Tuning databases for better performance

A project exploring Bayesian Optimization for RocksDB

Osama Eldawebi

May 2021

1 Intro: deep dive

Optimization

- Methods for black-box functions
  - Random search (RS)
    - Initialize with random sample -> move to a better position around sample
  - Grid search (GS)
    - Exhaustive search using manually specified parameters and their value boundaries.
  - Bayesian optimization (BO)
    - Next slide

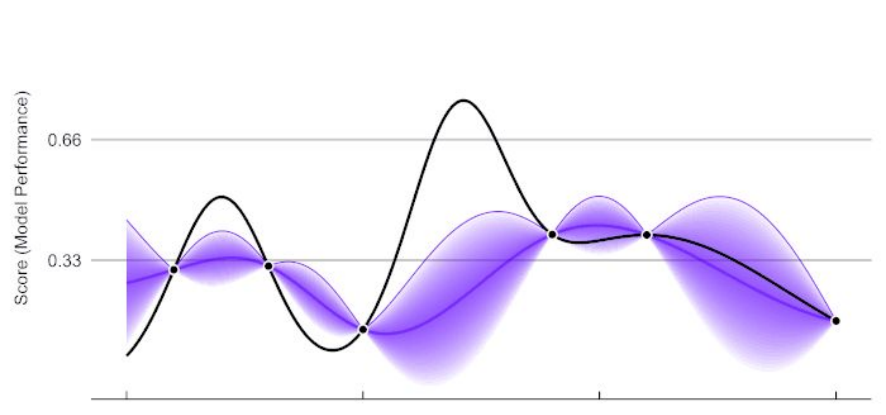
Bayesian Optimization

How it works:

1. Start with a prior for the function we want to optimize

2. Update posterior distribution (PD) by evaluating sample

3. Create acquisition function to decide next sample
   - Expected improvement (EI)



Source: Wikipedia

Application: Databases

- Optimize a database using its parameters
  - Many recent papers exploring this

- Examples: MySQL, PostgreSQL, Cassandra, and RocksDB.
  - Throughput/s (TPS) vs latency vs memory optimization, and more.

- Database parameters (features) are also called *knobs*
  - Varies between databases and seem to increase in number over time

2 Project overview

Outline

The project problem

Use Bayesian optimization (BO) to find highest TPS.

- consistent hardware

Choice of optimizer

DBtune

an online service and used as an API.

- Warm-up phase to decide search space feasibility
- ML model (trees) to decide next samples

The Database

RocksDB

- NoSQL relational database
- Fast and embedded data storage
- Stores keys and values as byte streams

Pipeline

Domain knowledge

Phase 0

Goal: limit parameter search space

Steps:

a) Review relevant literature

b) Review RocksDB documentation

c) Decide on best features

Feature importance

Phase 1

Goal: *Only use most significant features*

Steps:

a) evaluate the decision trees in the ML model of DBTune in its warm-up phase

b) Filter out features below a significance level of 2%

Find the optimum

Phase 2

Goal: Find set of knobs that lead to highest TPS

Steps:

a) DBtune will optimize over many iterations

b) Store best TPS and configuration found so far

Benchmarking with RocksDB

- Workload: random reads and writes on multiple threads using internal tool
  - Execute different ratios of reads to writes: **10:90, 50:50, 90:10**

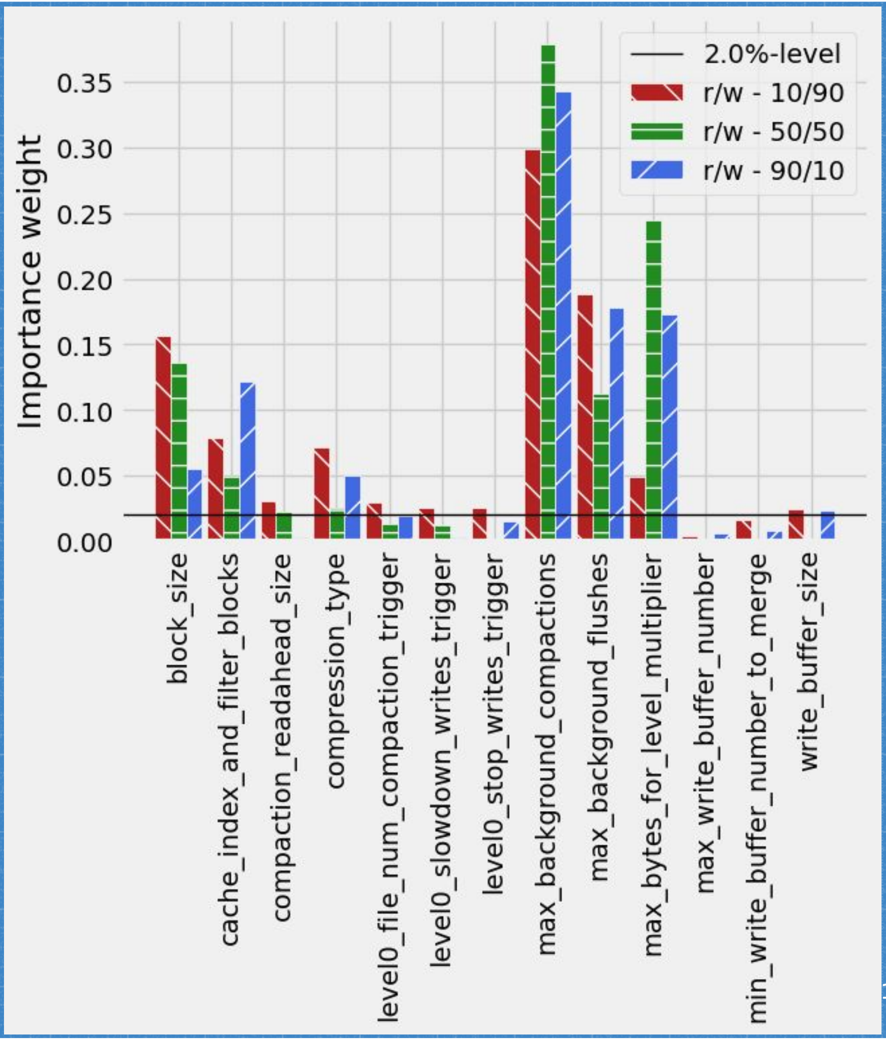
Steps:

1. Fill database with 5 million key-value pairs
2. Run workload benchmark for maximum X minutes (X = 5).
3. Evaluate TPS

3 Results

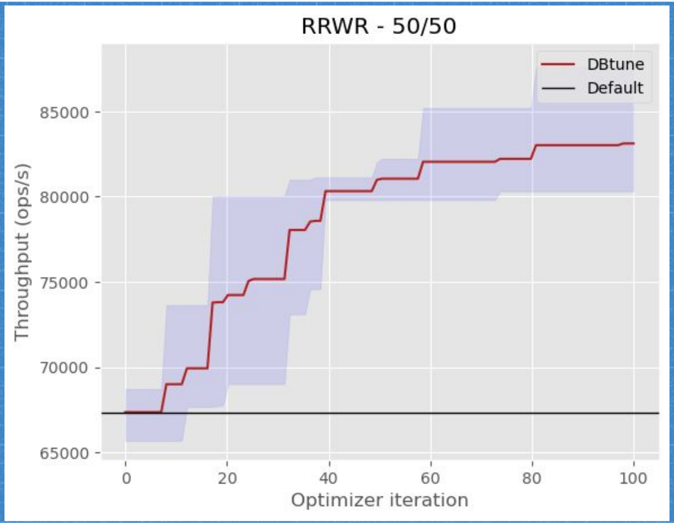
Phase 1:
Feature Importance

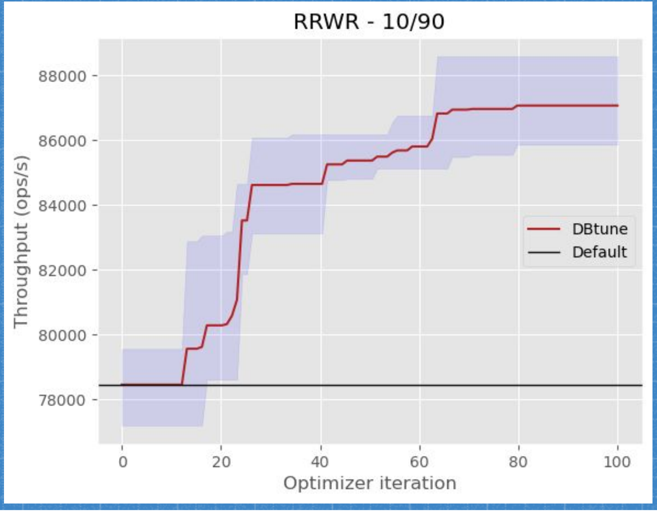
- ➔ 130 samples were used in DBTune's warm-up phase
- ➔ Importance weights are received from DBtune
- ➔ From 13 initial knobs to 11 at 2%-pruning condition

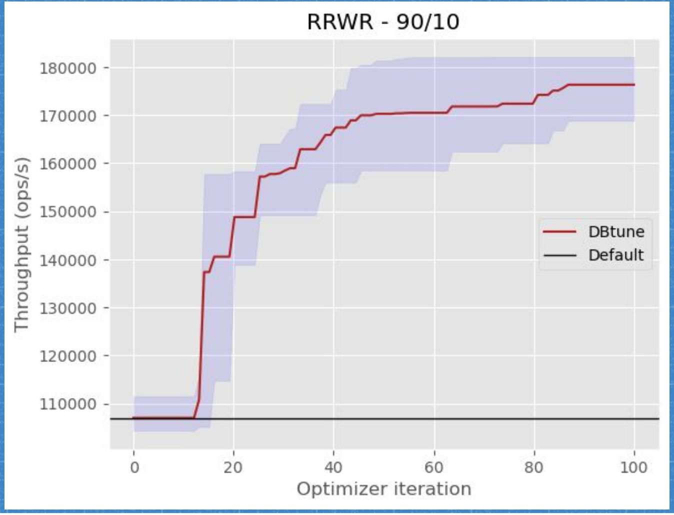


Phase 2:
Finding the optimum

Read/write ratio	Improvement (%)
10/90	**11.0**
50/50	**23.4**
90/10	**64.7**







Phase 2:
Finding the optimum

Insights:

- ➔ No sign of convergence, 100 iterations perhaps not enough.
- ➔ Higher read:write seems to cause greater improvement from default.
- ➔ Predominant read:write leads to high TPS.

3 Future work

Suggestions

Other methods

- ➔ Compare with search-based methods
- ➔ Other learning-based methods or try improving the BO method

Benchmarking

- ➔ General: increase number of runs to average out, explore more read:write ratios, try a different hardware setup
- ➔ Increase optimization iterations to see a better sign of convergence
- ➔ Try other, more varied workload patterns that are exciting

Larger scope

- ➔ Explore other databases, maybe try comparing SQL vs NoSQL
- ➔ Optimize for more/other objectives
- ➔ Explore how to minimize the effects of *Curse of Dimensionality, i.e. how do we limit the search space even further?*

Special thanks

to the people working with **DBtune**

Thanks!

Email: osama.eldawebi@gmail.com

Repository for the interested: github.com/deslay1/DB-tuning