AWS Machine Learning Blog

Tuning Your DBMS Automatically with Machine Learning

by Dana Van Aken, Geoff Gordon, and Andy Pavlo | on 02 JUN 2017 | Permalink | Comments | Share

This is a guest post by Dana Van Aken, Andy Pavlo, and Geoff Gordon of Carnegie Mellon University. This project demonstrates how academic researchers can leverage our AWS Cloud Credits for Research Program to support their scientific breakthroughs.

Database management systems (DBMSs) are the most important component of any data-intensive application. They can handle large amounts of data and complex workloads. But they're difficult to manage because they have hundreds of configuration "knobs" that control factors such as the amount of memory to use for caches and how often to write data to storage. Organizations often hire experts to help with tuning activities, but experts are prohibitively expensive for many.

OtterTune, a new tool that's being developed by students and researchers in the Carnegie Mellon Database Group, can automatically find good settings for a DBMS's configuration knobs. The goal is to make it easier for anyone to deploy a DBMS, even those without any expertise in database administration.

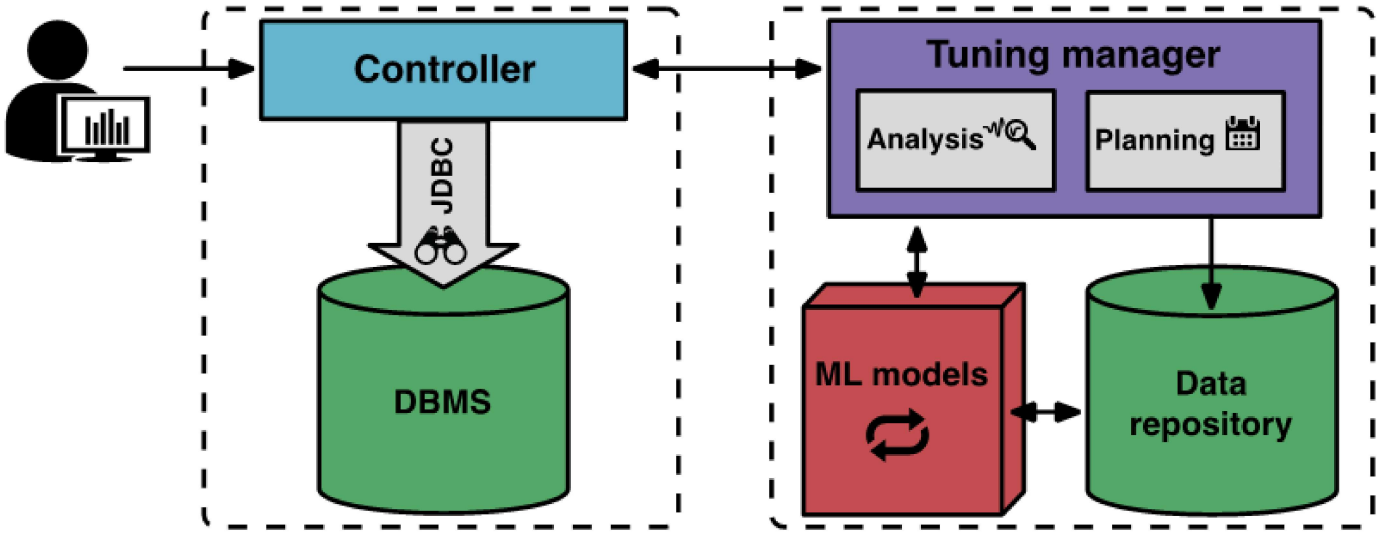
OtterTune differs from other DBMS configuration tools because it leverages knowledge gained from tuning previous DBMS deployments to tune new ones. This significantly reduces the amount of time and resources needed to tune a new DBMS deployment. To do this, OtterTune maintains a repository of tuning data collected from previous tuning sessions. It uses this data to build machine learning (ML) models that capture how the DBMS responds to different configurations. OtterTune uses these models to guide experimentation for new applications, recommending settings that improve a target objective (for example, reducing latency or improving throughput).

In this post, we discuss each of the components in OtterTune's ML pipeline, and show how they interact with each other to tune a DBMS's configuration. Then, we evaluate OtterTune's tuning efficacy on MySQL and Postgres by comparing the performance of its best configuration with configurations selected by database administrators (DBAs) and other automatic tuning tools.

OtterTune is an open source tool that was developed by students and researchers in the Carnegie Mellon Database Research Group. All code is available on GitHub, and is licensed under Apache License 2.0.

How OtterTune works

The following diagram shows the OtterTune components and workflow.



At the start of a new tuning session, the user tells OtterTune which target objective to optimize (for example, latency or throughput). The client-side controller connects to the target DBMS and collects its Amazon EC2 instance type and current configuration.

Then, the controller starts its first observation period, during which it observes the DBMS and records the target objective. When the observation period ends, the controller collects internal metrics from the DBMS, like MySQL's counters for pages read from disk and pages written to disk. The controller returns both the target objective and the internal metrics to the tuning manager.

When OtterTune's tuning manager receives the metrics, it stores them in its repository. OtterTune uses the results to compute the next configuration that the controller should install on the target DBMS. The tuning manager returns this configuration to the controller, with an estimate of the expected improvement from running it. The user can decide whether to continue or terminate the tuning session.

Notes

OtterTune maintains a blacklist of knobs for each DBMS version it supports. The blacklist includes knobs that don't make sense to tune (for example, path names for where the DBMS stores files), or those that could have serious or hidden consequences (for example, potentially causing the DBMS to lose data). At the beginning of each tuning session, OtterTune provides the blacklist to the user so he or she can add any other knobs that they want OtterTune to avoid tuning.

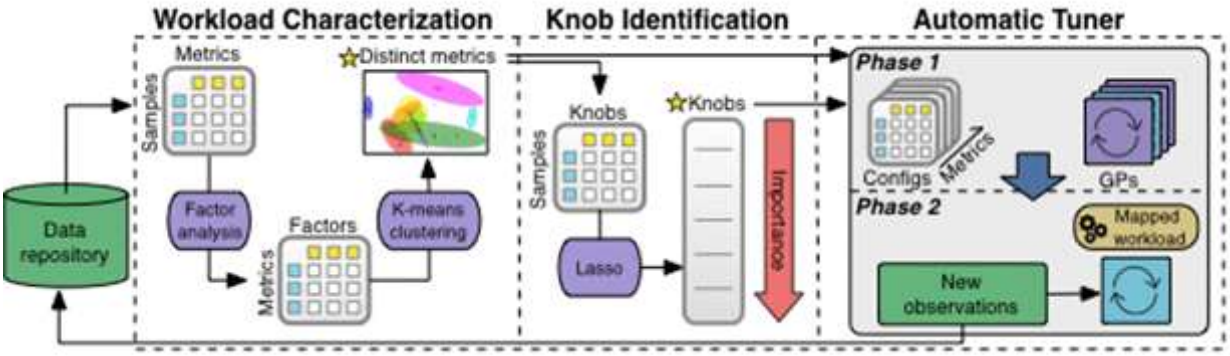
OtterTune makes certain assumptions that might limit its usefulness for some users. For example, it assumes that the user has administrative privileges that allow the controller to modify the DBMS's configuration. If the user doesn't, then he or she can deploy a second copy of the database on other hardware for OtterTune's tuning experiments. This requires the user to either replay a workload trace or to forward queries from the production DBMS. For a complete discussion of assumptions and limitations, see our paper.

The machine learning pipeline

The following diagram shows how data is processed as it moves through OtterTune's ML pipeline. All observations reside in OtterTune's repository.

OtterTune first passes observations into the Workload Characterization component. This component identifies a smaller set of DBMS metrics that best capture the variability in performance and the distinguishing characteristics for different workloads.

Next, the Knob Identification component generates a ranked list of the knobs that most affect the DBMS's performance. OtterTune then feeds all of this information to the Automatic Tuner. This component maps the target DBMS's workload to the most similar workload in its data repository, and reuses this workload data to generate better configurations.



Let's drill down on each of the components in the ML pipeline.

Workload Characterization: OtterTune uses the DBMS's internal runtime metrics to characterize how a workload behaves. These metrics provide an accurate representation of a workload because they capture many aspects of its runtime behavior. However, many of the metrics are redundant: some are the same measurement recorded in different units, and others represent independent components of the DBMS whose values are highly correlated. It's important to prune redundant metrics because that reduces the complexity of the ML models that use them. To do this, we cluster the DBMS's metrics based on their correlation patterns. We then select one representative metric from each cluster, specifically, the one closest to the cluster's center. Subsequent components in the ML pipeline use these metrics.

Knob Identification: DBMSs can have hundreds of knobs, but only a subset affects the DBMS's performance. OtterTune uses a popular feature-selection technique, called *Lasso*, to determine which knobs strongly affect the system's overall performance. By applying this technique to the data in its repository, OtterTune identifies the order of importance of the DBMS's knobs.

Then, OtterTune must decide how many of the knobs to use when making configuration recommendations. Using too many of them significantly increases OtterTune's optimization time. Using too few could prevent OtterTune from finding the best configuration. To automate this process, OtterTune uses an incremental approach. It gradually increases the number of knobs used in a tuning session. This approach allows OtterTune to explore and optimize the configuration for a small set of the most important knobs before expanding its scope to consider others.

Automatic Tuner: The Automated Tuning component determines which configuration OtterTune should recommend by performing a two-step analysis after each observation period.

First, the system uses the performance data for the metrics identified in the Workload Characterization component to identify the workload from a previous tuning session that best represents the target DBMS's workload. It compares the session's metrics with the metrics from previous workloads to see which ones react similarly to different knob settings.

Then, OtterTune chooses another knob configuration to try. It fits a statistical model to the data that it has collected, along with the data from the most similar workload in its repository. This model lets OtterTune predict how well the DBMS will perform with each possible configuration. OtterTune optimizes the next configuration, trading off exploration (gathering information to improve the model) against exploitation (greedily trying to do well on the target metric).

Implementation

OtterTune is written in Python.

For the Workload Characterization and Knob Identification components, runtime performance isn't a key concern, so we implemented the corresponding ML algorithms with scikit-learn. These algorithms run in background processes, incorporating new data as it becomes available in OtterTune's repository.

For the Automatic Tuner, the ML algorithms are on the critical path. They run after each observation period, incorporating new data so that OtterTune can pick a knob configuration to try next. Because performance is a consideration, we implemented these algorithms using TensorFlow.

To collect data about the DBMS's hardware, knob configurations, and runtime performance metrics, we integrated OtterTune's controller with the OLTP-Bench benchmarking framework.

Experiment design

To evaluate, we compared the performance of MySQL and Postgres using the best configuration selected by OtterTune with the following:

- Default: The configuration provided by the DBMS
- Tuning script: The configuration generated by an open source tuning advisor tool
- DBA: The configuration chosen by a human DBA
- RDS: The configuration customized for the DBMS that is managed by Amazon RD and deployed on the same EC2 instance type

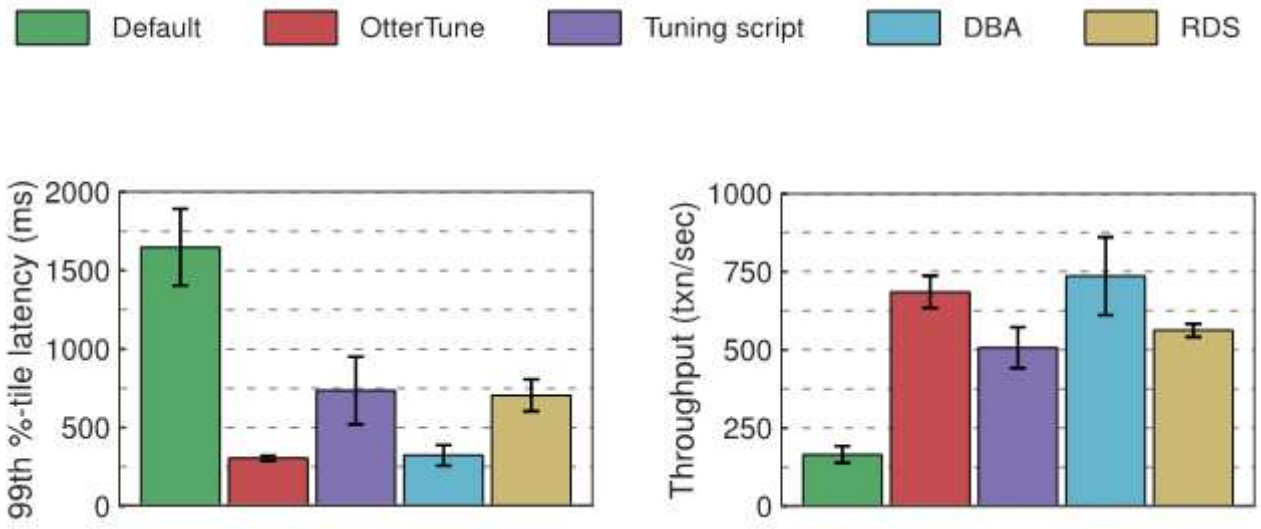
We conducted all of our experiments on Amazon EC2 Spot Instances. We ran each experiment on two instances: one for OtterTune's controller and one for the target DBMS deployment. We used the m4.large and m3.xlarge instance types, respectively. We deployed OtterTune's tuning manager and data repository on a local server with 20 cores and 128 GB of RAM.

We used the TPC-C workload, which is the industry standard for evaluating the performance of online transaction processing (OLTP) systems.

Evaluation

For each database we used in our experiment, MySQL and Postgres, we measured latency and throughput. The following graphs show the results. The first graph shows the amount of 99^{th} percentile latency, which represents the "worst case" length of time that it takes a transaction to complete. The second graph shows results for throughput, measured as the average number of transactions completed per second.

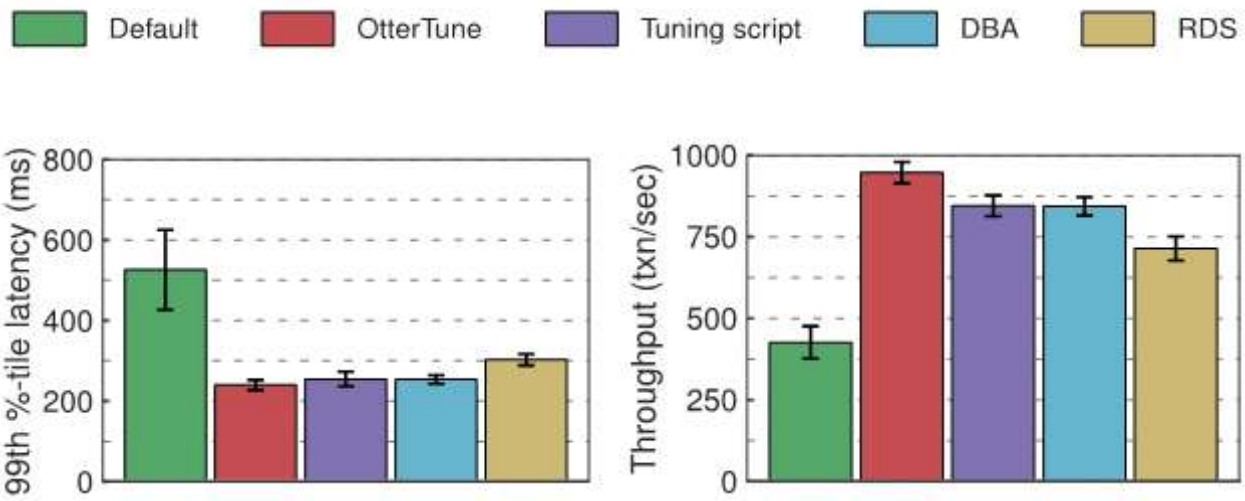
MySQL results



Comparing the best configuration generated by OtterTune with configurations generated by the tuning script and RDS, MySQL achieves approximately a 60% reduction in latency and 22% to 35% better throughput with the OtterTune configuration. OtterTune also generates a configuration that is almost as good as one chosen by the DBA.

Just a few of MySQL's knobs significantly affect its performance for the TPC-C workload. The configurations generated by OtterTune and the DBA provide good settings for each of these knobs. RDS performs slightly worse because it provides a suboptimal setting for one knob. The tuning script's configuration performs the worst because it modifies only one knob.

Postgres results



For latency, the configurations generated by OtterTune, the tuning tool, the DBA, and RDS all achieve similar improvements over Postgres' default settings. We can probably attribute this to the overhead required for round trips between the OLTP-Bench client and the DBMS over the network. For throughput, Postgres performs approximately 12% better with the configuration suggested by OtterTune than with the configurations chosen by the DBA and the tuning script, and approximately 32% better compared to RDS.

Similar to MySQL, only a few knobs significantly affect Postgres' performance. The configurations generated by OtterTune, the DBA, the tuning script, and RDS all modified these knobs, and most provided reasonably good settings.

Conclusion

OtterTune automates the process of finding good settings for a DBMS's configuration knobs. To tune new DBMS deployments, it reuses training data gathered from previous tuning sessions. Because OtterTune doesn't need to generate an initial dataset for training its ML models, tuning time is drastically reduced.

What's next? To accommodate the growing popularity of DBaaS deployments, where remote access to the DBMS's host machine isn't available, OtterTune will soon be able to automatically detect the hardware capabilities of the target DBMS without requiring remote access.

For more details about OtterTune, see our paper or the code on GitHub. Keep an eye on this website, where we will soon make OtterTune available as an online-tuning service.

About the Authors

Dana Van Aken is a PhD student in Computer Science at Carnegie Mellon University advised by Dr. Andrew Pavlo. Her broad research interest is in database management systems. Her current work focuses on developing automatic techniques for tuning database management systems using machine learning.

Dr. Andy Pavlo is an Assistant Professor of Databaseology in the Computer Science Department at Carnegie Mellon University. At CMU, he is a member of the Database Group and the Parallel Data Laboratory. His work is also in collaboration with the Intel Science and Technology Center for Big Data.

Dr. Geoff Gordon is Associate Professor and Associate Department Head for Education in the Department of Machine Learning at Carnegie Mellon University. His research interests include artificial intelligence, statistical machine learning, educational data, game theory, multi-robot systems, and planning in probabilistic, adversarial, and general-sum domains.

Related Posts

AWS awarded PROTECTED certification in Australia

Alerting, monitoring, and reporting for PCI-DSS awareness with Amazon Elasticsearch Service and AWS

Lambda

Add a layer of security for AWS SSO user portal sign-in with context-aware email-based verification

Using AWS IoT Analytics to Prepare Data for QuickSight Time-Series Visualizations

Migrate Wildfly Cluster to Amazon ECS using Service Discovery

Build and deploy an application for Hyperledger Fabric on Amazon Managed Blockchain

Archiving Data to Amazon S3 Glacier using PowerShell

AWS Security profiles: Michael South, Principal Business Development Manager for Security Acceleration