Automatic Database Management System Tuning Through Large-scale Machine Learning

연세대학교 컴퓨터과학과 서주연

과제명: IoT 환경을 위한 고성능 플래시 메모리 스토리지 기반 인메모리 분산 DBMS 연구개발

과제번호: 2017-0-00477

목차

목차

00 Terms

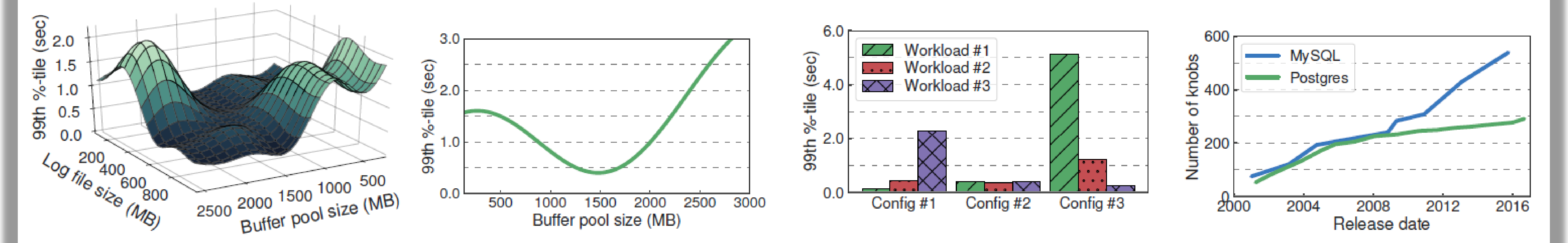
- Workloads
  - » YCSB, TPC-C, Wikipedia, TPC-H
- Knobs (configuration knobs)
  - » Ex. Innodb_buffer_pool_size, Innodb_log_file_size, …
- Configuration (knob configurations)
  - » A set of knobs
- Metrics
  - » Ex. Innodb_data_read, Inno_pages_read, latency, throughput, …

01 Introduction & Motivation

- Achieving good performance in DBMSs is non-trivial
  - » They are complex systems with many tunable options
- As databases grow in both size and complexity, optimizing a DBMS to meet the needs of an application has surpassed the abilities of humans
  - » The correct configurations of a DBMS is highly dependent on a number of factors
- Present a technique **to reuse training data** gathered from previous sessions to tune new DBMS deployments
  - » Select the most important knobs
  - » Map previously unseen database workloads to known workloads, so that we can transfer previous experience
  - » Recommend knob settings that improves a target objective
- Reduce the amount of time and resources it takes to tune a DBMS for a new application

01 Introduction & Motivation

- **Dependencies**
  - » DBA only change one knob at a time, which is not helpful because changing one knob may affect the benefits of another
- **Continuous Settings**
  - » There are many possible settings for knobs, and the differences in performance from one setting to the next could be irregular
- **Non-Reusable Configurations**
  - » The effort that a DBA spends on tuning one DBMS does not make tuning the next one any easier because the best configuration for one application may not be the best for another
- **Tuning Complexity**
  - » The number of DBMS knobs is always increasing as new versions and features are released



(a) Dependencies (b) Continuous Settings (c) Non-Reusable Configurations (d) Tuning Complexity

목차

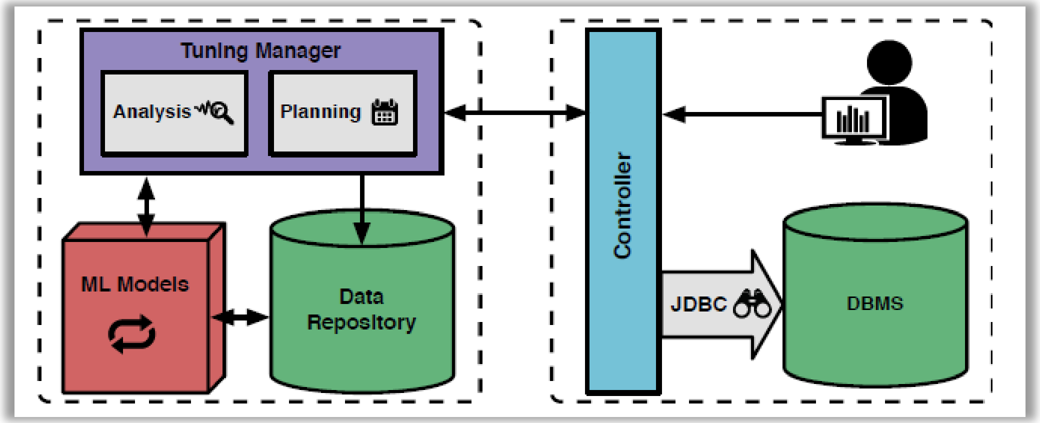
02 System Overview

└ OtterTune

- OtterTune is a tuning service that works with any DBMS
  - » Maintain a repository of data collected from previous tuning sessions, and uses this data to build models of how the DBMS responds to different knob configurations
  - » For a new application, it uses these models to guide experimentation and recommend optimal settings

- The client-side ***Controller***
  - » Connect to the DBMS and collect runtime information about the performance of the system

- OtterTune's ***Tuning Manager***
  - » Receive the information about the performance of system and store it in its repository
  - » Build models that are used to select an optimal configuration for the DBMS



02 System Overview

└ Example

- The DBA tells OtterTune what metric to optimize when selecting a configuration (e.g., latency, throughput)

- The OtterTune *controller* connects to the target DBMS and collects its hardware profile and current knob configuration

- The *controller* starts the first *observation period*
  - » Measure **DBMS-independent external** metrics chosen by the DBA (e.g., latency)
  - » Collect additional **DBMS-specific internal** metrics (e.g., counters for pages read to or written from disk)

- Ottertune's *tuning manager* receives the result of a new observation period from the controller
  - » Store that information in its repository
  - » Compute the next configuration that the controller should install on the target DBMS

- Determine what kind of configuration the system will recommend
  - » Map the target workload to a workload for the same DBMS and hardware profile that it has seen (and tuned) in the past
  - » After finding the best match using the data that it has collected, it recommends a knob configuration that is specifically designed to improve the target objective for the current workload, DBMS, and hardware

02 System Overview

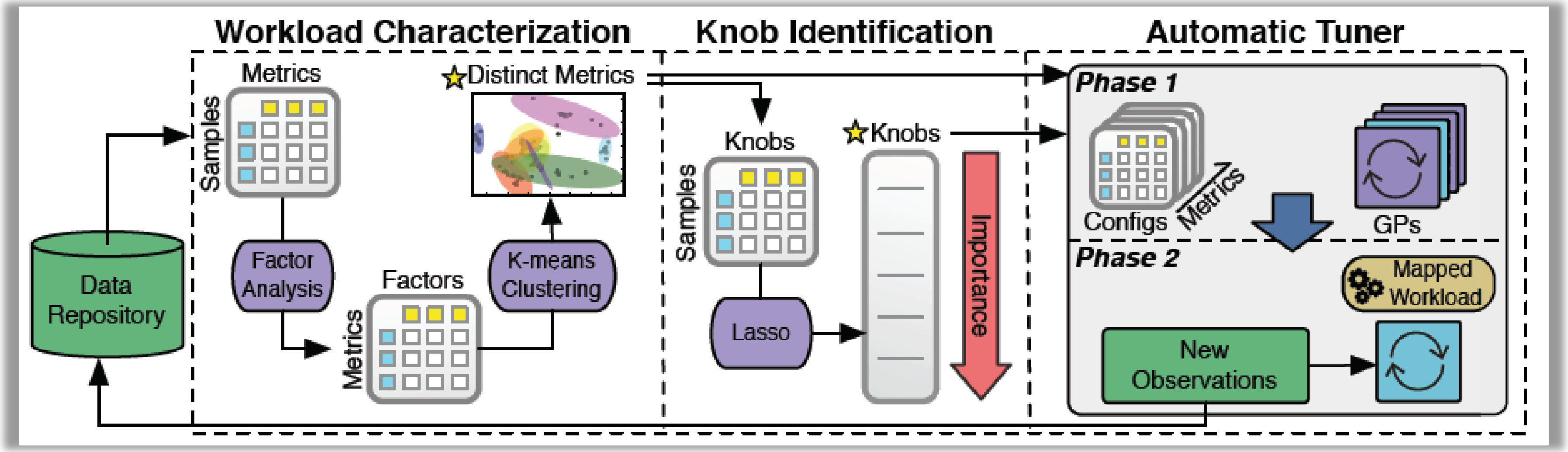
└ OtterTune Machine Learning Pipeline

- All previous observations reside in its repository

- **Workload Characterization**
  - » Identify the most distinguishing DBMS metrics

- **Knob Identification**
  - » Generate a ranked list of the most important knobs

- **Automatic Tuner**
  - » Map the target DBMS's workload to a previously seen workload and generate better configurations



목차

03 Workload Characterization

- To discover a model that best represents the distinguishing aspects of the target workload
  - » Can identify which previously seen workloads in the repository are similar to it
  - » Enable OtterTune to leverage the information that it has collected from previous tuning sessions

- Use the **DBMS's internal runtime metrics** to characterize how a workload behaves
  - » Provide a more accurate representation of a workload because they capture more aspects of its runtime behavior
  - » Be directly affected by the knobs' settings

- **Statistics Collection**
  - » At the beginning of each observation period, the controller resets all of the statistics for the target DBMS
  - » Retrieve the new statistics data at the end of the period

- **Pruning Redundant Metrics**
  - » Remove the superfluous metrics so that OtterTune only has to consider the smallest set of metrics that capture the variability in performance and distinguishing characteristics for different workloads

03 Workload Characterization

└ Pruning Redundant Metrics

- **Factor Analysis (FA)**
  - » Transform the high dimensional DBMS metric data into lower dimensional data
  - » Reduce a set of real-valued variables to a smaller set of *factors* that capture the correlation patterns of the original variables
  - » Input *X* – row: metrics, column: knob configurations
  - » Output *U* – row: metrics, column: factors

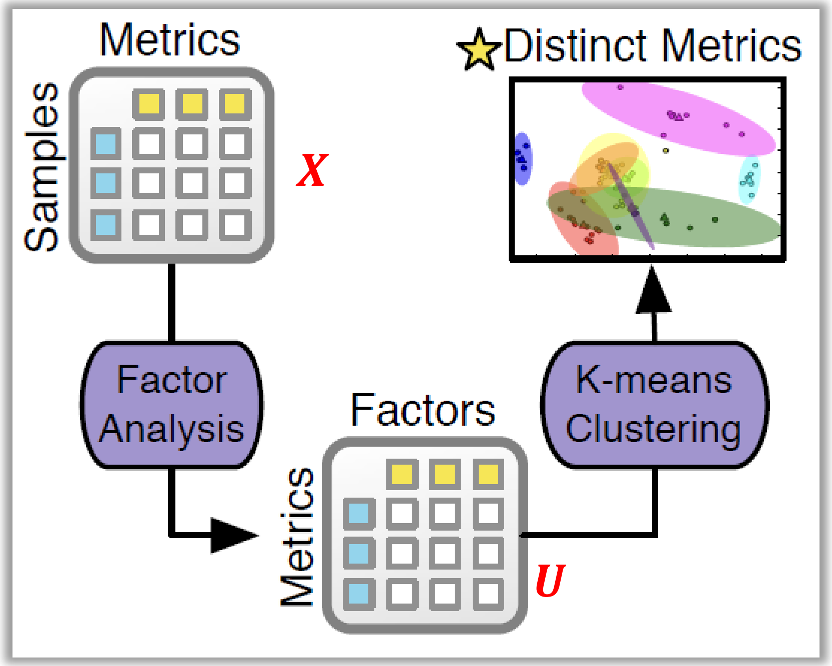
Find correlations among metrics using Factor Analysis

$$M_1 = 0.9F_1 + 0.4F_2 + \cdots + 0.01F_{10}$$

$$M_2 = 0.4F_1 + 0.2F_2 + \cdots + 0.02F_{10}$$

$$\vdots$$

$$M_{100} = 0.6F_1 + 0.3F_2 + \cdots + 0.01F_{10}$$



03 Workload Characterization

└ Pruning Redundant Metrics

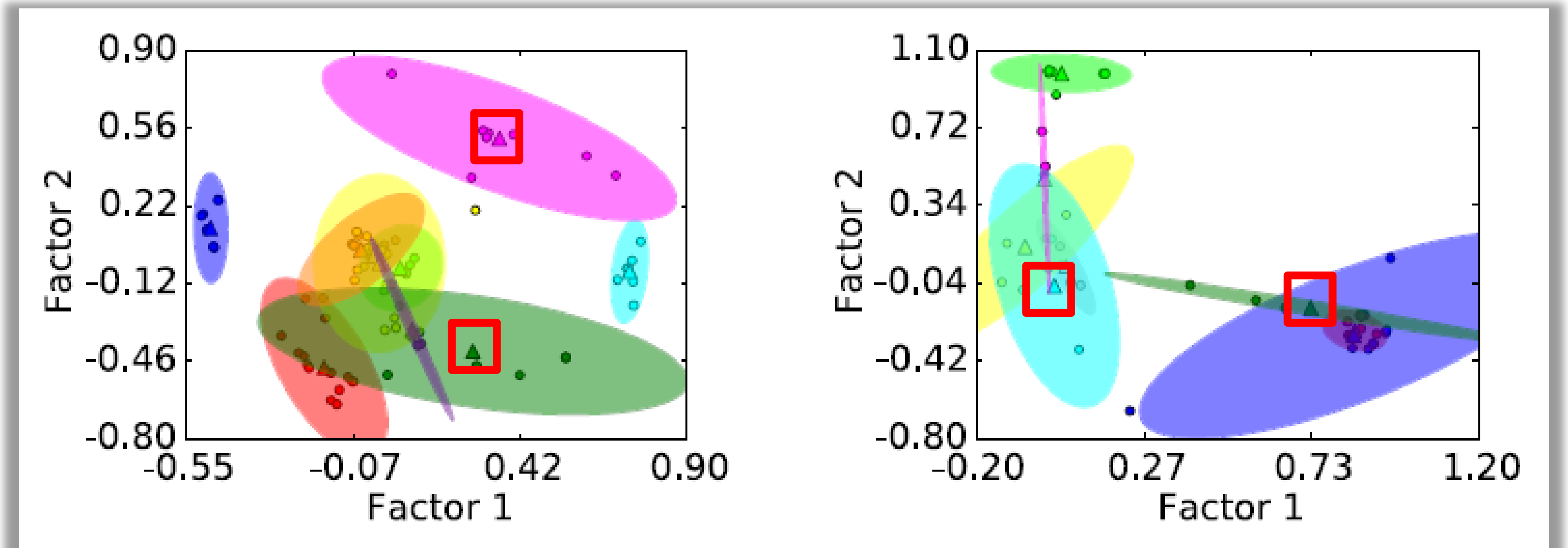
- **K-means**
  - » Grouping DBMS metrics based on how similar they are to each other as identified by Factor Analysis
  - » Choosing a single metric for each cluster as the one closest to the cluster center
  - » The clusters correspond to distinct aspects of a DBMS's performance

$$M_1 = 0.9F_1 + 0.4F_2 \cancel{+ \cdots + 0.01F_{10}}$$

$$M_2 = 0.4F_1 + 0.2F_2 \cancel{+ \cdots + 0.02F_{10}}$$

$$\vdots$$

$$M_{100} = 0.6F_1 + 0.3F_2 \cancel{+ \cdots + 0.01F_{10}}$$



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

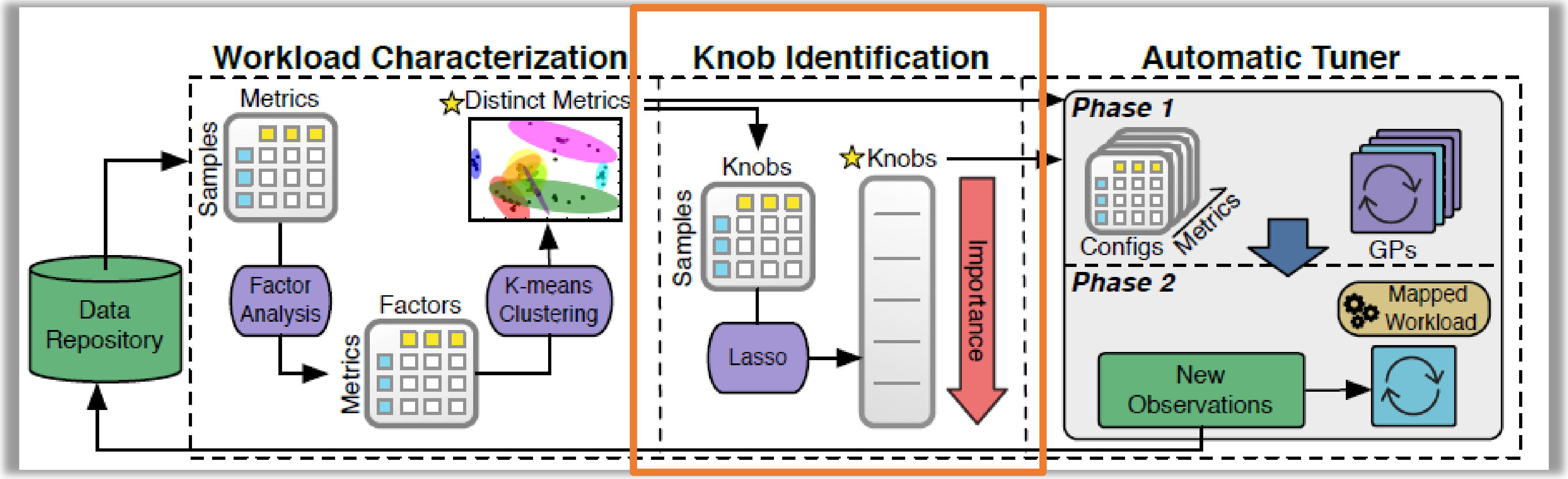
목차

04 Identifying Important Knobs

- After pruning the redundant metrics, OtterTune next identifies which knobs have the strongest impact on the DBA's target objective function

- DBMSs can have hundreds of knobs, but only a subset actually affect the DBMS's performance
  - » Ex. Reducing the amount of memory allocated for the DBMS's buffer pool is likely to degrade the system's overall latency

- To expose the knobs that have the strongest correlation to the system's overall performance, OtterTune uses a popular feature selection technique for linear regression, called Lasso



04 Identifying Important Knobs

└ Feature Selection with Lasso

- Linear Regression is a statistical method used to determine the strength of the relationship between one or more dependent variables y and each of the independent variables X
  - » X: DBMS's konbs   y: the metrics that OtterTune collects during an observation period from the DBMS

- OtterTune employs a regularized version of least squares, known as *Lasso*, that reduces the effect of irrelevant variables in linear regression models by penalizing models with large weights

- *Lasso* linear regression : $MSE + L_1 = \frac{1}{\mathrm{n}} \sum_{i=1}^{n} (y_i - \hat{y}_i)^2 + \lambda \sum_{j=1}^{m} |w_j|$
  - » Weights close non-zero mean more relevant features
  - » Others(zeros) are discarded

04 Identifying Important Knobs

└ Feature Selection with Lasso

- OtterTune uses *the Lasso path algorithm* to determine the order of importance of the DBMS's konbs
  - » Starts with a high penalty setting where all weights are zero
  - » Thus no features are selected in the regression model
  - » Then decreases the penalty in small increments, recomputes the regression
  - » And tracks what features are added back to the model at each step

- Finally, OtterTune uses the order in which the knobs first appear in the regression to determine how much of an impact they have on target metric → the first selected knob is the most important

- Before OtterTune computes this model, it executes two preprocessing steps to normalize the knobs data → provides higher quality results
  - » when features are (1) continuous, (2) have approximately the same order of magnitude, and (3) have similar variances

04 Identifying Important Knobs

└ Dependencies & Incremental Knob Selection

- Dependencies
  - » Many of a DBMS's knobs are non-independent, which means changing one may affect another
  - » Including polynomial features in the regression captures dependencies between knobs

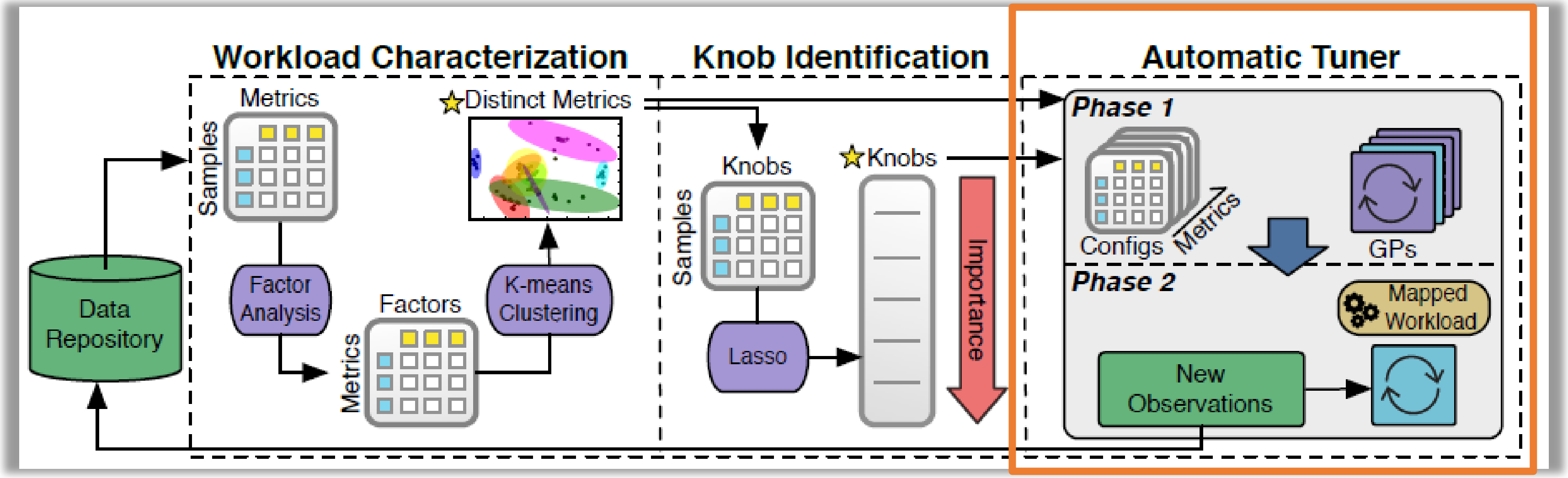
- Incremental Knob Selection
  - » OtterTune now has a ranked list of all knobs
  - » Must decide how many of these knobs to use in its recommendation
  - » Using too many → increases optimization time significantly because the size of the configuration space grow exponentially
  - » Using too few → prevents finding the best configuration
  - » To resolve this, uses an incremental approach → dynamically increases the number of knobs

목차

05 Automated Tuning

- Now at this point OtterTune has (1) the set of non-redundant metrics, (2)the set of most impactful configuration konbs, and (3) the data from previous tuning sessions stored in its repository

- Two-step analysis
  1. The system identifies which workload from a previous tuning session is most emblematic of the target workload
  2. Choose a configuration that is explicitly selected to maximize the target objective

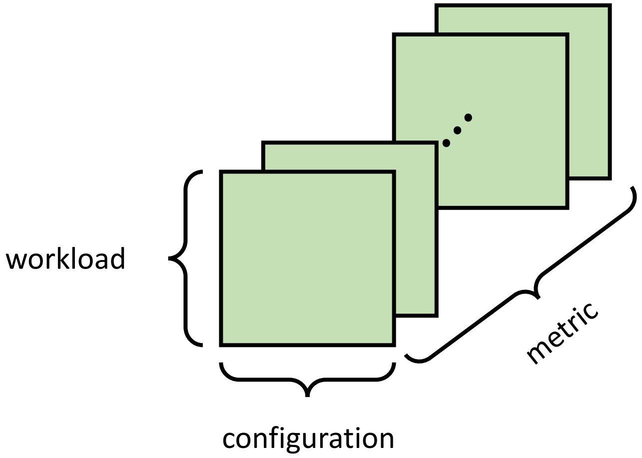


05 Automated Tuning

└ Step #1 – Workload Mapping

- The goal of this step is to match the target DBMS's workload with the most similar workload in its repository based on the performance measurements for the selected group of metrics

- Build a set S of N matrices
  - » $S = \{X_0, X_1, \ldots, X_{N-1}\}$, identical row and column labels
  - » Row in X_m: workload in repository
  - » Column in X_m: DBMS configuration
  - » $X_{m,i,j}$ $m: metric, i: workload, j: configuration$

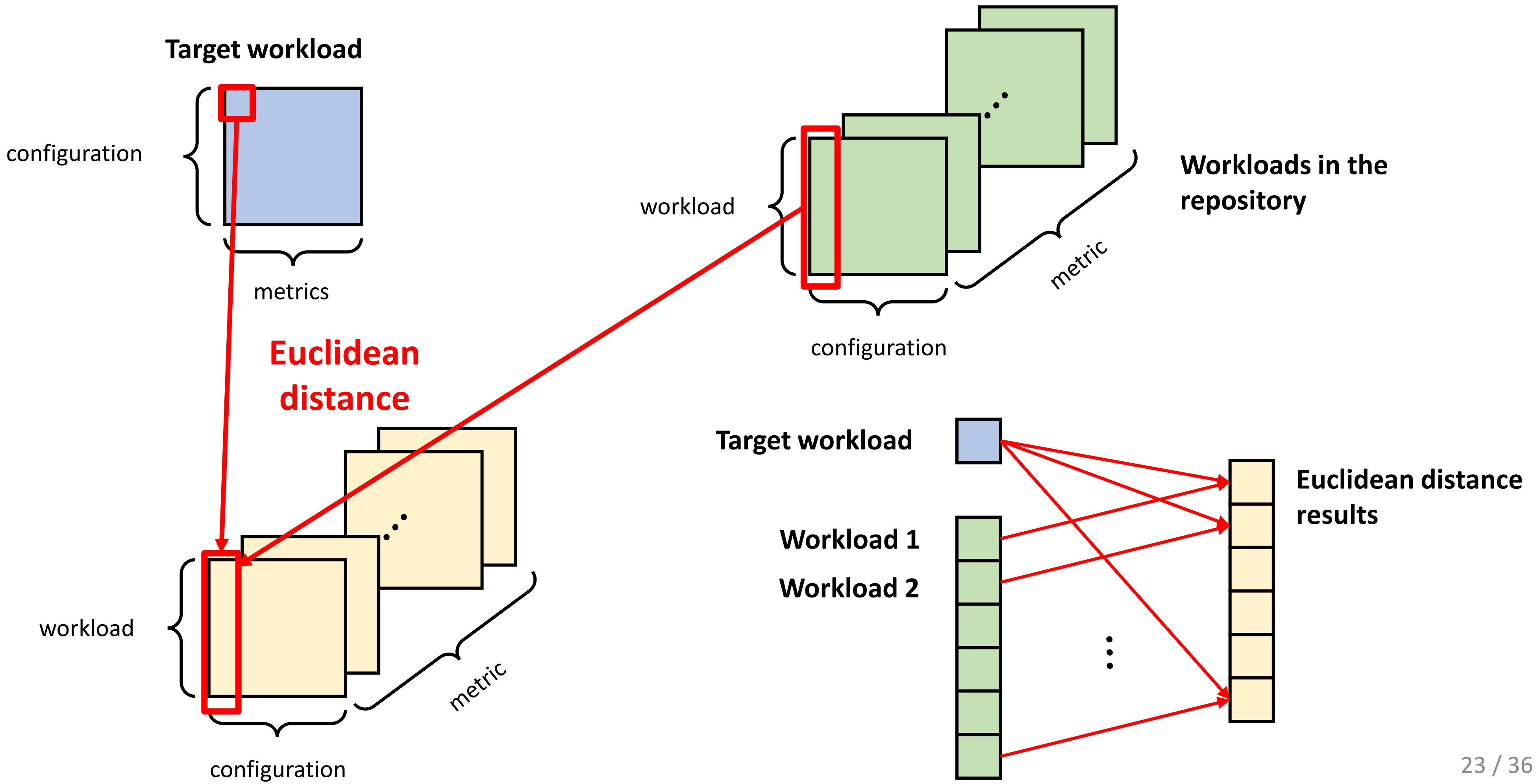


- Calculate the Euclidean distance between the vector of measurements for the target workload and the corresponding vector for each workload i in the matrix X_m

05 Automated Tuning

└ Step #1 – Workload Mapping

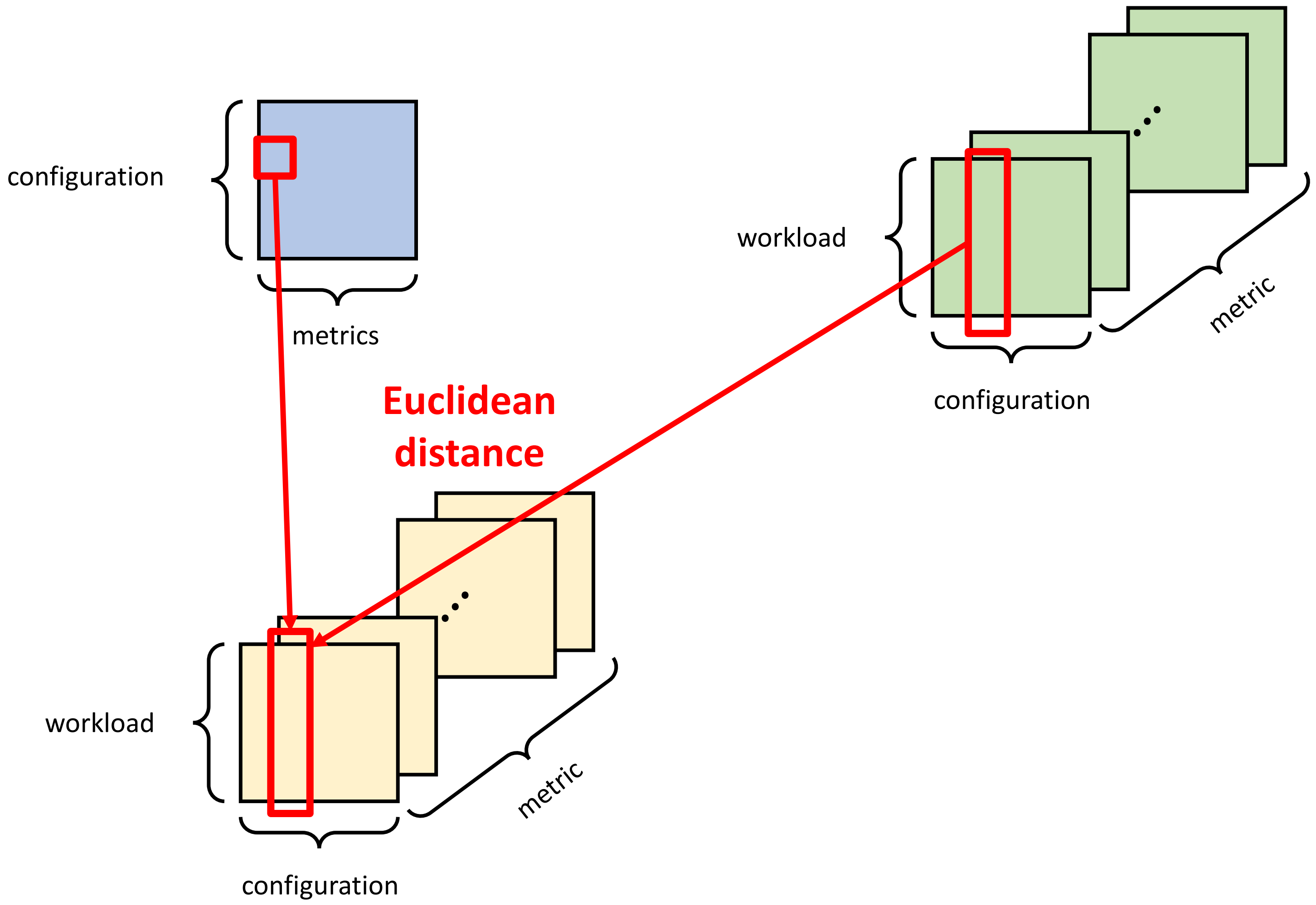
- Calculate the Euclidean distance between the vector of measurements for the target workload and the corresponding vector for each workload i in the matrix X_m



05 Automated Tuning

└ Step #1 – Workload Mapping

- Calculate the Euclidean distance between the vector of measurements for the target workload and the corresponding vector for each workload i in the matrix X_m

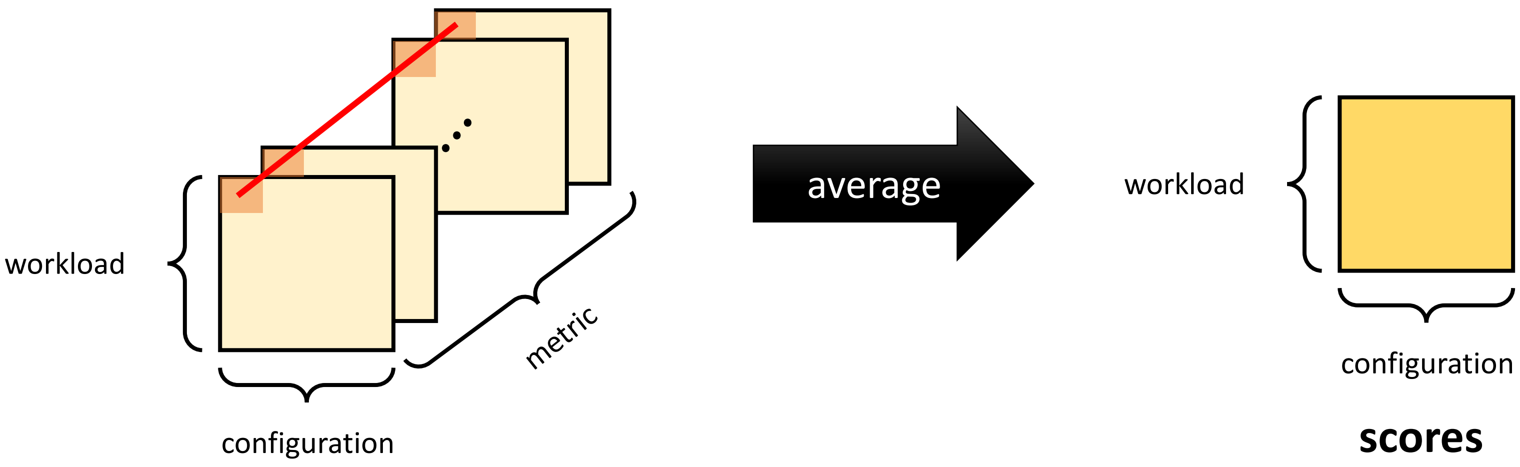


05 Automated Tuning

└ Step #1 – Workload Mapping

- Computes a "score" for each workload i by taking the average of these distances over all metrics m

- The algorithm then chooses the workload with the lowest score as the one that is most similar to the target workload for that observation period



05 Automated Tuning

└ Step #2 – Configuration Recommendation

- In this step, OtterTune uses *Gaussian Process(GP) regression* to recommend configurations that it believes will improve the target metric
  - » Not estimate variable, $y = ax + b$
  - » Estimate Gaussian distribution
- Starts the recommendation step by reusing the data from the workload that it selected previously to train a GP model
- Add a ridge term to the covariance → mapped workload is not exactly identical to the unknown one
- Also add a smaller ridge term for each configuration that OtterTune selects

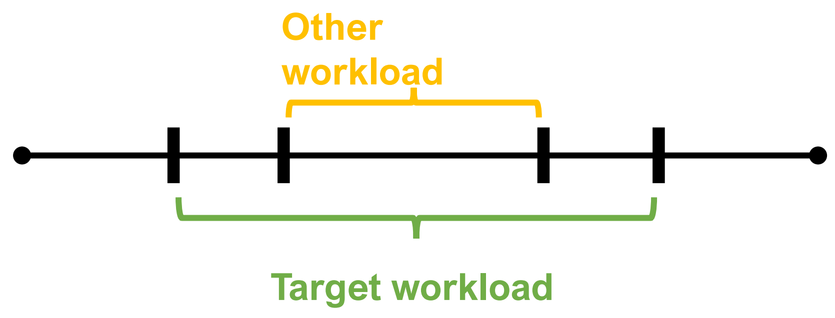
05 Automated Tuning

└ Step #2 – Configuration Recommendation

- In this step, OtterTune tries to find a better configuration than the best configuration that it has seen thus far in this session

- Do this by either
  1) Exploration
     - searching an unknown region in its GP (workloads for which it has little to no data for)
     - This helps OtterTune look for configurations where knobs are set to values that are beyond the minimum or maximum values that it has tried in the past (ex. Where the upper limit depend on the underlying hardware)

  2) Exploitation
     - Selecting a configuration that is near the best configuration in its GP
     - This is where OtterTune has found a good configuration and It tries slight modifications to the knobs to see whether it can further improve the performance



목차

06 Evaluation & Conclusion

└ Experiment

- **Using three different DBMS**
  - » MySQL(v5.6)
  - » Postgres(v9.3)
  - » Actian Vector(v4.2)

- **Workloads**
  - » YCSB
  - » TPC-C
  - » Wikipedia
  - » TPC-H

06 Evaluation & Conclusion

└ Number of Knobs

- Analysis of OtterTune's performance when optimizing different numbers of knobs
  - » The goal is to show that OtterTune can properly identify the optimal number of knobs
- TPC-C benchmark for the OLTP DBMSs – MySQL and Postgres
- TPC-H benchmark for the OLAP DBMS - Vector
- Two type of knob count settings – (1) fixed (2) incremental

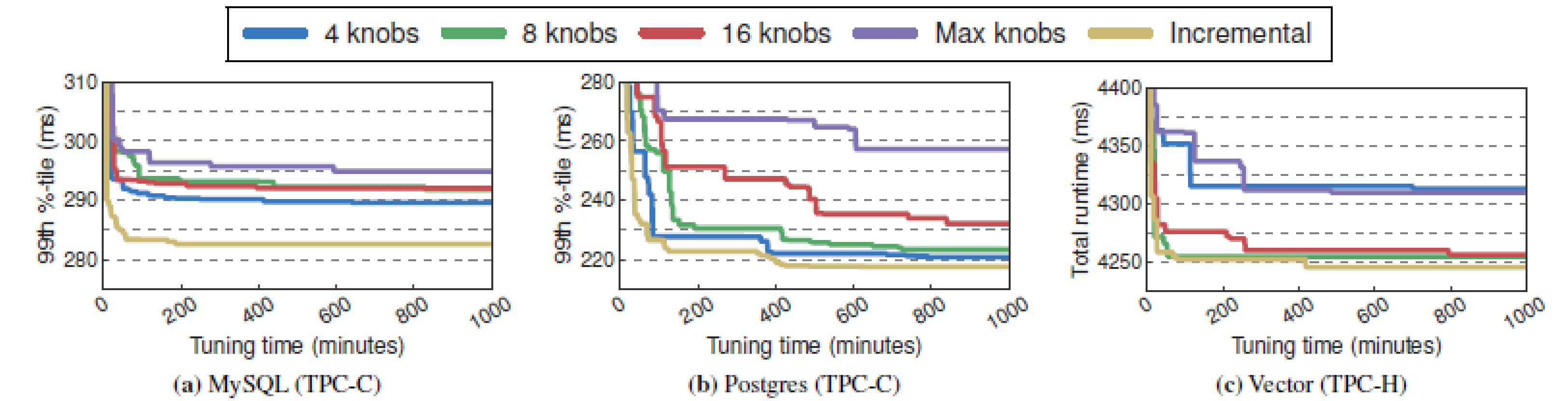


Figure 5: Number of Knobs – The performance of the DBMSs for TPC-C and TPC-H during the tuning session using different configurations generated by OtterTune that only configure a certain number of knobs.

06 Evaluation & Conclusion

└ Number of Knobs

- **MySQL (TPC-C)**
  - » Best is incremental method
  - » The next is using 4 knobs
    - – DBMS's buffer pool and log file size
    - – The method used to flush data to storage
  - » The larger knob count setting
    - – Ability to control additional thread policies
    - – The number of pages prefetched into the buffer pool
  - » Including these less impactful knobs increases the amount of noise in the model, making it harder to find the values of knobs

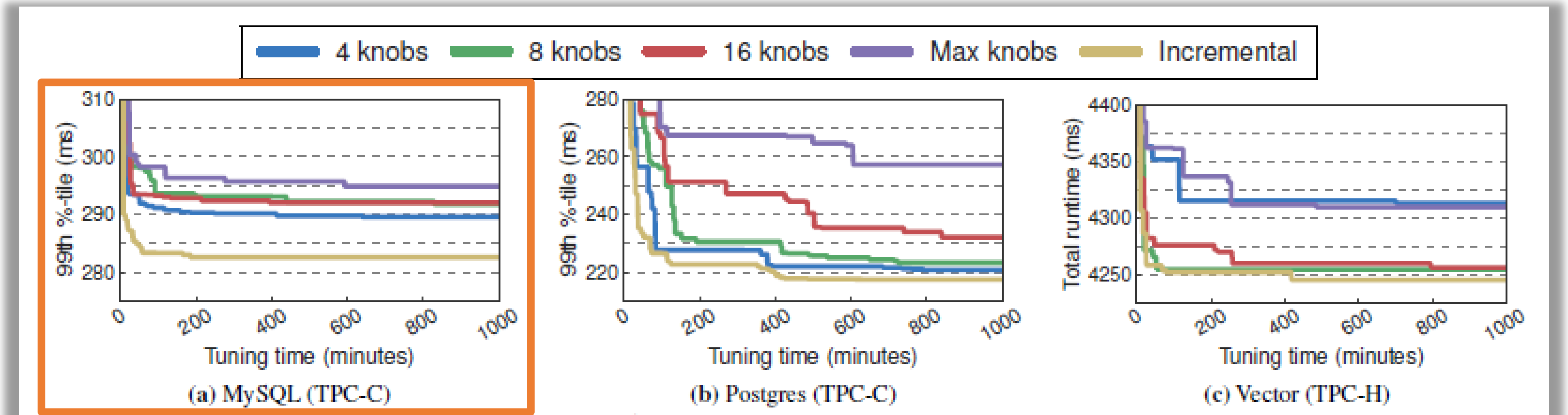


Figure 5: Number of Knobs – The performance of the DBMSs for TPC-C and TPC-H during the tuning session using different configurations generated by OtterTune that only configure a certain number of knobs.

06 Evaluation & Conclusion

└ Number of Knobs

- **Postgres (TPC-C)**
  - » Best is incremental method
  - » The next is using 4 knobs
    - – DBMS's buffer pool size
    - – The knob influences which query plans are selected by the optimizer
  - » Including these less impactful knobs increases the amount of noise in the model, making it harder to find the values of knobs

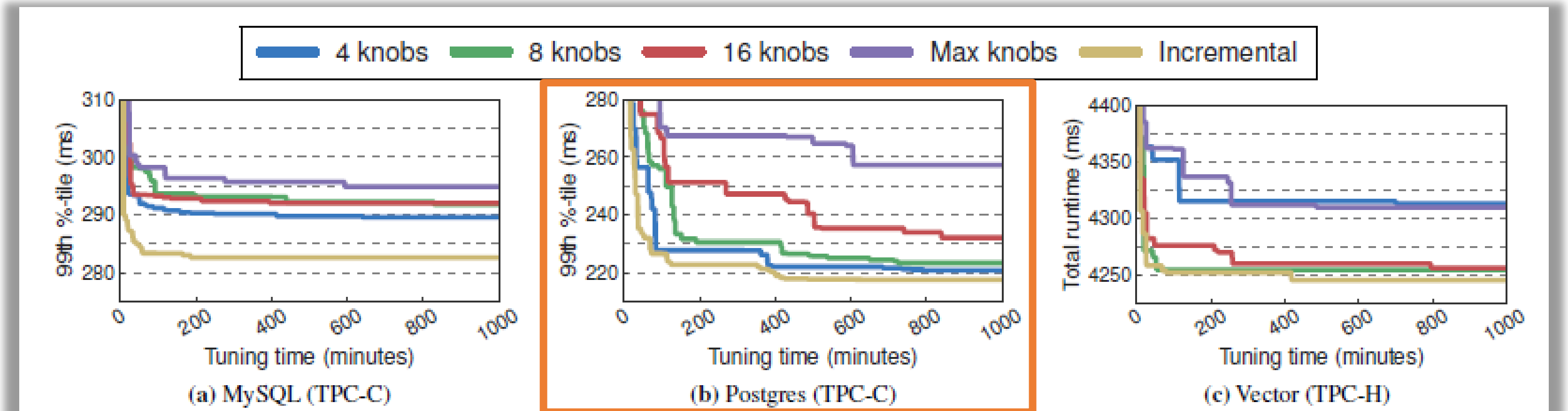


Figure 5: Number of Knobs – The performance of the DBMSs for TPC-C and TPC-H during the tuning session using different configurations generated by OtterTune that only configure a certain number of knobs.

06 Evaluation & Conclusion

└ Number of Knobs

- **Vector (TPC-H)**
  - » Best is 8 knobs, 16 knobs and incremental
  - » This is because some of Vector's more impactful knobs are present in the 8 knobs setting but not in the four knob
  - » 4 knobs include
    - – The level of parallelism for query execution and the buffer pool's size
    - – Prefetching option and the SIMD capabilities of the DBMS
  - » Including these less impactful knobs increases the amount of noise in the model, making it harder to find the values of knobs

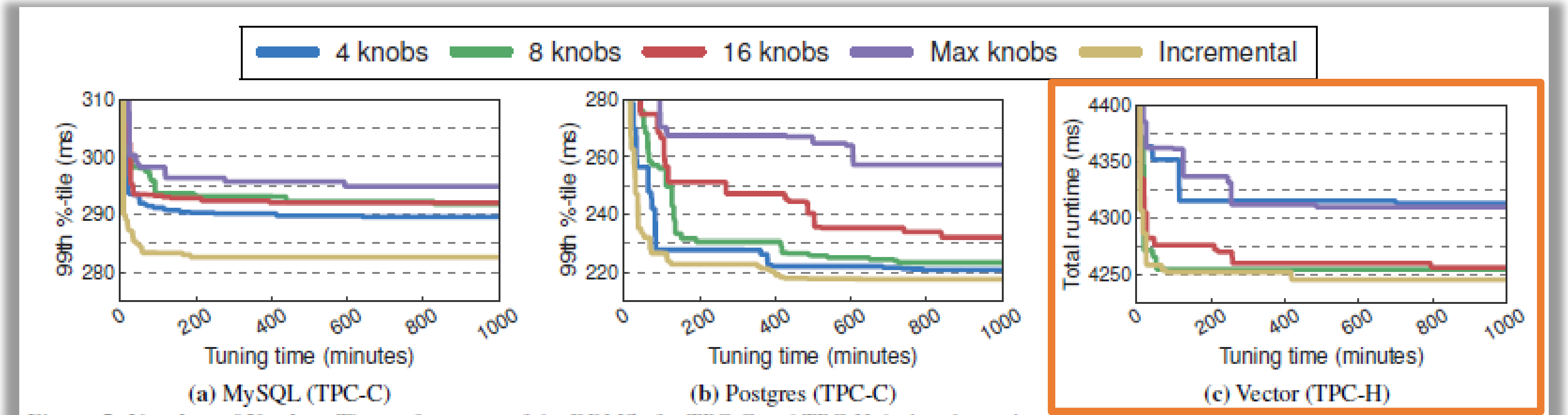
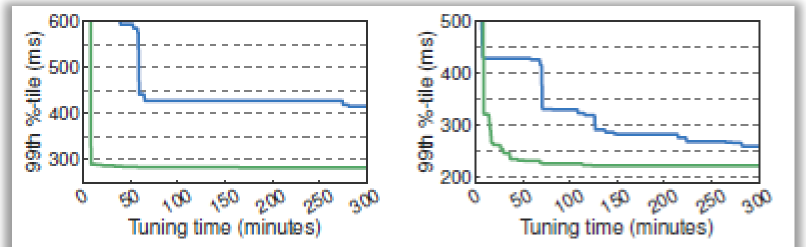


Figure 5: Number of Knobs – The performance of the DBMSs for TPC-C and TPC-H during the tuning session using different configurations generated by OtterTune that only configure a certain number of knobs.

06 Evaluation & Conclusion

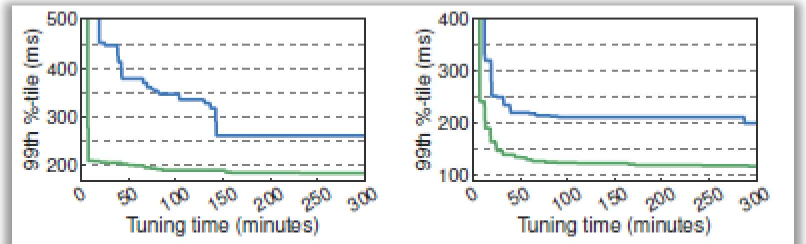
└ Tuning Evaluation

- Demonstrate how learning from previous tuning sessions improves OtterTune's ability to find a good DBMS knob configuration
- Compared another tuning tool, iTuned
  - » Use Gaussian process
  - » But does not train its GP models using data collected from previous tuning session
- OtterTuned uses incremental knobs



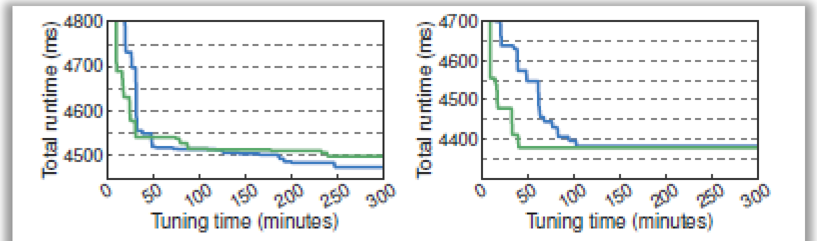
(a) MySQL (b) Postgres

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.

06 Evaluation & Conclusion

└ Conclusion

- We presented a technique for tuning DBMS knob configurations by reusing training data gathered from previous tuning sessions

- Uses a combination of supervised and unsupervised machine learning methods
  1) Select most impactful knobs
  2) Map previously unseen database workloads to known workloads
  3) Recommend knob settings

- Achieves up to 94% lower latency compared to default settings or configurations generated by other tuning advisors

감사합니다

Q/A