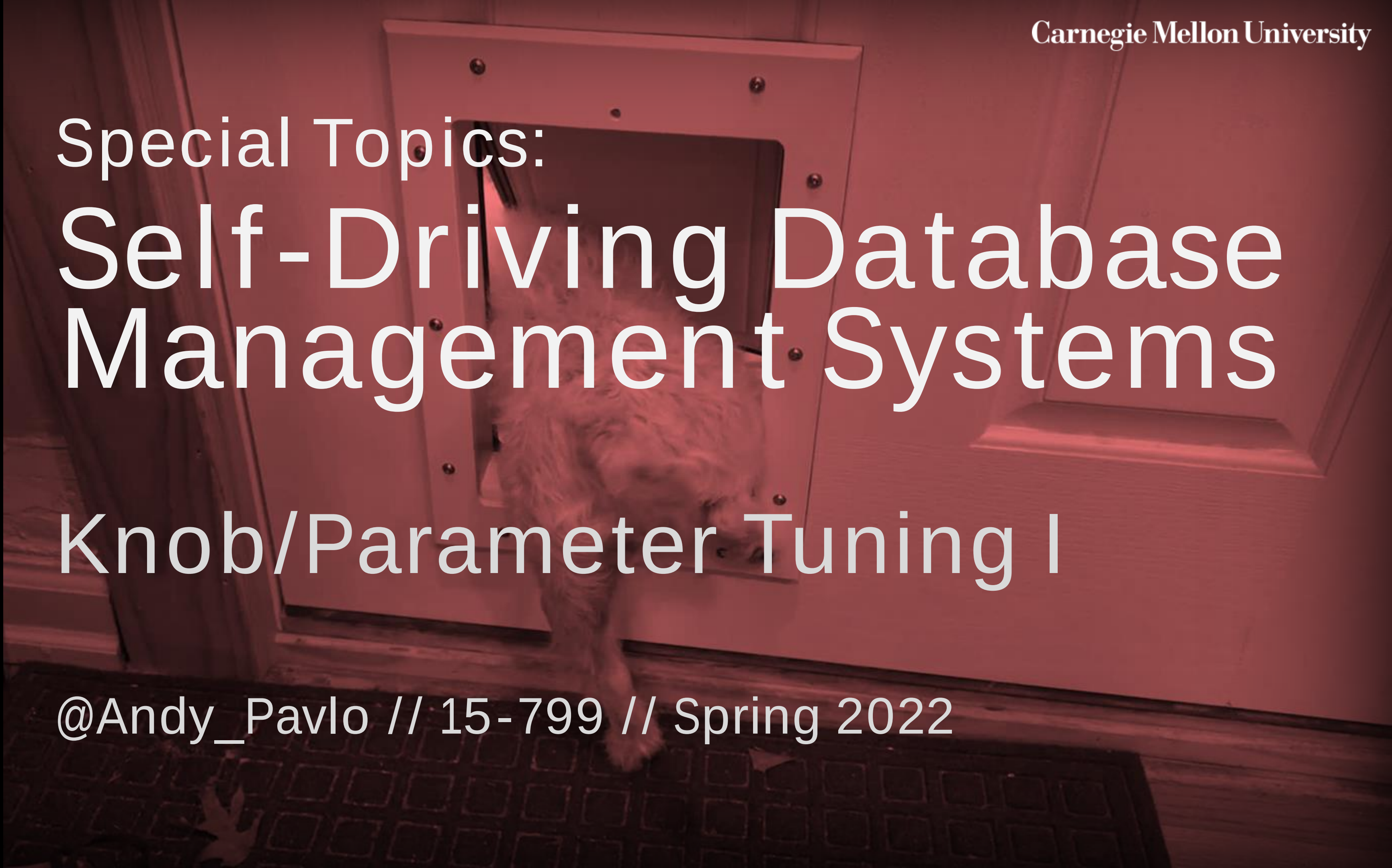

Lecture #06
Carnegie Mellon University
Special Topics:
Self-Driving Database Management Systems
Knob/Parameter Tuning I
@Andy_Pavlo // 15-799 // Spring 2022

ADMINISTRIVIA

Project #1 Instructions Updated

DB Speaker Today @ 4:30pm ET (Zoom)
→ Jake Moshenko – SpiceDB

LAST CLASS

We spent the last two weeks talking about how to do automatic index selection.

The key idea from this work was that an automated tool could leverage the DBMS's optimizer to determine whether an index configuration is good or not.

→ Query Plan
→ Cost Model Estimations

CONFIGURATION KNOBS

Every DBMS exposes knobs that control the runtime behavior of the system.

Tuning the knobs for a workload improves the DBMS's performance & efficiency.

TODAY'S AGENDA

Knob Tuning Basics
OtterTune
Challenges in the Real-World
Project #1

WHY KNOB TUNING IS HARD

Managing many configuration knobs on many DBMS instances exceeds the abilities of humans.

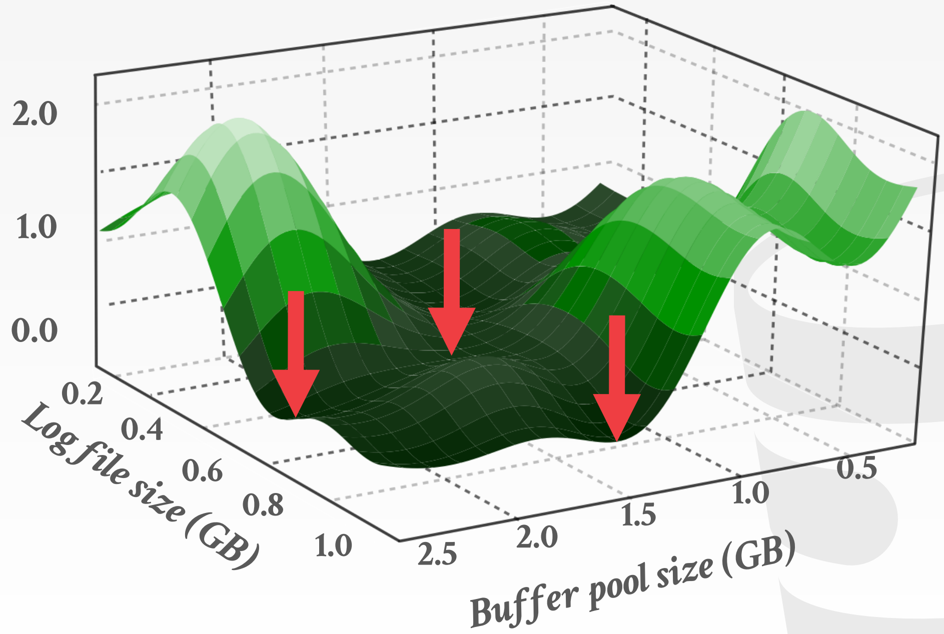
Three problems:

→ Implicit Dependencies Between Knobs
→ Non-Reusable Configurations
→ Complexity

PROBLEM #1: DEPENDENCIES

MySQL v5.6, YCSB: update-heavy workload
VM - 2GB RAM, 2 vCPUs

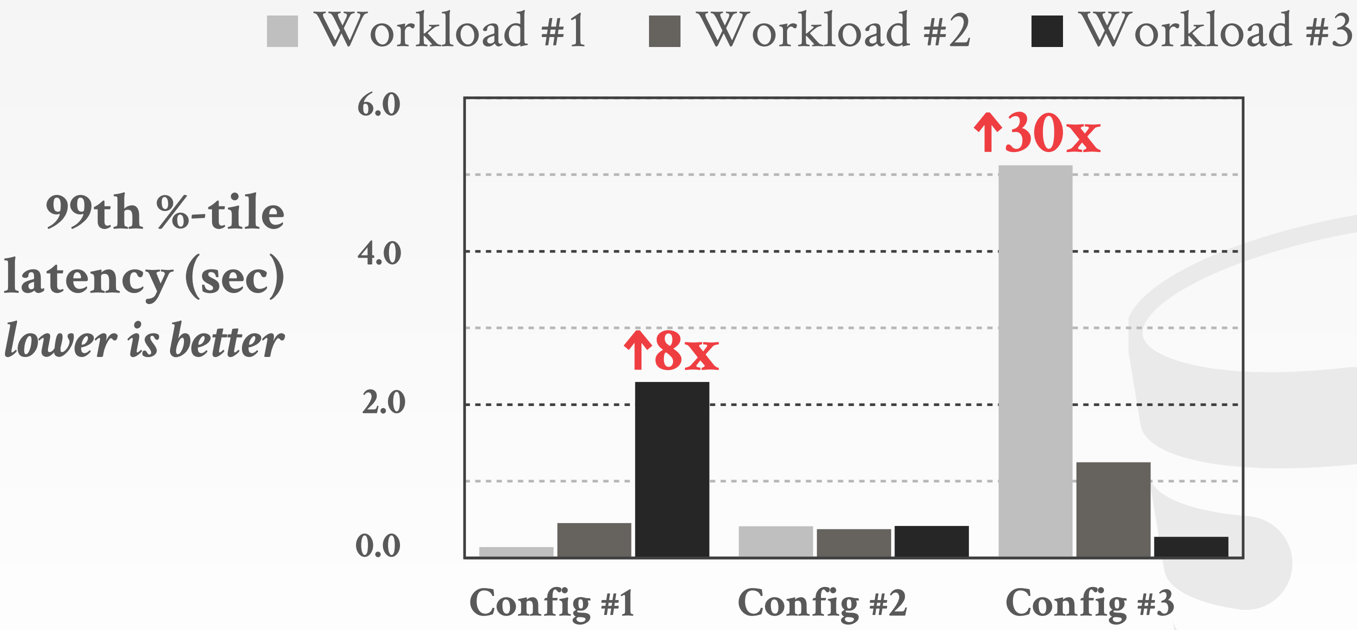
99th %-tile latency (sec)
lower is better



Source: Dana Van Aken

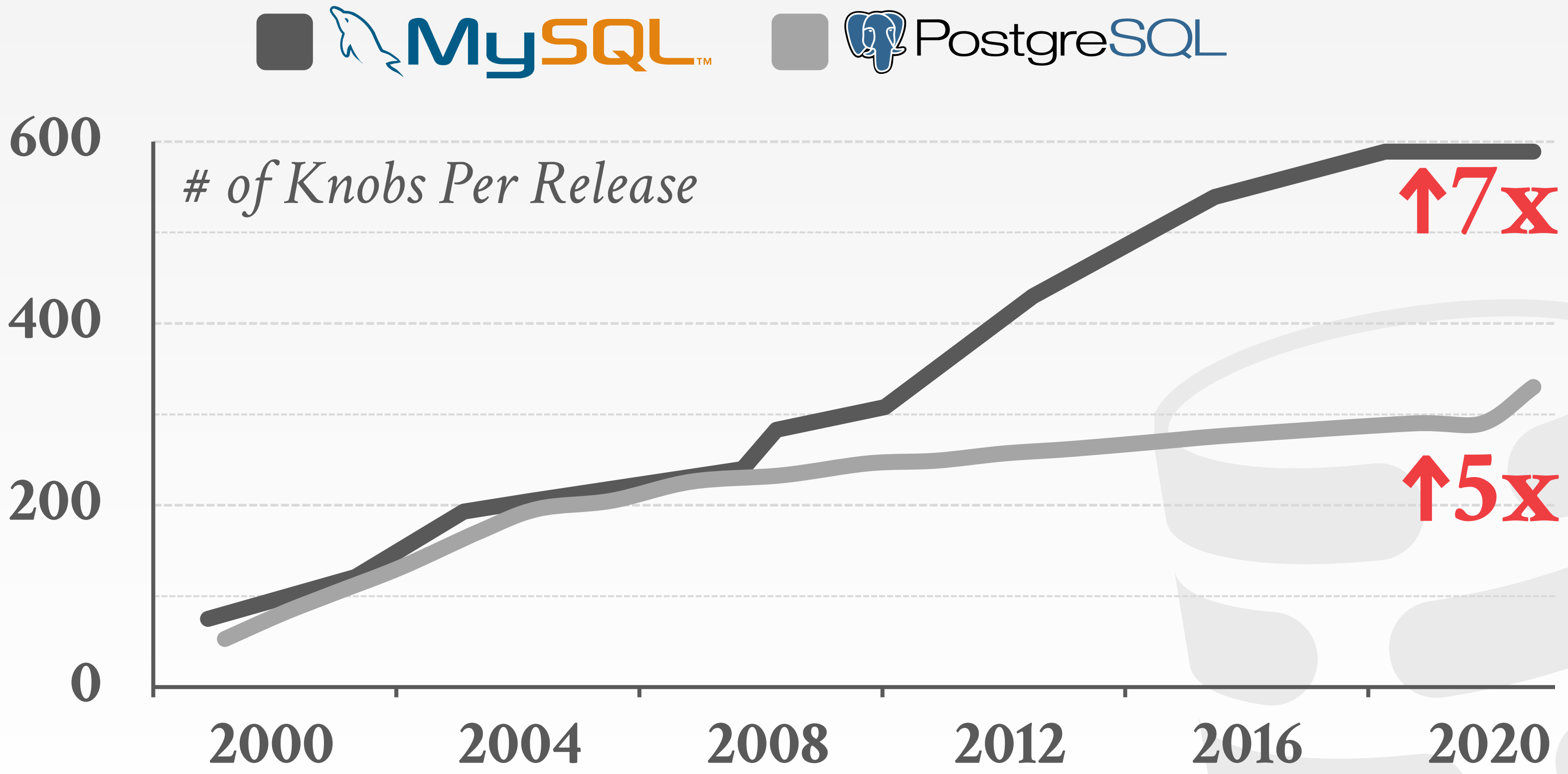
PROBLEM #2: NON-REUSABLE CONFIGS

MySQL v5.6, 3 different YCSB workloads



Source: Dana Van Aken

PROBLEM #3: COMPLEXITY



OVERVIEW

OtterTune is a knob configuration tuning service.

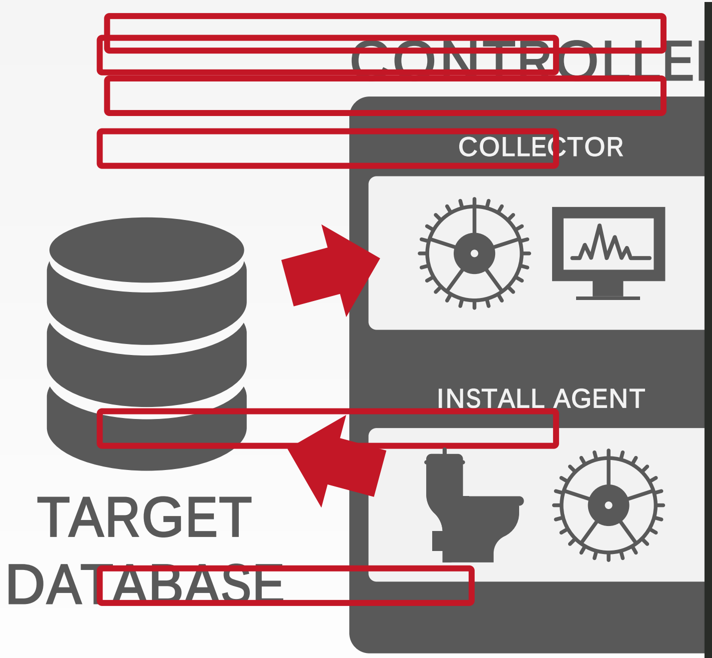
It uses machine learning to automatically optimize the knob configuration of DBMSs to improve their performance and reduce hardware/software costs.

Key Idea: Try to reuse data from previous tuning sessions to reduce tuning times for new databases.

AUTOMATIC DATABASE MANAGEMENT SYSTEM TUNING THROUGH LARGE-SCALE MACHINE LEARNING
SIGMOD 2017

A DEMONSTRATION OF THE OTTERTUNE AUTOMATIC DATABASE MANAGEMENT SYSTEM TUNING SERVICE
VLDB 2018

OTTERTUNE ARCHITECTURE




```
mysql> SHOW GLOBAL STATUS;
+-----------------------------------+-----------+
| METRIC_NAME                       | VALUE     |
+-----------------------------------+-----------+
| ABORTED_CLIENTS                   | 0         |
| ABORTED_CONNECTS                  | 0         |
•••
| INNODB_BUFFER_POOL_BYTES_DATA     | 129499136 |
| INNODB_BUFFER_POOL_BYTES_DIRTY    | 76070912  |
| INNODB_BUFFER_POOL_PAGES_DATA     | 7904      |
| INNODB_BUFFER_POOL_PAGES_DIRTY    | 4643      |
| INNODB_BUFFER_POOL_PAGES_FLUSHED  | 25246     |
| INNODB_BUFFER_POOL_PAGES_FREE     | 0         |
| INNODB_BUFFER_POOL_PAGES_MISC     | 288       |
| INNODB_BUFFER_POOL_PAGES_TOTAL    | 8192      |
| INNODB_BUFFER_POOL_READS          | 15327     |
| INNODB_BUFFER_POOL_READ_AHEAD     | 0         |
| INNODB_BUFFER_POOL_READ_AHEAD_EVICT | 0       |
| INNODB_BUFFER_POOL_READ_AHEAD_RND | 0         |
| INNODB_BUFFER_POOL_READ_REQUESTS  | 2604302   |
| INNODB_BUFFER_POOL_WAIT_FREE      | 0         |
| INNODB_BUFFER_POOL_WRITE_REQUESTS | 562763    |
| INNODB_DATA_FSYNCS                | 2836      |
| INNODB_DATA_PENDING_FSYNCS        | 1         |
| INNODB_DATA_WRITES                | 28026     |
•••
| UPTIME                            | 5996      |
| UPTIME_SINCE_FLUSH_STATUS         | 5996      |
+-----------------------------------+-----------+
```

WORKLOAD CHARACTERIZATION

Goal: Find a set of metrics in OtterTune's data repository that best characterize workload.

Problem: Redundant metrics
→ Same but different units
→ Highly correlated

Solution: Prune to reduce dimensionality
→ Factor analysis to capture correlations
→ K-means to group correlated metrics
→ Select one metric from each cluster

KNOB IDENTIFICATION

Goal: Find the best knobs to tune.

Problem: Knobs have varying degrees of impact on the DBMS's performance

→ Which knobs matter?

→ How many to tune?

Solution: Automated ranking based on impact

→ Lasso to rank knobs by impact

→ Incremental knob selection to gradually increase the # of knobs

OTTERTUNE PIPELINE

Step #1: Workload Mapping

Step #2: Configuration Recommendation

STEP #1: WORKLOAD MAPPING

Goal: Match the target DBMS's workload to the most similar known workload in the data repository.

Method: For each known workload, compute a similarity score by comparing the performance measurements.

→ Similarity Score: average distance in performance measurements over all metrics.

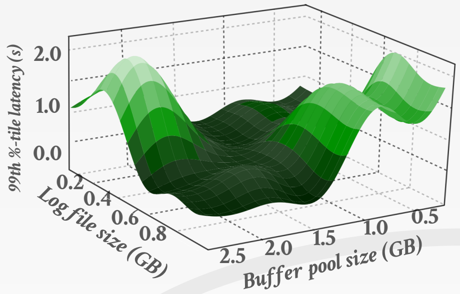
STEP #2: RECOMMENDATION

Goal: Recommend configurations to optimize the target objective.

Method: Reuse data from step #1 to train a statistical model that selects the next configuration to try.

RECOMMENDATION STEPS

Use similar historical data & new data to train a Gaussian Process model



Predict the latency & variance of any configuration

Optimize the next configuration, trading off:

→ Exploration: gathering more info to improve the model

→ Exploitation: greedily trying to do well on the target objective

EXPERIMENTAL SETUP

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

Training data collection

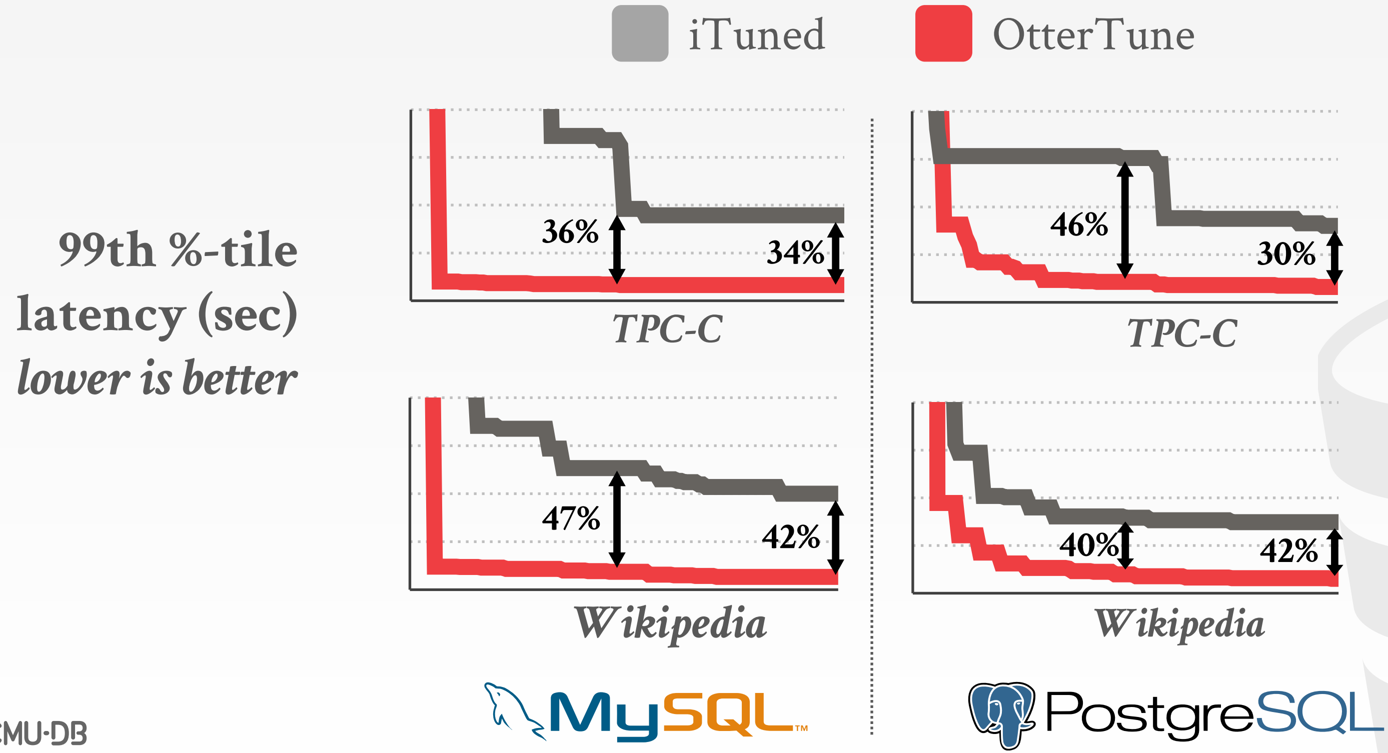
→ 15 YCSB workload mixtures

→ ~30k trials per DBMS

Experiments conducted on EC2

DATA VS. NO DATA

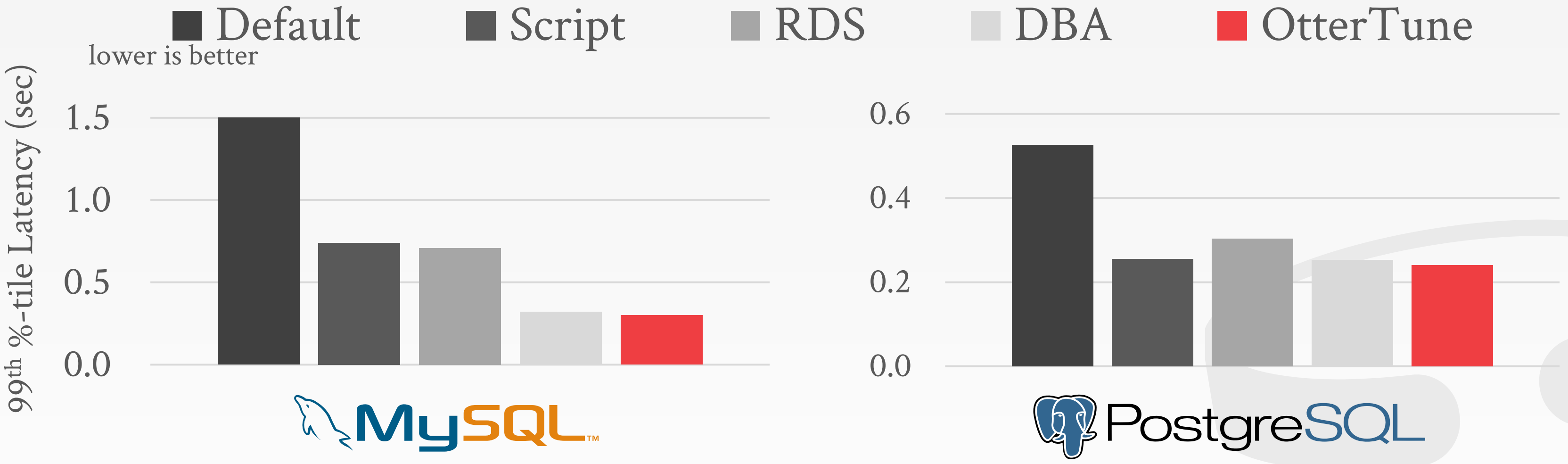
Training Time: Two Hours



Source: Dana Van Aken

EFFICACY COMPARISON: TPC-C

TPC-C: 200 warehouses, 50 terminals



Instance Type	99th-%tile Latency (ms)	Annual Cost
i3.4xlarge	372 (**default**)	\$6,964
i3.2xlarge	341 (**OtterTune**)	\$3,482

Annual savings of \$3,482 per instance

CHALLENGES

Table Knobs

Hardware Variations

Replaying Workload

Bad Configurations

External Factors

Restarts

TABLE KNOBS

OtterTune assumes that the number of knobs available to tune are fixed.

But some DBMSs support knobs that target individual tables, thereby overriding global knobs.

→ Example: Postgres has auto-vacuum knobs per table

Solution: Rank the top (5?) most "important" tables in the database and create virtual knobs.

HARDWARE VARIATIONS

OtterTune assumes that the hardware configuration is the same for all previous databases in its training data.

Solution: Include the hardware context in the input features of either the workload mapping or recommendation models.

REPLAYING WORKLOAD

Workload replay time depends on the configuration, but bad configurations could run for hours.

→ We need to keep the workload (almost) the same in each iteration to avoid metrics falsely reporting lower results.

Solution: Avoid long-running iterations by aborting the workload replay early.

BAD CONFIGURATIONS

When there is little prior data, algorithms will choose bad configurations that cause failures.

→ Scenario #1: Slow Execution
→ Scenario #2: DBMS Fails to Start
→ Scenario #3: DBMS Fails After Delay

Need to provide feedback to the algorithms that a configuration was bad.

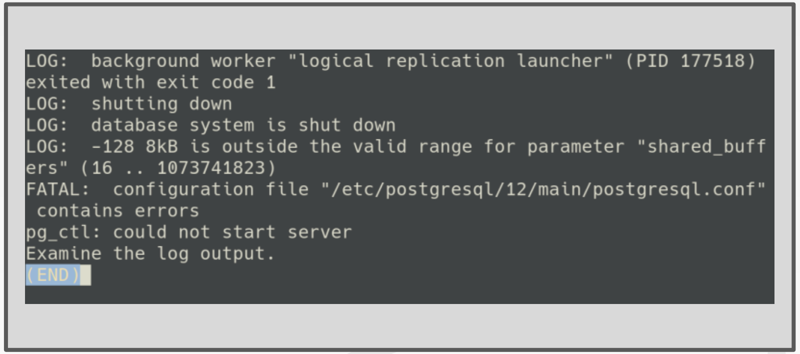
BAD CONFIGURATIONS

How to identify failure?

Solution: Scrape log for indicators of DBMS status.

How to incorporate results from bad configuration in the algorithm's training data?

Solution: Set the objective value to the worst configuration seen so far.



```
LOG:  background worker "logical replication launcher" (PID 177518)
exited with exit code 1
LOG:  shutting down
LOG:  database system is shut down
LOG:  -128 8kB is outside the valid range for parameter "shared_buff
ers" (16 .. 1073741823)
FATAL:  configuration file "/etc/postgresql/12/main/postgresql.conf"
 contains errors
pg_ctl: could not start server
Examine the log output.
(END)
```

EXTERNAL FACTORS

DBAs may set knobs to certain values due to reasons that are not readily discernible by an automated tuning service.

→ Example: InnoDB buffer pool size

Solution: Allow humans to specify the allowed value ranges of knobs to guide the tuning algorithms.

Approach 1: Rule-of-Thumb Method

The most commonly followed practice is to set this value at 70%-80% of the system RAM. Though it works well in most cases, this method may not be optimal in all configurations. Let's take the example of a system with 192GB of RAM. Based on the above method, we arrive at about 150GB for the buffer pool size. However, this isn't really an optimal number, as it does not fully leverage the large RAM size that's available in the system and leaves behind about 40GB of memory. This difference can be even more significant as we move to systems with larger configurations where we should be utilizing the available RAM to a greater extent.

Knob name	Allowed values
☐ innodb_adaptive_hash_index Whether innodb adaptive hash indexes are enabled or disabled	on, off
☑ innodb_adaptive_max_sleep_delay Allows InnoDB to automatically adjust the value of innodb_thread_sleep_delay up or down according to the current workload.	0 us – 300000 us
☑ innodb_buffer_pool_instances The number of regions that the InnoDB buffer pool is divided into	1 – 2
☑ innodb_buffer_pool_size The size in bytes of the memory buffer innodb uses to cache data and indexes of its tables	5242880 B – 805306368 B
☑ innodb_change_buffering Controls InnoDB change buffering	inserts, deletes, purges, changes, all

SYSTEM RESTARTS

Some DBMS knobs do not take effect until you restart the system. But nobody likes to restart their DBMS unless they really must in order to minimize downtime.

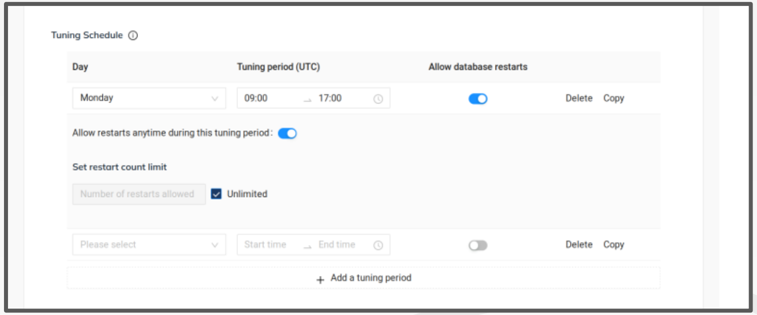
The time it takes to bring the DBMS back online after restart can be non-deterministic based on the configuration changes.

→ Example: MySQL log file size

SYSTEM RESTARTS

How to know whether the service is allowed to restart DBMS?

Solution: A human must tell us.



How to estimate how long it will take the DBMS to restart?

Solution: Open problem. Lots of factors…

PARTING THOUGHTS

Knob tuning is an important step in database administration.

The performance difference is significant.

PROJECT #1: POSTGRES AUTO TUNER

You will build an automatic tuning tool for Postgres:

→ The focus is on indexes, but you are free to make other changes you want.
→ The database must return correct results.

Due Date: March 2nd @ 11:59pm

More Info:

https://15799.courses.cs.cmu.edu/spring2022/project1.html

PROJECT #1: POSTGRES AUTO TUNER

PROJECT #1: GETTING STARTED

Start with a simple implementation.

→ Remove indexes from TPC-C, then add them back with your tool (hardcoded).

Iteratively improve your workload analysis.

→ Select the top 5 most frequently referenced columns that are used together in **WHERE** clauses.

Then connect to the DBMS and retrieve additional information from the system.

Project #1: Resources

[Python Doit Framework]
[Dexter Index Tuner]
[BenchBase Documentation]

NEXT CLASS

More knob tuning!