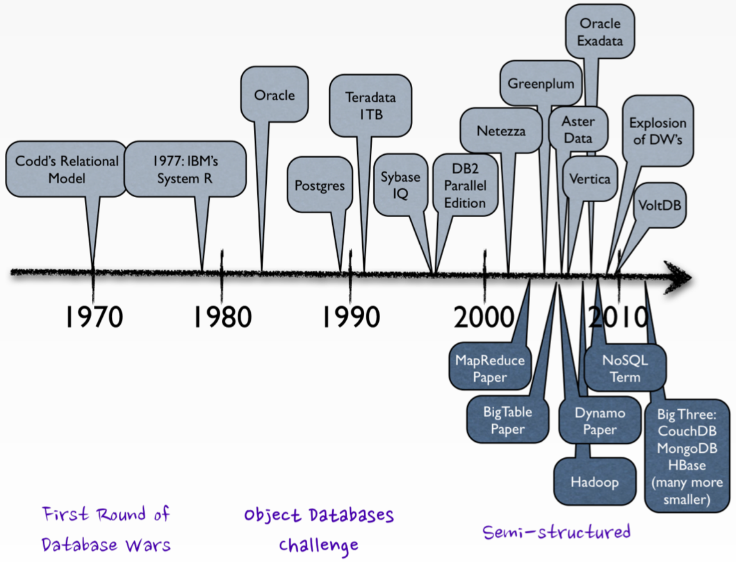
- Automatic Database Management System Tuning Through Large-scale Machine Learning
- Dana Van Aken, Andrew Pavlo, Geoffrey J. Gordon, Bohan Zhang
- Published in SIGMOD'17
- https://ottertune.cs.cmu.edu/
- https://github.com/cmu-db/ottertune

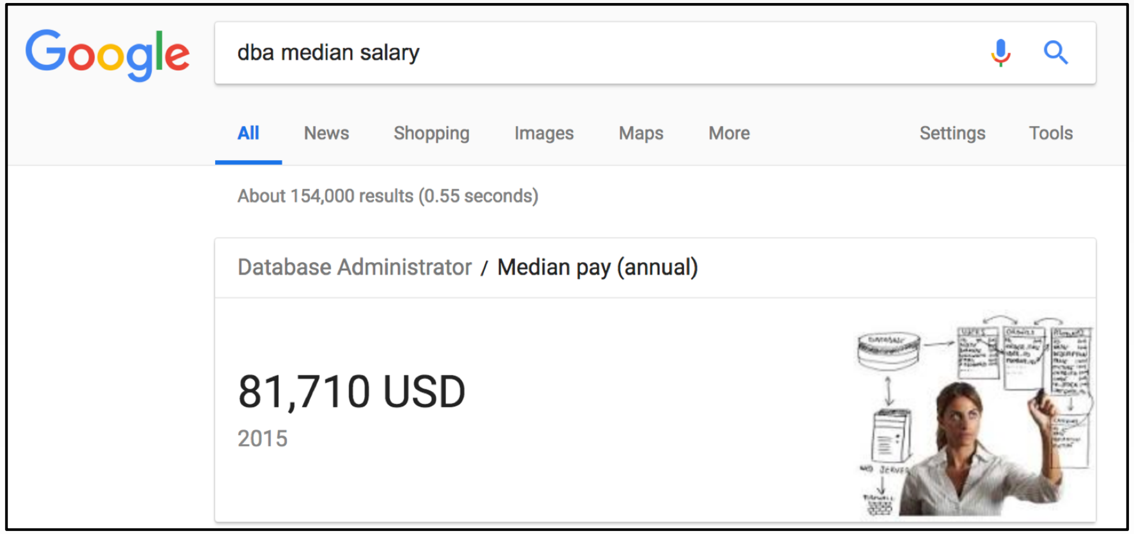
TODAY'S AGENDA

- Problem Overview
- Key Ideas
- Technical Details
- Experiments
- Discussion

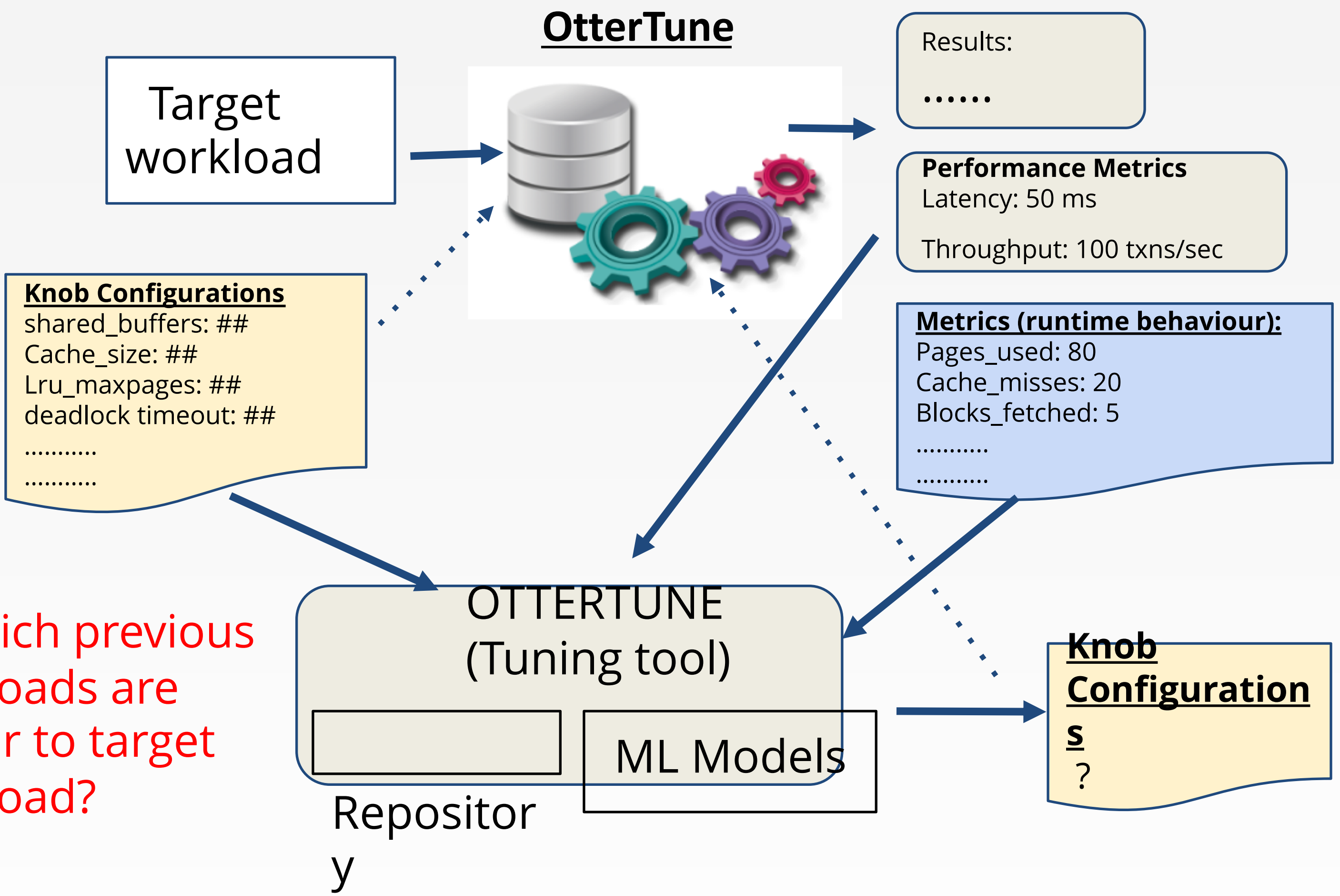
WHAT'S THE CHALLENGE?

- DBMSs **have hundreds of configuration knobs** that control everything in the system
- Knobs are **not standardized not independent ,not universal**
- Often information about the **effects of the knobs** typically comes only from **a lot of experience.**





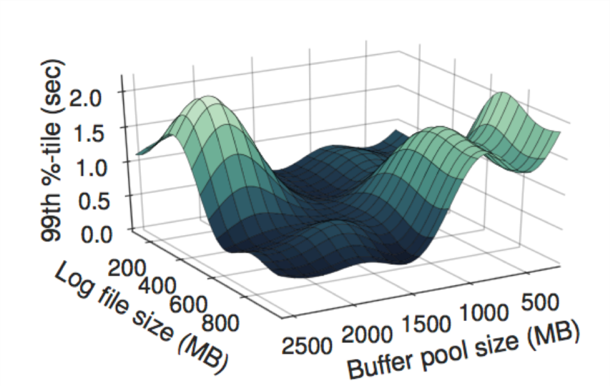
WHAT'S THE PROPOSED SOLUTION



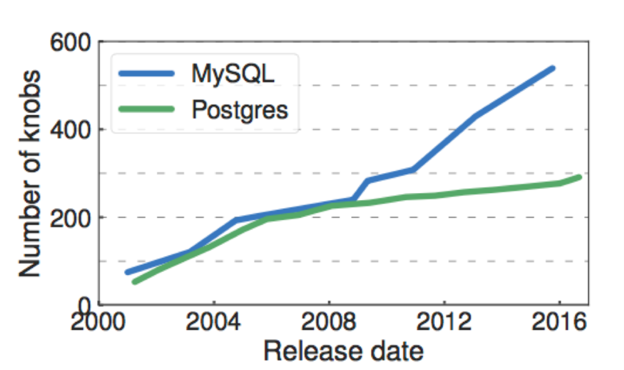
3) Which previous workloads are similar to target workload?

1) What knobs are important?
2) What values to set?

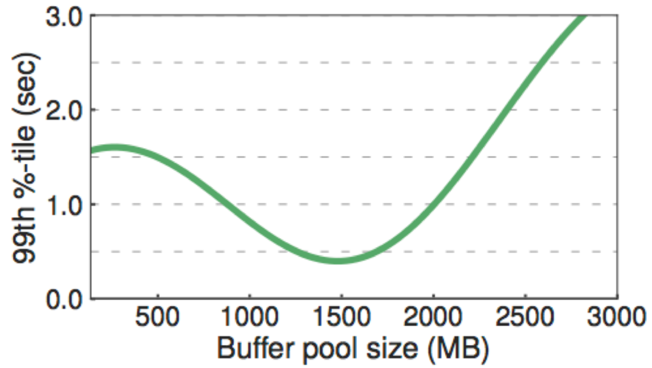
MOTIVATION



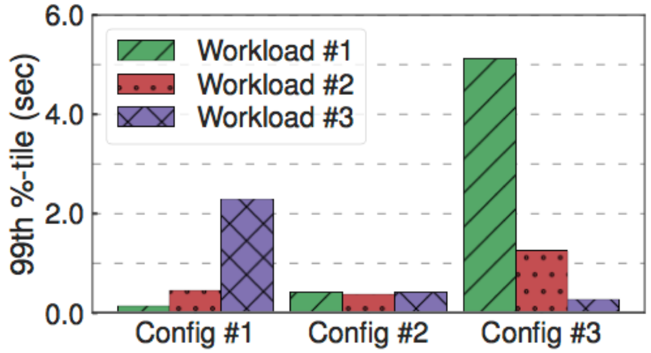
(a) Dependencies



(d) Tuning Complexity



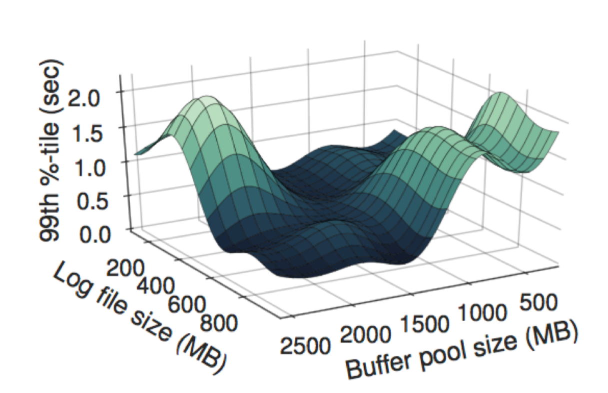
(b) Continuous Settings



(c) Non-Reusable Configurations

MOTIVATION: DEPENDENCIES

- DBMS tuning guides strongly suggest that a DBA **only change one knob at a time**
- **Slow Process**
- Different combination of Knob settings **is NP-hard**



(a) Dependencies

MOTIVATION: CONTINUOUS SETTINGS

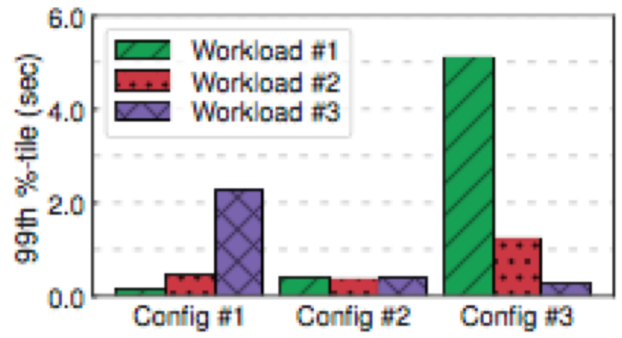
- **Many possible settings** for knobs
- Difference in Performance can be **irregular**

Example: size of the DBMS's buffer pool can be an arbitrary value from zero to the amount of DRAM on the system.

- 0.1 GB increase in this knob could be inconsequential, while in other ranges, a 0.1 GB increase could cause performance to drop precipitously as the DBMS runs out of physical memory.

MOTIVATION: NON-REUSABLE CONFIGURATIONS

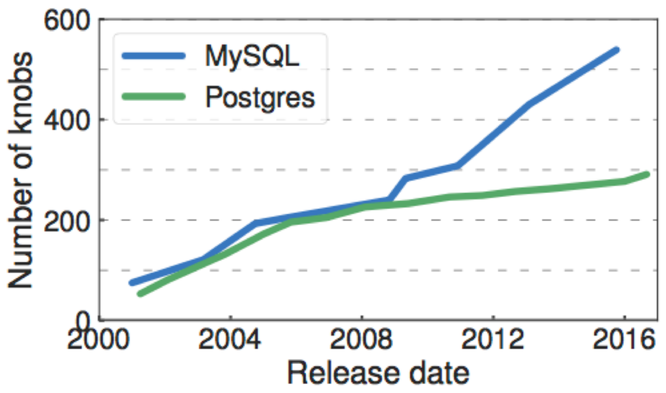
- Best configuration for one application may not be the best for another.
- 3 YCSB workloads using three MySQL knob configuration



(c) Non-Reusable Configurations

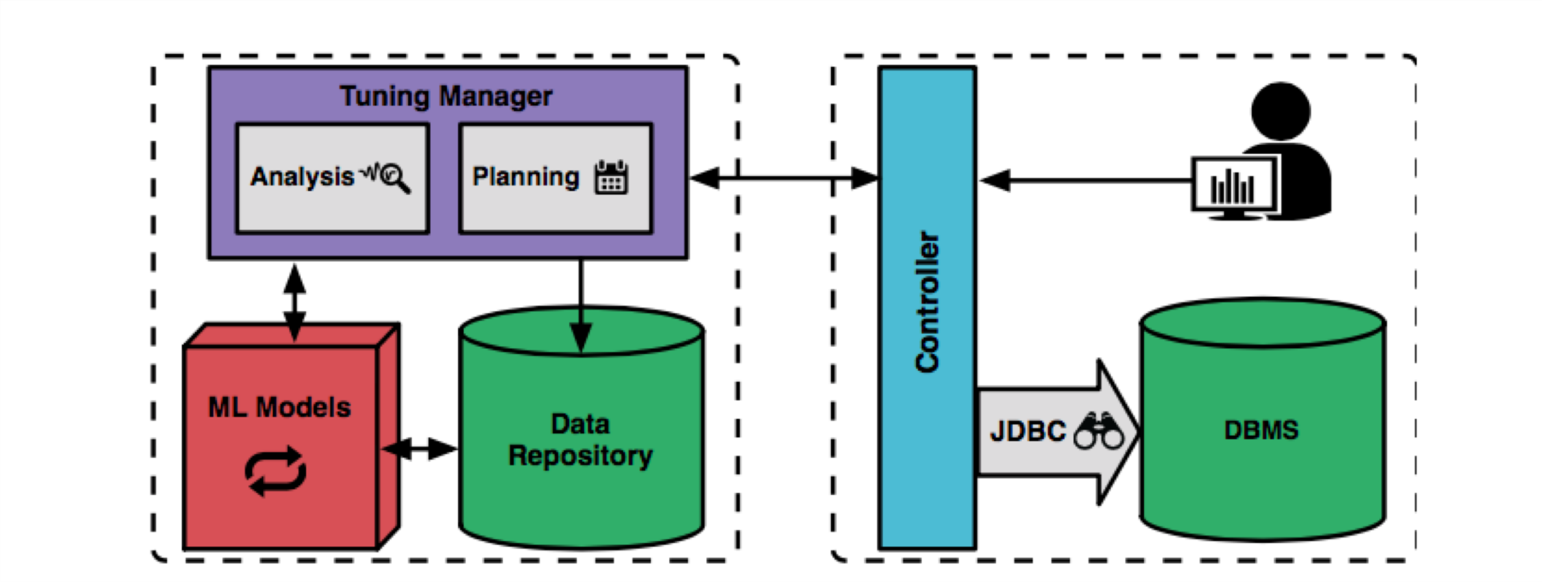
MOTIVATION: TUNING COMPLEXITY

- Number of DBMS knobs is **always increasing** as **new versions and features are released**
- **Difficult for DBAs to keep up to date** with these changes and understand how that will affect their system



(d) Tuning Complexity

SYSTEM OVERVIEW



EXAMPLE WORKFLOW

DBA tells what metric to optimize for → OtterTune connects with DBMS and collects hardware profile and knob settings → Controller starts first observation period → End of observation: controller collects additional DBMS specific settings → Tuning Manager receives results and tries to find best config

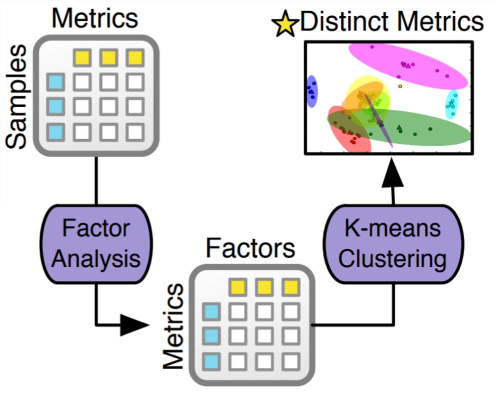
MACHINE LEARNING PIPELINE

Workload Characterization	Knobs Identification	Automatic Tuner
Minimal set of metrics to identify the workload.	What knobs are critical for a particular system?	What values to set for knobs such that performance improves?

MACHINE LEARNING PIPELINE

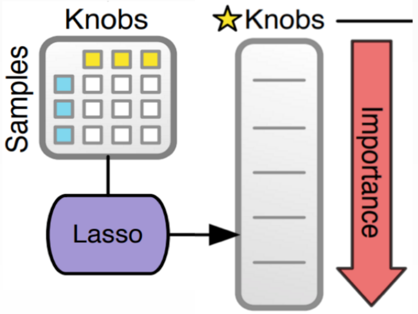
Workload Characterization

Minimal set of metrics to identify the workload.



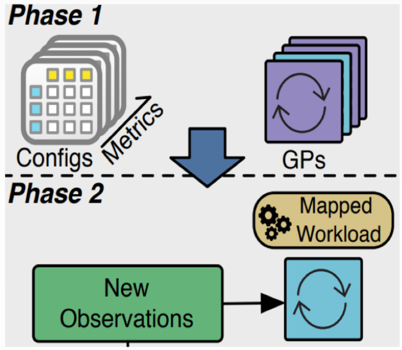
Knobs Identification

What knobs are critical for a particular system?



Automatic Tuner

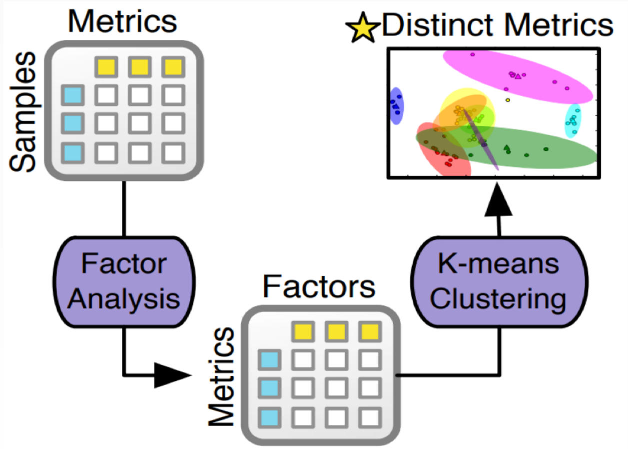
What values to set for knobs such that performance improves?



WORKLOAD CHARACTERIZATION

1. **Discover a model** that best represents distinguishing aspects of the target workload so that it can identify which previously seen workloads in the repo are similar to it.
2. **Enables OtterTune to leverage previous tuning sessions to help guide the search**

- **OtterTune characterizes a workload using the runtime statistics recorded while executing it.**
- Accurate representation of a workload because they capture more aspects of its runtime behavior



WORKLOAD CHARACTERIZATION: STATISTICS COLLECTION

- OtterTune's controller supports a **modular architecture** → enables it to perform the **appropriate operations for different DBMSs** to collect their runtime statistics.
- Controller first **resets** all of the statistics for the target DBMS
- Collects **numeric metric** that the DBMS makes available and stores it as a **key/value pair** in its repository
- Challenge:
  - **Represent metrics for sub-elements of the DBMS and database**
  - e.g MySQL, only report aggregate statistics for the entire DBMS. Other systems, however, provide separate statistics for tables or databases.
  - **OtterTune instead stores the metrics with the same name as a single sum scalar value**
  - **OtterTune currently only considers global knobs**

WORKLOAD CHARACTERIZATION: PRUNING REDUNDANT METRICS

- **Automatically remove** the superfluous metrics
- **Smallest set of metrics** that capture the variability in performance and distinguishing characteristics for different workload
- Reducing the size of this set **reduces the search space of ML algorithms**, which in **turn speeds up the entire process**

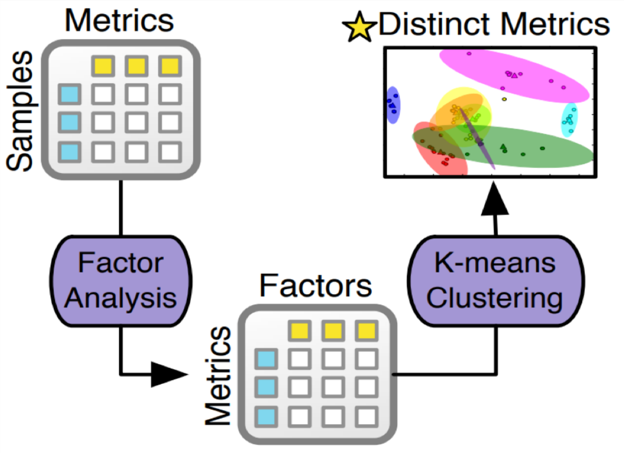
WORKLOAD CHARACTERIZATION: PRUNING REDUNDANT METRICS

- Redundant DBMS metrics occur for two reasons
  - The first are ones that provide different granularities for the exact same metric in the system
  - The other type of redundant metrics are ones that represent independent components of the DBMS but whose values are strongly correlated

WORKLOAD CHARACTERIZATION: PRUNING REDUNDANT METRICS

Phase 1 (Dimensionality Reduction)

- Find correlations among metrics using Factor Analysis
  - M1= 0.9F1 + 0.4F2 + … + 0.01F10
  - M2 =0.4F1 +0.2F2 + … + 0.02F10
  - ….
  - M100=0.6F1+ 0.3F2 + … + 0.01F10

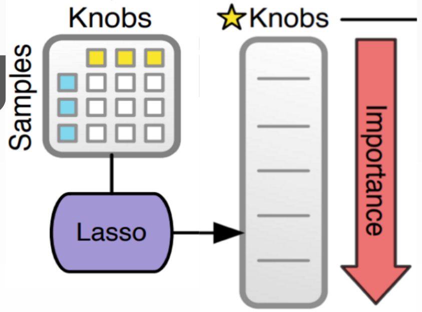


Phase 2 (Clustering)

- Apply K-Means clustering using a few factors.
- Select one representative metric from each cluster

IDENTIFYING IMPORTANT KNOBS

- **Identify knobs** which have **strongest impact** on DBA's target objective function
- **Lasso Regression** is used for feature selection
- **Tuning Manager** performs these computations in background as **new data arrives from different tuning**



FEATURE SELECTION WITH LASSO

- LASSO: Least Absolute Shrinkage Selector Operator
- **Lasso regression are some of the simple techniques to reduce model complexity and prevent over-fitting which may result from simple linear regression.**

Cost function for linear regression

$$\sum_{i=1}^{M}(y_i-\hat{y}_i)^2=\sum_{i=1}^{M}\left(y_i-\sum_{j=0}^{p}w_j\times x_{ij}\right)^2 \tag{1.2}$$

Cost function for lasso regression

$$\sum_{i=1}^{M}(y_i-\hat{y}_i)^2=\sum_{i=1}^{M}\left(y_i-\sum_{j=0}^{p}w_j\times x_{ij}\right)^2+\lambda\sum_{j=0}^{p}|w_j| \tag{1.4}$$

AUTOMATED TUNING

Available data so far

(1)the set of non-redundant metrics,

(2)the set of most impactful configuration knobs

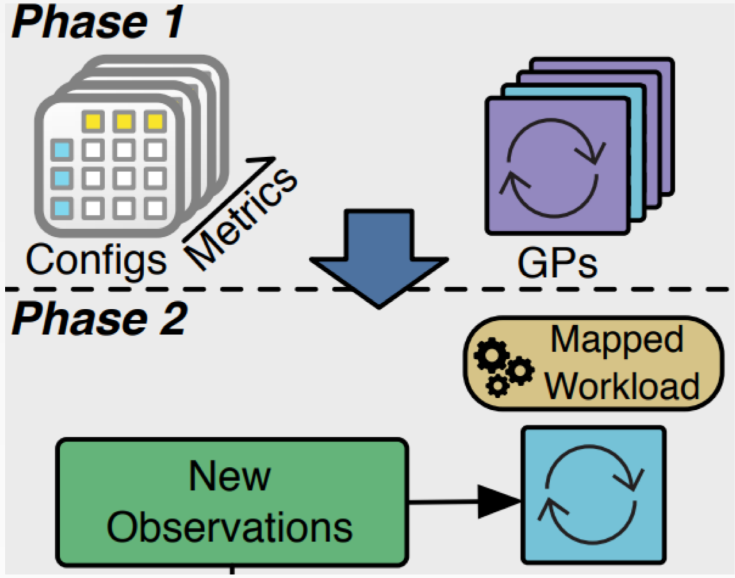
(3) the data from previous tuning sessions stored in its repository

WORKLOAD MAPPING

Recommends knobs configurations to try.

Phase 1: Workload Mapping

- Identifies workload from a previous tuning session that is most similar to the target workload.
- For measuring similarity between workloads: uses **Average Euclidean Distance**



CONFIGURATION RECOMMENDATION

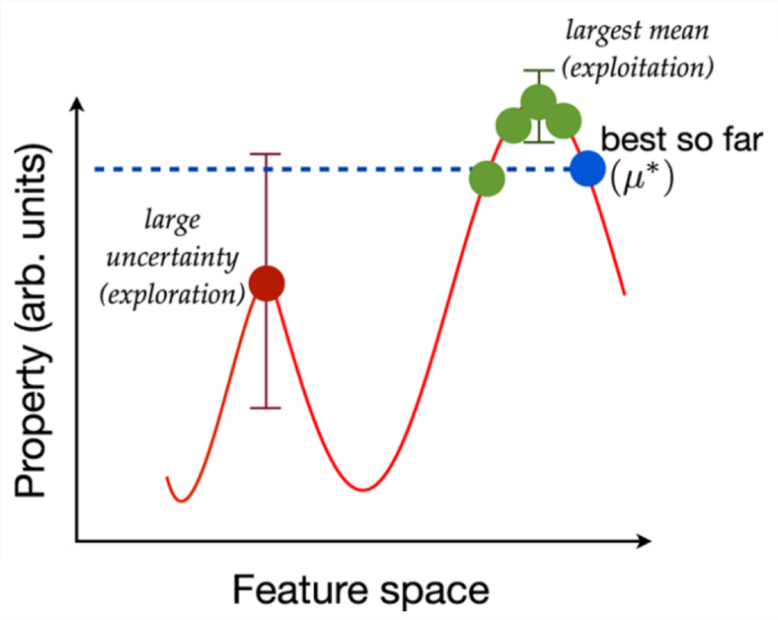
Phase 2: Configuration Recommendation

- Fits **Gaussian Process (GP) Regression** model to data from mapped and current workload

- GP provides a principled framework for **Exploration vs Exploitation**

  Exploitation: Search for configurations near to current best.

  Exploration: Search for configurations in unexplored areas.



GP DETAILED

```
>>> from sklearn.datasets import make_friedman2
>>> from sklearn.gaussian_process import GaussianProcessRegressor
>>> from sklearn.gaussian_process.kernels import DotProduct, WhiteKernel
>>> X, y = make_friedman2(n_samples=500, noise=0, random_state=0)
>>> kernel = DotProduct() + WhiteKernel()
>>> gpr = GaussianProcessRegressor(kernel=kernel,
...         random_state=0).fit(X, y)
>>> gpr.score(X, y)
0.3680...
>>> gpr.predict(X[:2,:], return_std=True)
(array([653.0..., 592.1...]), array([316.6..., 316.6...]))
```

EXPERIMENTAL SETUP

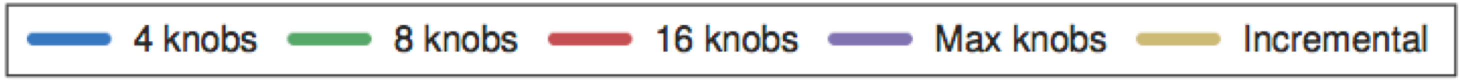
DBMSs:
MySQL (v5.6), Postgres (v9.3), Actian Vector (OLAP)

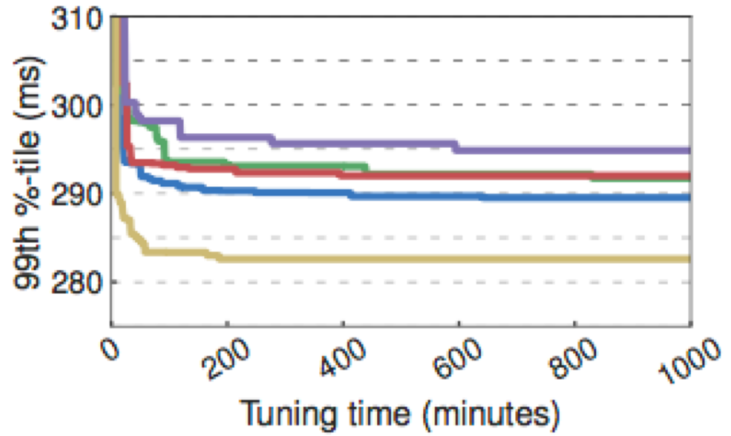
Training data collection:

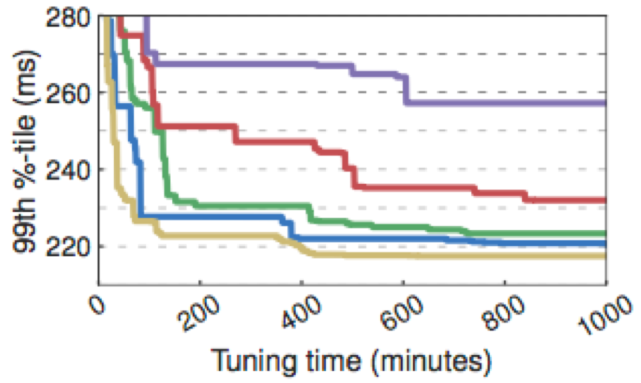
- 15 YCSB workload mixtures
- 4 sets of TPC-H queries
- Random knob configurations
- ~30k trials per DBMS

Experiments conducted on Amazon EC2

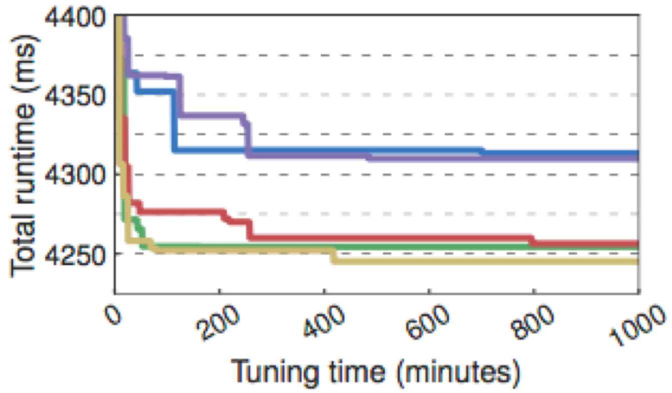
NUMBER OF KNOBS





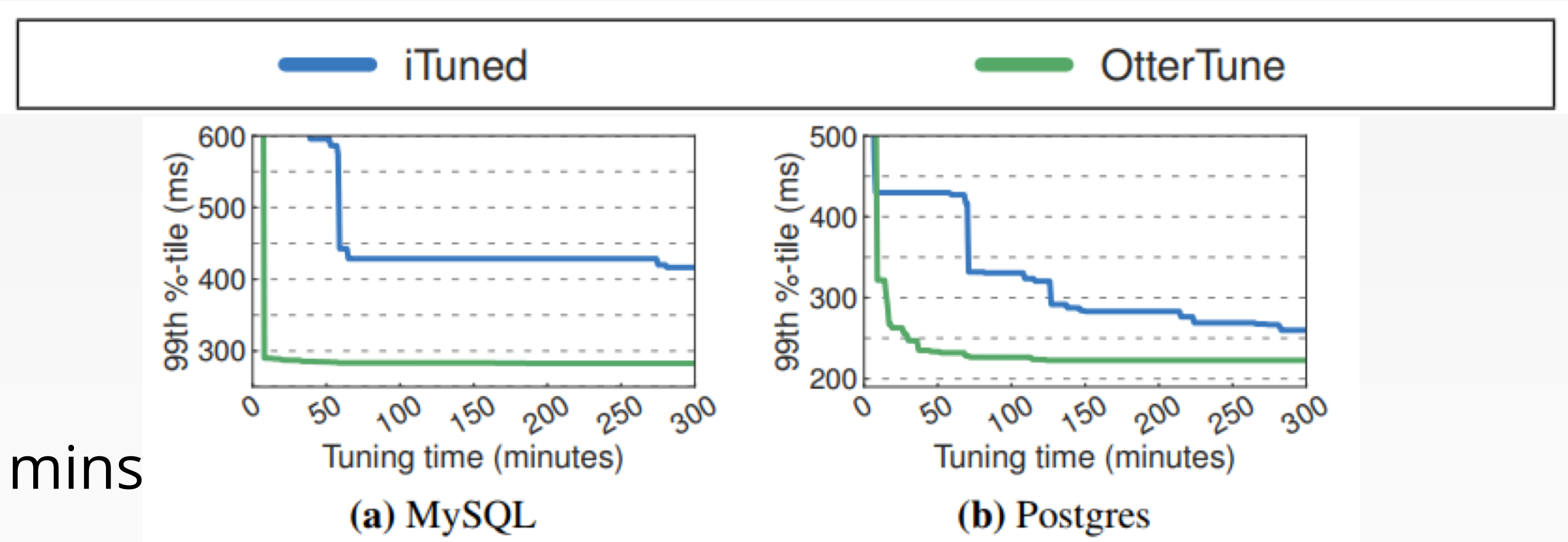


1. Incremental approach works well in MySQL
2. Incremental and fixed 4 knobs works well for Postgres
3. 8, 16, incremental works well for Actian Vector



TUNING TIME (Training data helps)

25 mins



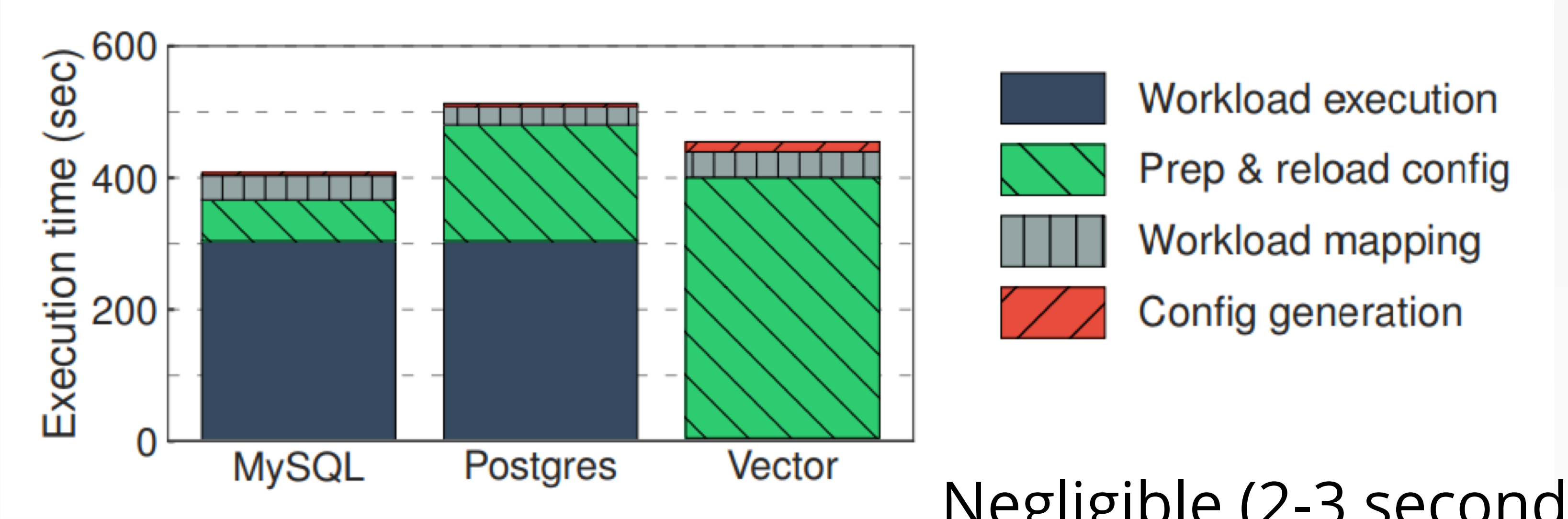
(a) MySQL **(b)** Postgres

- iTuned: Opensource tuning tool.
- Both use GP regression for config search.
- Both use incremental knob selection
- iTuned trained on only 10 different configurations vs OtterTune 30k observation period.

Execution Time Breakdown

DBMS Restart

Data reload during restart

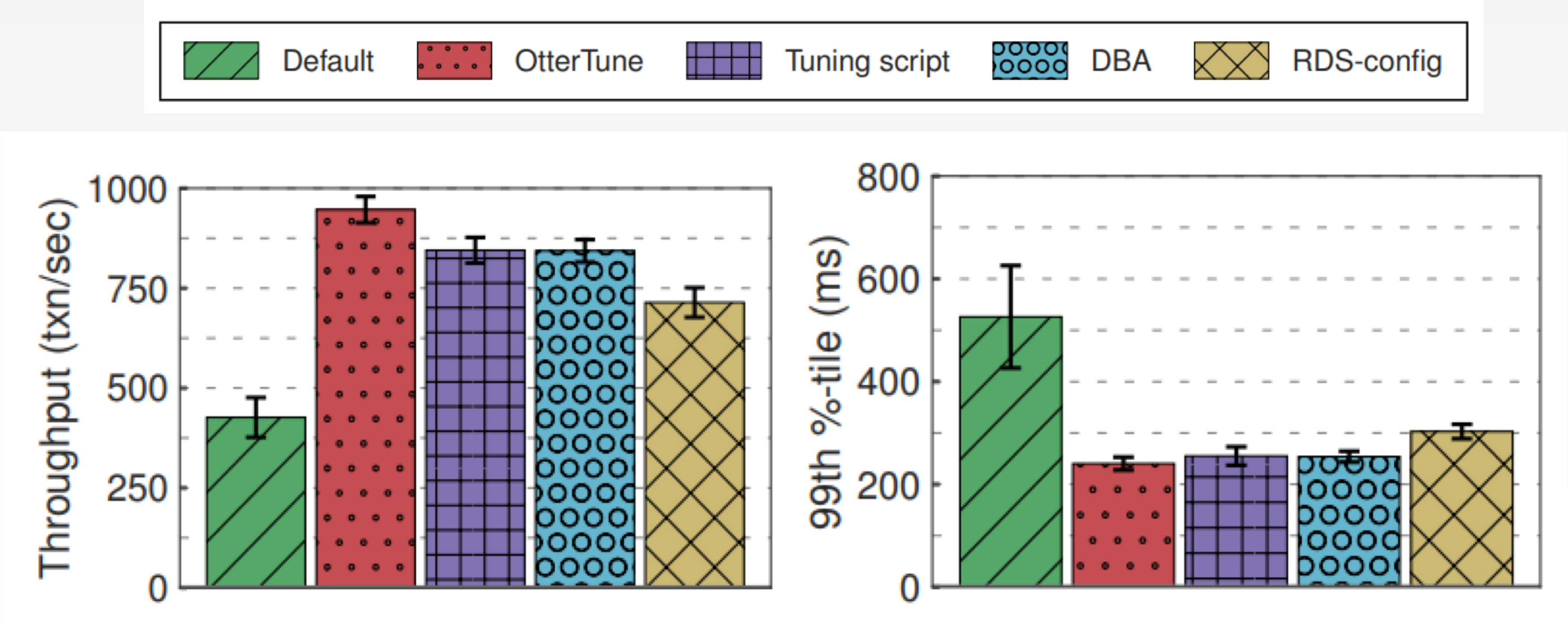


Negligible (2-3 seconds)

Observation period (5 mins)

Figure: The average amount of time that OtterTune spends in the parts of the system during an observation period.

Performance when compared with other approaches



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

CONCLUSION

Takeaways

- Generic, modular tuning system which doesn't depend on DBMS type and version.
- Automates database tuning in a short time.
- Machine learning can simplify complexity to a great extent.

Limitations

- Does not support multi-objective optimization : Tradeoffs always there. (e.g., Latency vs recovery).
- No comparison with db specific tuning tools. (PgTune for Postgres, myTune for MySQL)
- Ignores physical database design: data model, index.
- Agnostic of hardware capabilities
- Restarts, not have enough privileges, interacts via REST API (extra latency).

FUTURE DIRECTIONS

- CherryPick: Adaptively Unearthing the Best Cloud Configurations for Big Data Analytics: *Bayesian Optimization*
- Reinforcement learning based solution which tries different configuration to optimize