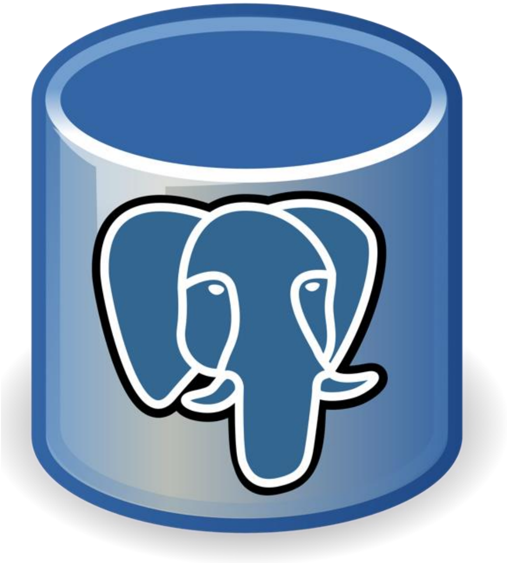
Robust Database Tuning with ENDURE

Andy Huynh, Harshal A. Chaudhari, Evimaria Terzi, Manos Athanassoulis

Databases Have Settings…



- effective_cache_size
- work_mem
- wal_sync_method
- max_prepared_transactions
- random_page_cost
- checkpoint_segments
- maintenance_work_mem

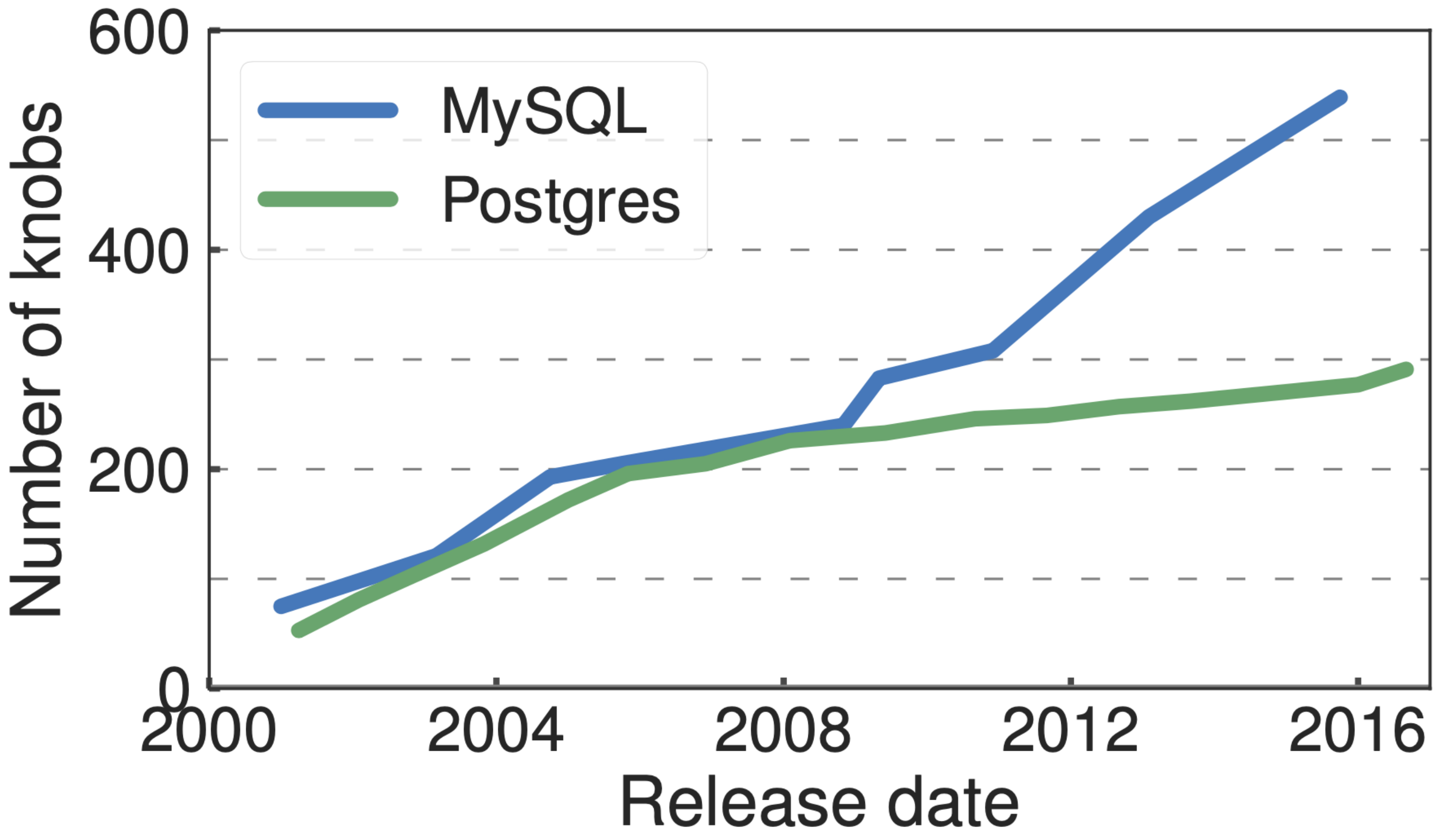
…

- shared_buffers

200+ settings

Determines performance

Database Complexity



Van Aken D. et. al., “Automatic Database Management System Tuning Through Large-scale Machine Learning”. SIGMOD 2017

Tuning Makes a HUGE Difference

Databricks Sets Official Data Warehousing Performance Record

by **Reynold Xin** and **Mostafa Mokhtar**
Posted in **COMPANY BLOG** | **Novembe**

Today, we are proud to announce that **Dat** **100TB TPC-DS**, the gold standard perforn **Databricks SQL outperformed the previo** benchmark news, this result has been forı

These results were corroborated by resea which frequently runs TPC-DS on popular **benchmarked Databricks and Snowflake** **and 12x better in terms of price performa** warehouses such as Snowflake become p production.

AUTHOR

Benoit Dageville

Thierry Cruanes

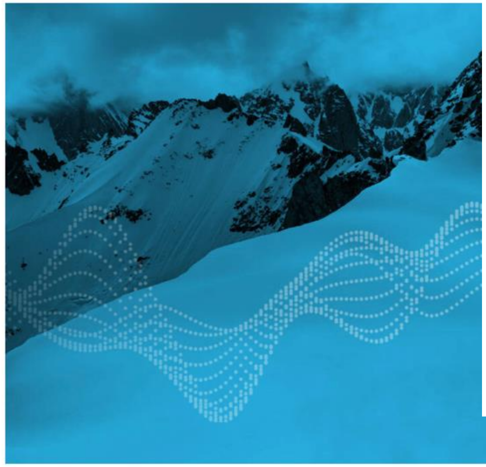
SHARE

SUBSCRIBE

NOV 12, 2021

Industry Benchmarks and Compet

Thought Leadership > Executive Platform



When we founded Snowflake, we set out to build an innovative platform. We had the opportunity to take into account what had worked well and what hadn't in prior architectures and implementations. We saw how we could leverage the cloud to rethink the limits of what was possible. We also focused on ease of use and building a system that "just worked." We knew there were many opportunities to improve upon prior implementations and innovate to lead on performance and scale, simplicity of administration, and data-driven collaboration.

Snowflake Claims Similar Price/Performance to Databricks, but Not So Fast!

by **Mostafa Mokhtar**, **Arsalan Tavakoli-Shiraji**, **Reynold Xin** and **Matei Zaharia**
Posted in **COMPANY BLOG** | **November 15, 2021**

On Nov 2, 2021, we announced that **we set the official world record** for the fastest data warehouse with our Databricks SQL lakehouse platform. These results were audited and reported by the official Transaction Processing Performance Council (TPC) in a 37-page document **available online** at tpc.org. We also shared a third-party benchmark by the Barcelona Supercomputing Center (BSC) outlining that Databricks SQL is significantly faster and more cost effective than Snowflake.

A lot has happened since then: many congratulations, some questions, and some sour grapes. We take this opportunity to reiterate that **we stand by our blog post and the results: Databricks SQL provides superior performance and price performance over Snowflake, even on data warehousing workloads (TPC-DS)**.

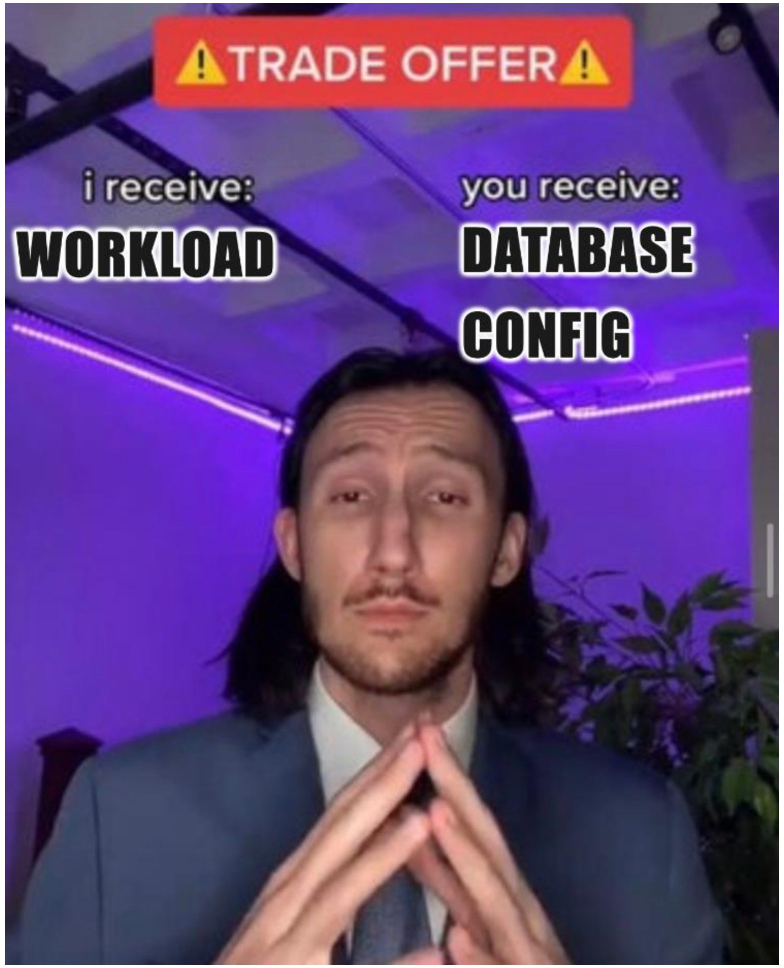
Tuning is Simple…?

Workload → "Tuner" → Database Configuration

What is a workload?

How do we tune?

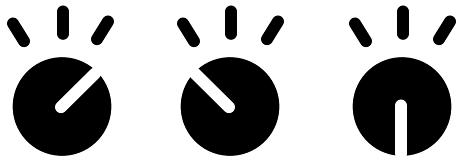
What are you optimizing for?



Age of Log-Structured Merge-Trees

RocksDB
levelDB
SQLite
cassandra
WT
accumulo™
APACHE HBASE
DynamoDB

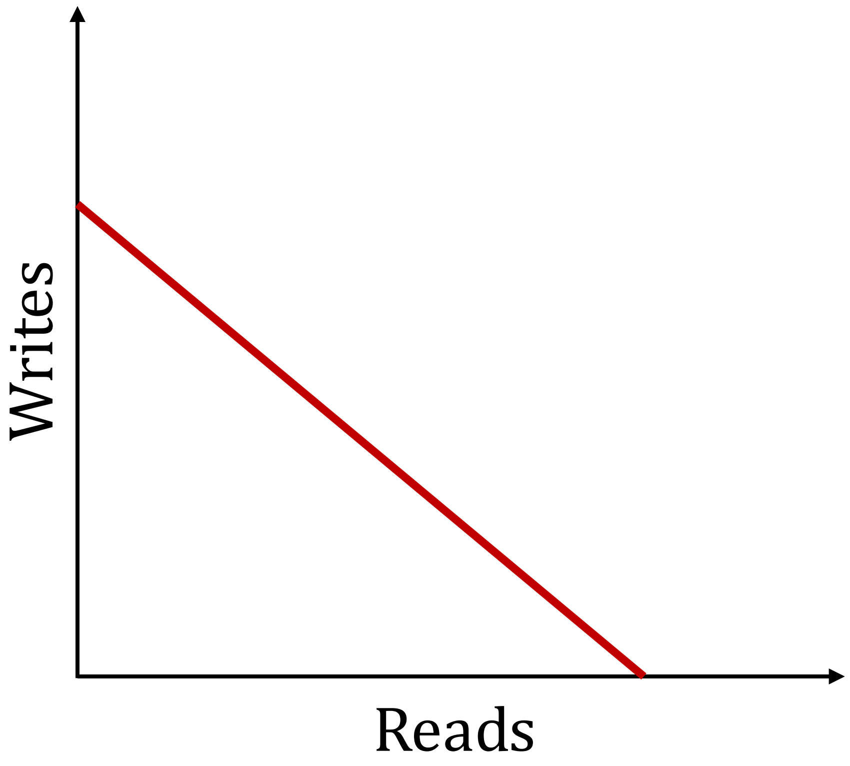
High impact tuning knobs

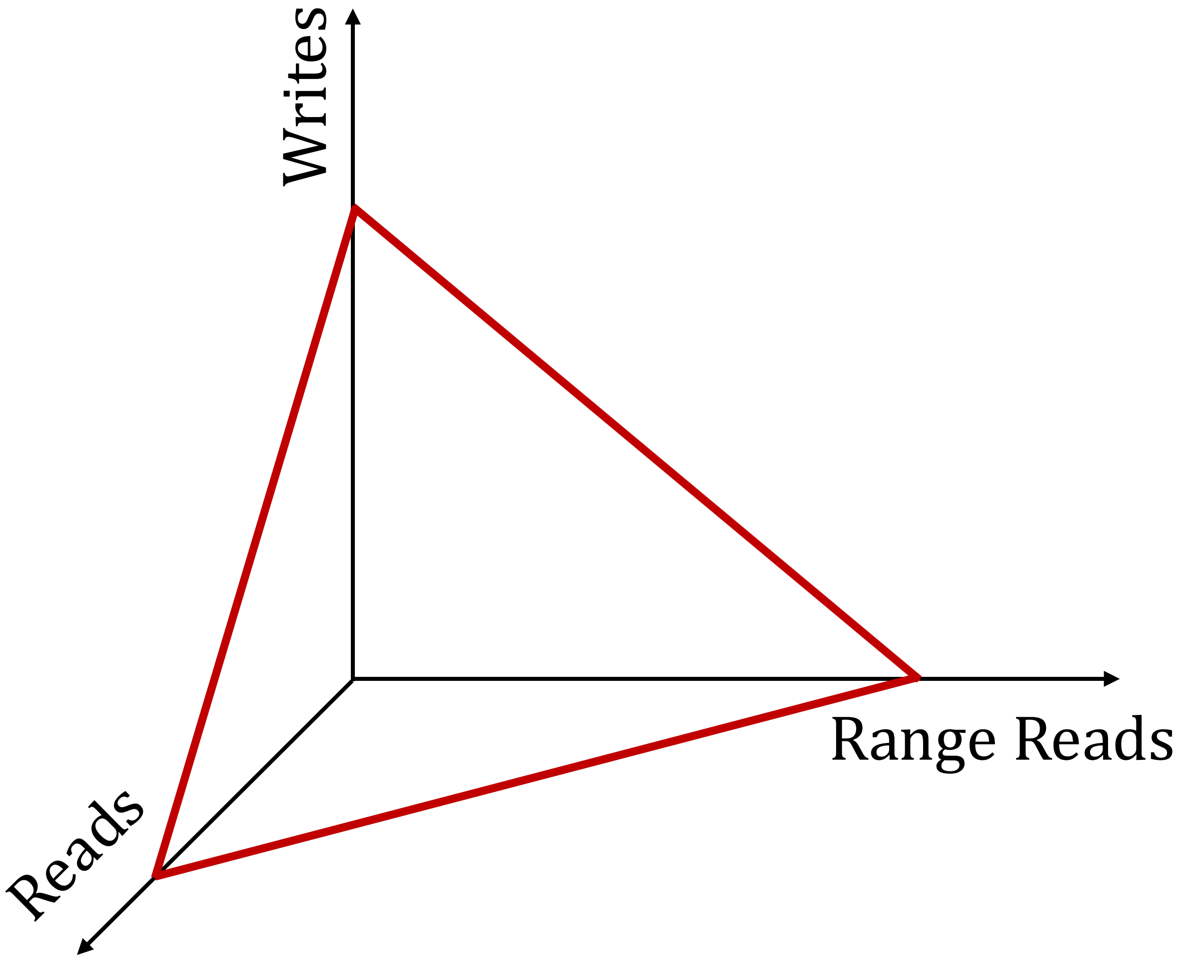


Dictates performance!

How do we go about tuning these knobs?

What is a Workload?

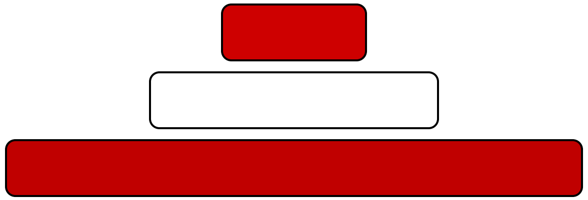




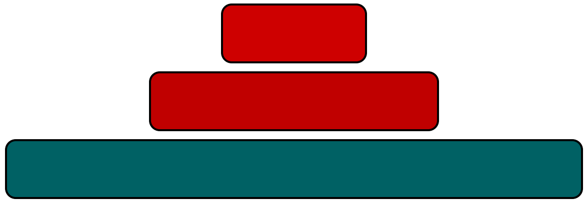
Query Types

Workload : (z_0, z_1, q, w)

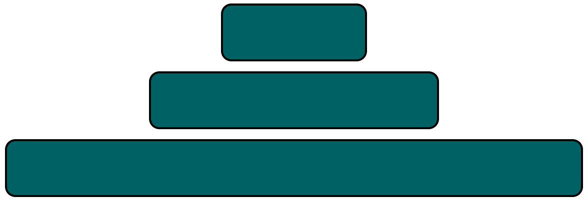
Empty Reads : z_0



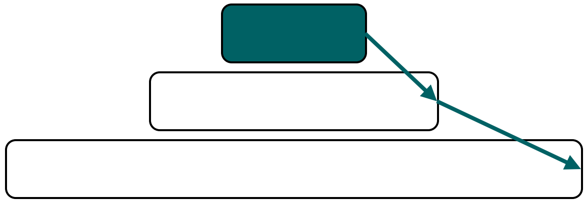
Non-Empty Reads : z_1



Range Reads: q

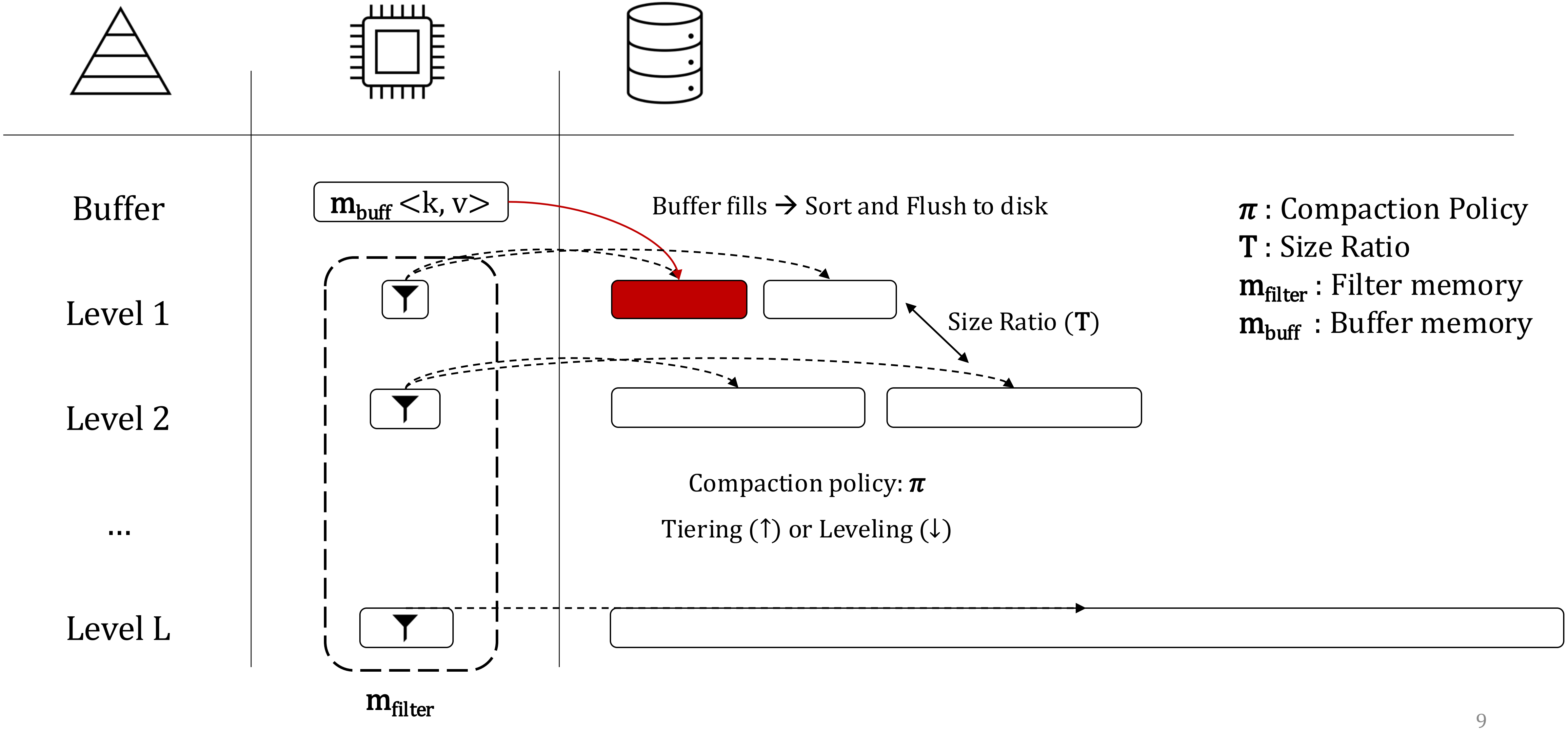


Writes : w



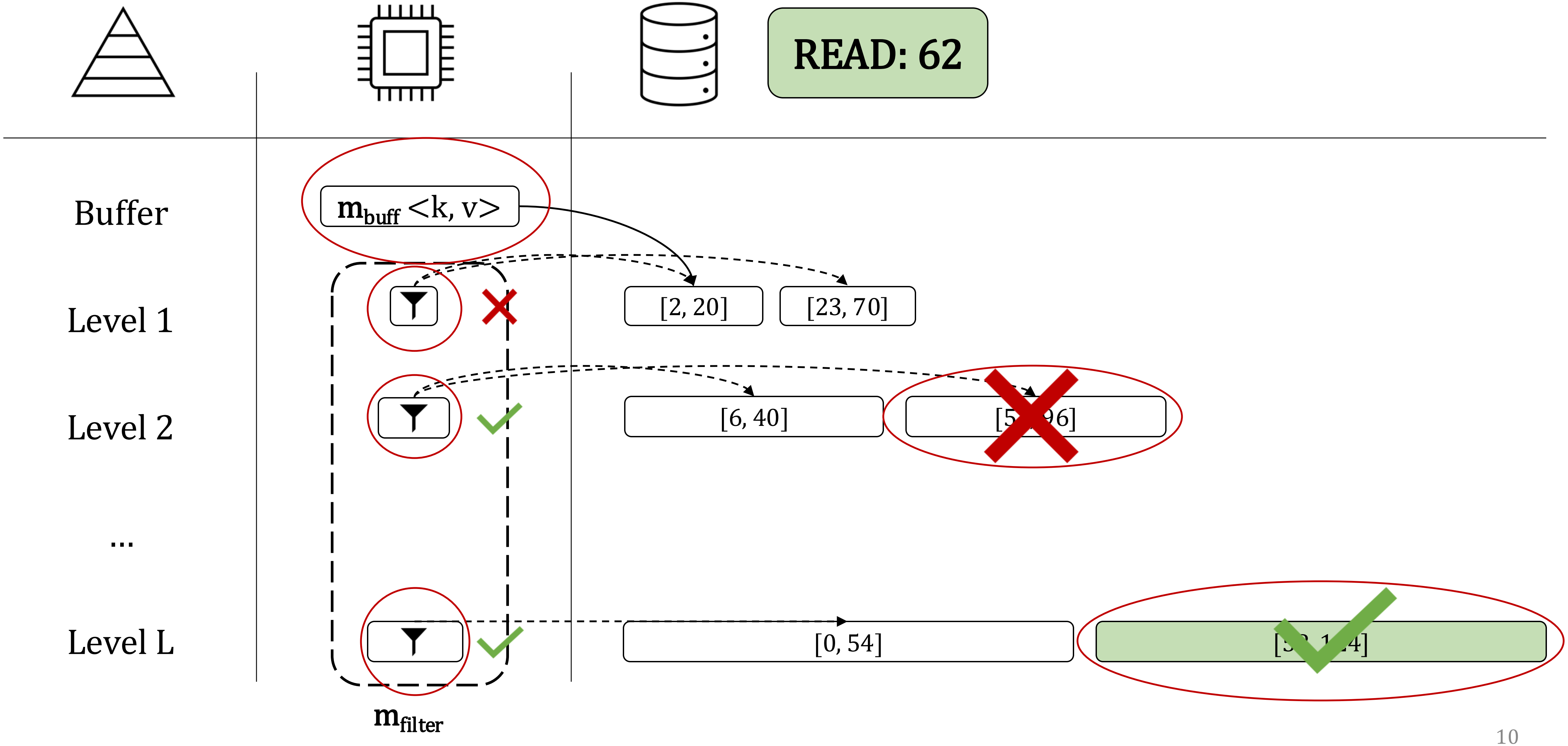
Cool! How do we go about tuning?

LSM Trees



π : Compaction Policy
T : Size Ratio
m_{filter} : Filter memory
m_{buff} : Buffer memory

LSM Trees – A Point Read



The LSM-Tuning Problem

$\boldsymbol{w}$: Workload (z_0, z_1, q, w)

Φ : LSM Tree Design $(m_{buff}, m_{filter}, T, \pi)$

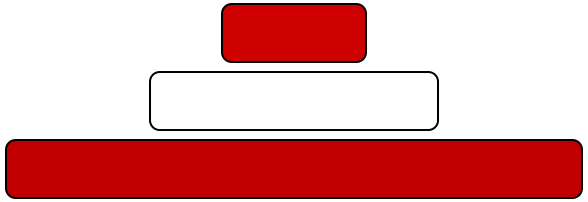
C : Cost

$$\Phi^* = argmin_{\Phi}\, C(\boldsymbol{w}, \Phi)$$

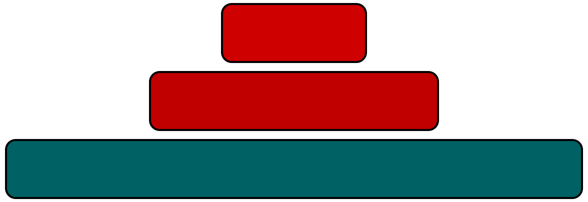
Workload $\boldsymbol{w}$ → Minimizer! → DB Config Φ

Point Reads

Empty Reads : z_0



Non-Empty Reads : z_1



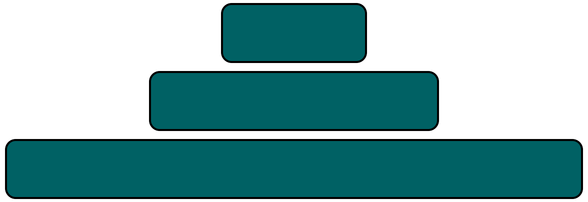
$$Z_0(\Phi) = \overbrace{\sum_{i=1}^{L} f_i}^{\text{Sum of false positives}}$$

$$Z_1(\Phi) = \overbrace{\sum_{i=1}^{L} \frac{T^{i-1} \cdot (T-1)}{N_f(T)} \cdot \frac{m_{buf}}{E}}^{\text{Probability query is satisfied at level i}} \overbrace{\left(1 + \sum_{j=1}^{i-1} f_j\right)}^{\text{False positives from levels above}}$$

[1] *Niv Dayan, Manos Athanassoulis, and Stratos Idreos. 2017. Monkey: Optimal Navigable Key-Value Store. In Proceedings of the 2017 ACM International Conference on Management of Data (SIGMOD '17).*

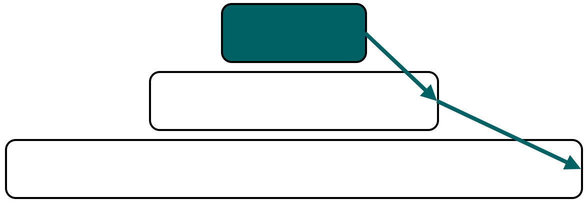
Range-Reads and Writes

Range Reads: q



$$Q(\Phi) = \overbrace{S_{RQ} \cdot \frac{N}{B}}^{\text{Sequential read based on selectivity}} + \left. L \right\} \text{1 I/O per Seek per level}$$

Writes : w



$$W(\Phi) = \overbrace{\underbrace{\frac{L}{B}}_{\text{Writes only flush once buffer is full}} \cdot \frac{T-1}{2} \cdot (1 + A_{rw})}^{\text{Average number of merges a write will participate in}}$$

[1] *Niv Dayan, Manos Athanassoulis, and Stratos Idreos. 2017. Monkey: Optimal Navigable Key-Value Store. In Proceedings of the 2017 ACM International Conference on Management of Data (SIGMOD '17).*

The LSM-Tuning Problem

$\boldsymbol{w}$: Workload (z_0, z_1, q, w)

Φ : LSM Tree Design $(m_{buff}, m_{filter}, T, \pi)$

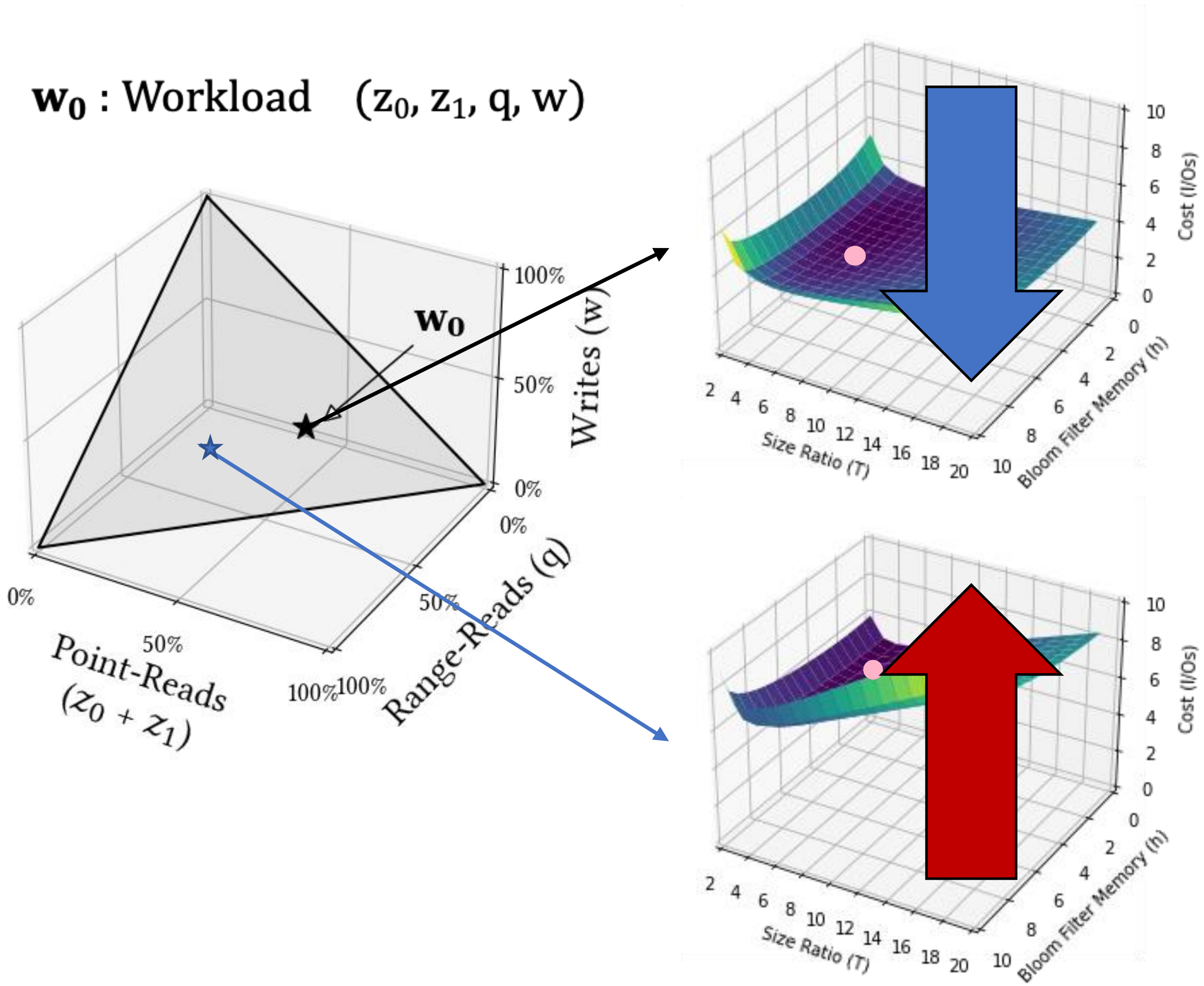
C : Cost (I/O)

$$\Phi^* = argmin_{\Phi}\, C(\boldsymbol{w}, \Phi)$$

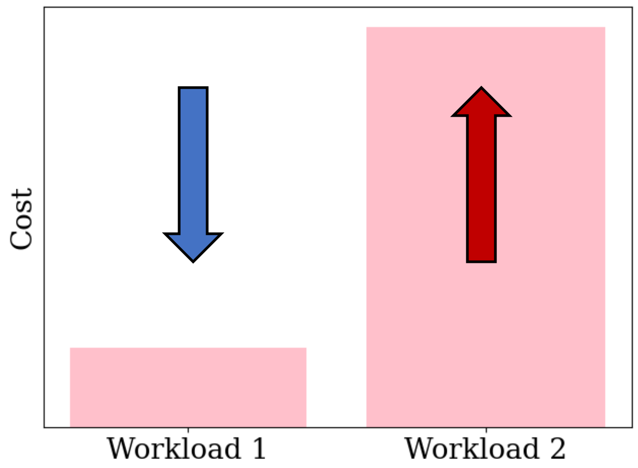
Define our cost function

$$C(\hat{\mathbf{w}}, \Phi) = \hat{\mathbf{w}}^{\intercal}\mathbf{c}(\Phi) = z_0 \cdot Z_0(\Phi) + z_1 \cdot Z_1(\Phi) + q \cdot Q(\Phi) + w \cdot W(\Phi)$$

Tuning Problems



'Best' Configuration



Optimal tuning depends on workload

Workload uncertainty leads to **<u>sub-optimal</u>** tuning

Outline

Introduction

LSM Trees Notation

Nominally Tuning LSM Trees

ENDURE: Robustly Tuning LSM Trees

The ENDURE Pipeline

ENDURE Evaluation

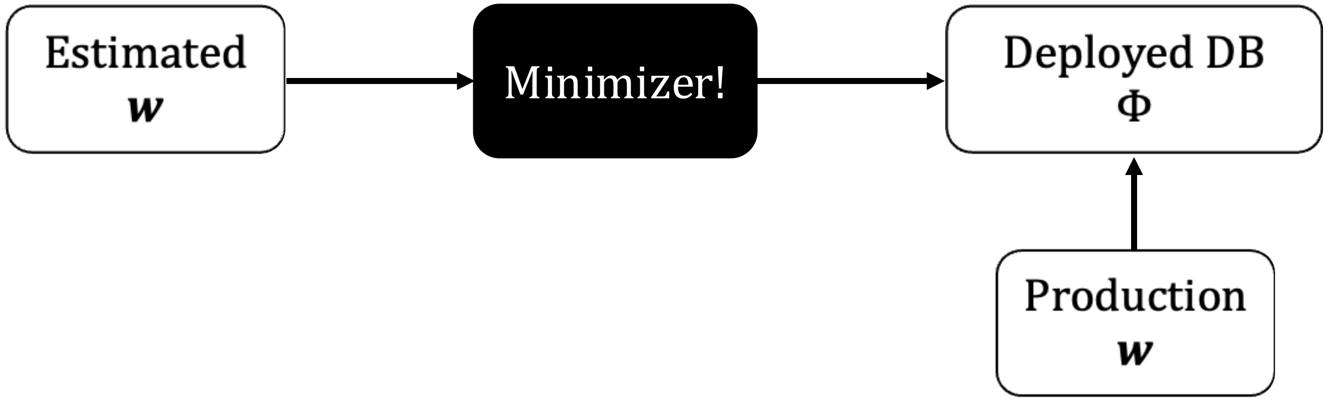
We've Got a Problem…

$\mathbf{w}$: Workload (z_0, z_1, q, w)

Φ : LSM Tree Design $(m_{buff}, m_{filter}, T, \pi)$

C : Cost (I/O)

$$\Phi^* = argmin_{\Phi}\, C(\mathbf{w}, \Phi)$$



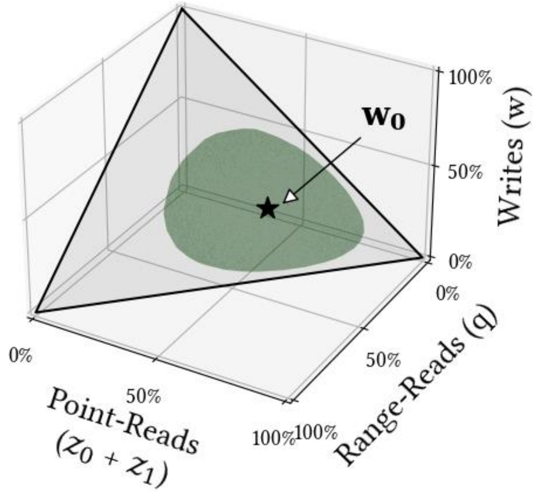
"It works on my machine!!!"

The LSM-Tuning Problem

w : Workload (z_0, z_1, q, w)

Φ : LSM Tree Design $(m_{buff}, m_{filter}, T, \pi)$

C : Cost (I/O)



$$\Phi^* = argmin_{\Phi}\, C(\mathbf{w}, \Phi)$$

Nominal

U_{w}^{ρ}: Uncertainty Neighborhood of Workloads

ρ : Size of this neighborhood

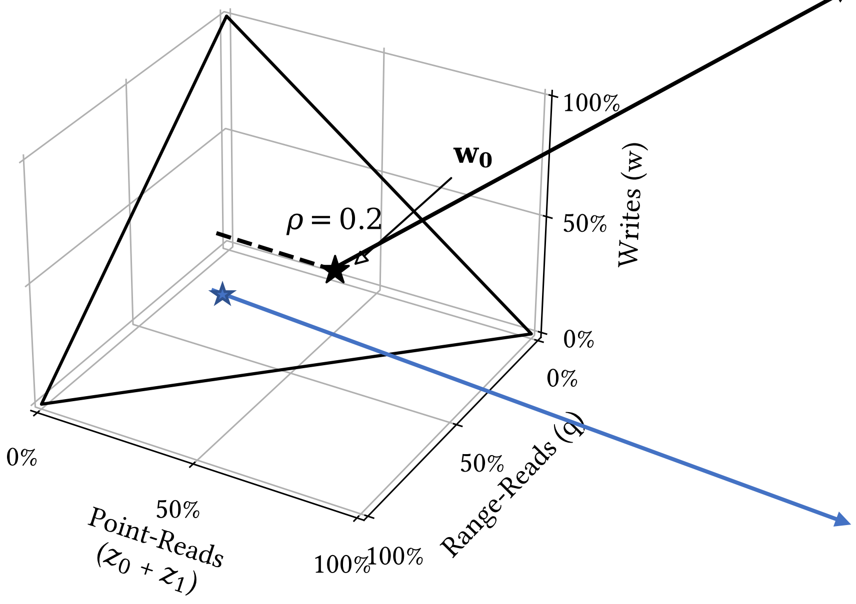
Robust

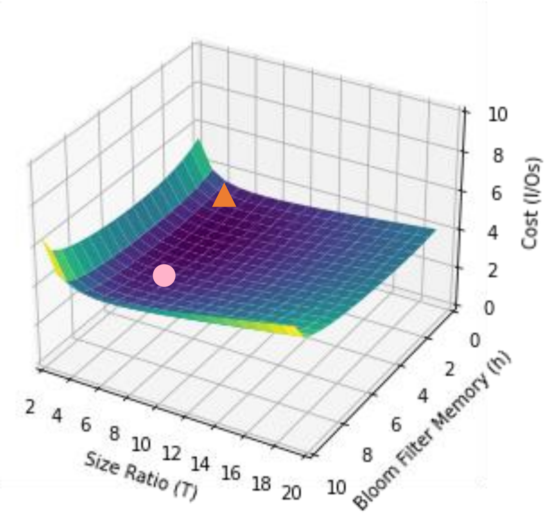
$$\Phi^* = \mathrm{argmin}_{\Phi}\, C(\hat{\mathbf{w}}, \Phi)$$

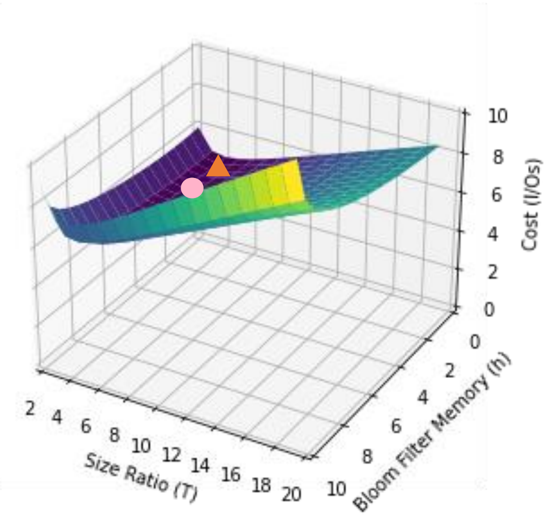
$$s.t., \qquad \hat{\mathbf{w}} \in U_{\mathbf{w}}^{\rho}$$

Robust Tuning

$\mathbf{w_0}$: Workload (z_0, z_1, q, w)





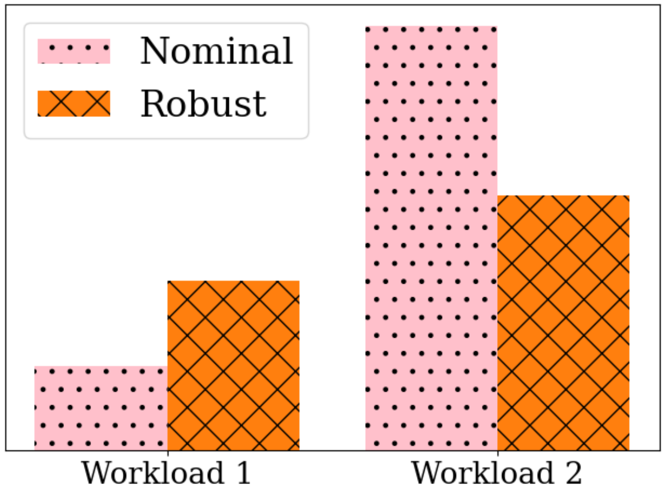


$$\Phi^* = \text{argmin}_\Phi \; C(\hat{\boldsymbol{w}}, \Phi)$$

$$s.t., \qquad \hat{\boldsymbol{w}} \in U_w^\rho$$

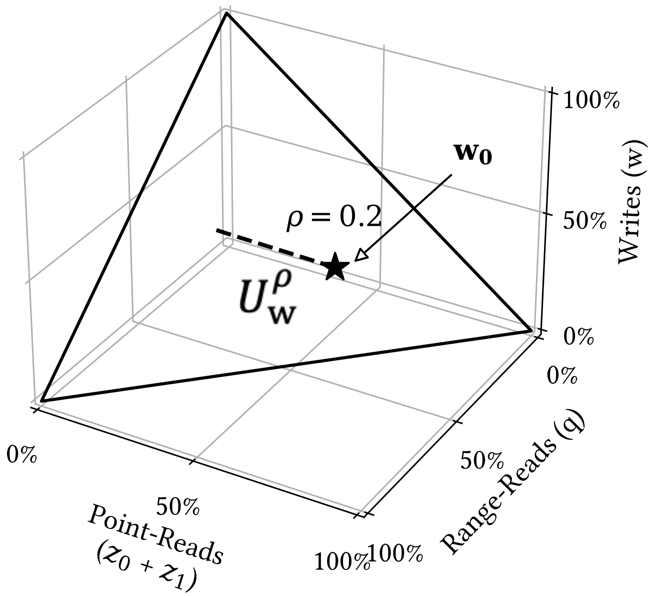
Optimal configuration for expected workload

Robust configuration for the workload neighborhood



Uncertainty Neighborhood

Workload Characteristic

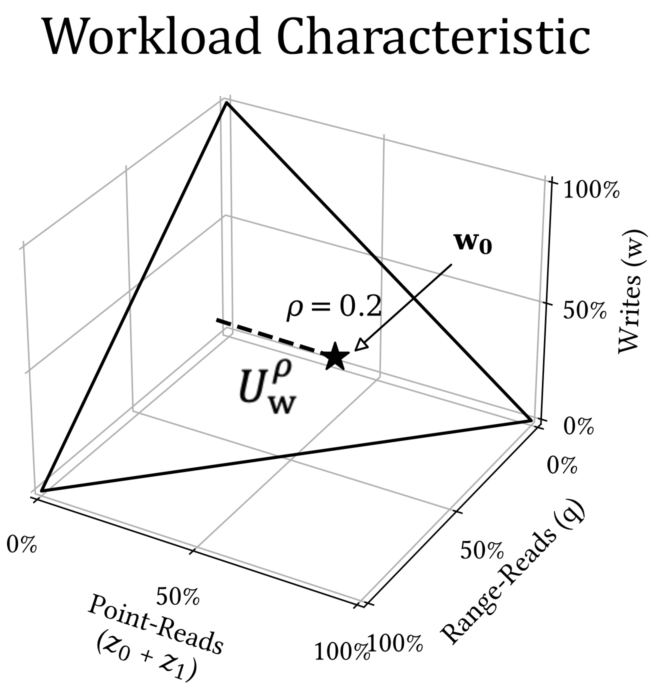


Neighborhood of workloads (ρ) via the KL-divergence

$$I_{KL}(\widehat{w}, w) = \sum_{i=1}^{m} \widehat{w}_i \cdot \log\left(\frac{\widehat{w}_i}{w_i}\right)$$

U_{w}^{ρ}: Uncertainty Neighborhood of Workloads
ρ : Size of this neighborhood

Calculating Neighborhood Size



Historical workloads
maximum/average uncertainty among workload pairings

User provided workload uncertainty

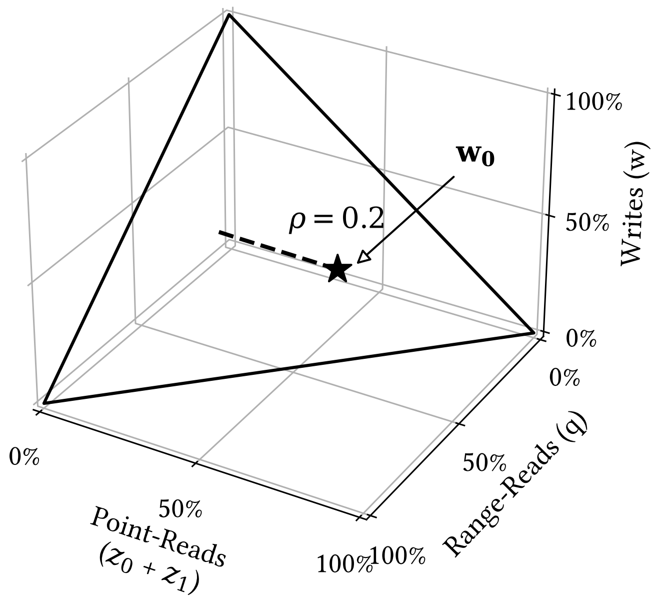
U_{w}^{ρ}: Uncertainty Neighborhood of Workloads
ρ : Size of this neighborhood

Solving Robust Problem

Iterating over every possible workload is expensive

$$\Phi_R = \arg\min_{\Phi} \ \hat{\mathbf{w}}^\intercal \mathbf{c}(\Phi)$$

$$\text{s.t.} \quad \hat{\mathbf{w}} \in \mathcal{U}^{\rho}_{\mathbf{w}}$$



Solving Robust Problem

Iterating over every possible workload is expensive

$$\Phi_R = \arg\min_{\Phi} \ \hat{\mathbf{w}}^\intercal \mathbf{c}(\Phi)$$
$$\text{s.t.} \quad \hat{\mathbf{w}} \in \mathcal{U}_{\mathbf{w}}^{\rho}$$

$$\Downarrow$$

Rewrite as a min-max

$$\min_{\Phi} \max_{\hat{\mathbf{w}} \in \mathcal{U}_{\mathbf{w}}^{\rho}} \hat{\mathbf{w}}^\intercal \mathbf{c}(\Phi)$$

$$\Downarrow$$

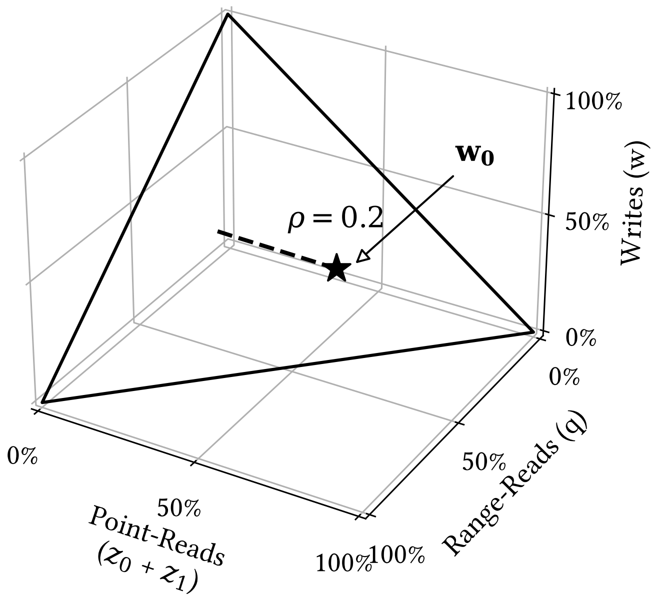
Find the dual of the maximization problem to reduce to a feasible problem [2]

$$\min_{\Phi, \lambda \geq 0, \eta} \left\{ \eta + \rho\lambda + \lambda \sum_{i=1}^{m} w_i \phi_{KL}^{*} \left(\frac{c_i(\Phi) - \eta}{\lambda} \right) \right\}$$

[2] Aharon Ben-Tal, Dick den Hertog, Anja De Waegenaere, Bertrand Melenberg, and Gijs Rennen. 2013. Robust Solutions of Optimization Problems Affected by Uncertain Probabilities.

ENDURE Pipeline

Workload Characteristic



System Information

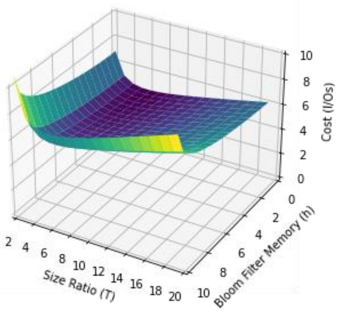
Page Size

Memory Budget

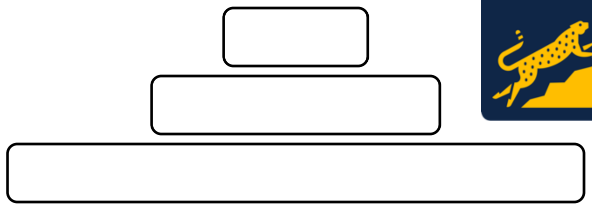
ENDURE

Solves the Robust Problem

Expected performance



RocksDB Configuration

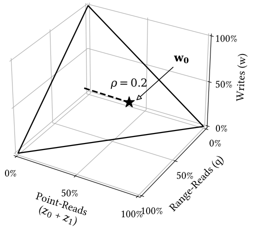


Testing Suite

ENDURE in Python, implemented in tandem with RocksDB

Uncertainty benchmark

- 15 expected workloads
- 10K randomly sampled workloads as a test-set



Index	(z_0, z_1, q, w)				Type
0	25%	25%	25%	25%	**Uniform**
1	97%	1%	1%	1%	**Unimodal**
2	1%	97%	1%	1%	
3	1%	1%	97%	1%	
4	1%	1%	1%	97%	
5	49%	49%	1%	1%	**Bimodal**
6	49%	1%	49%	1%	
7	49%	1%	1%	49%	
8	1%	49%	49%	1%	
9	1%	49%	1%	49%	
10	1%	1%	49%	49%	
11	33%	33%	33%	1%	**Trimodal**
12	33%	33%	1%	33%	
13	33%	1%	33%	33%	
14	1%	33%	33%	33%	

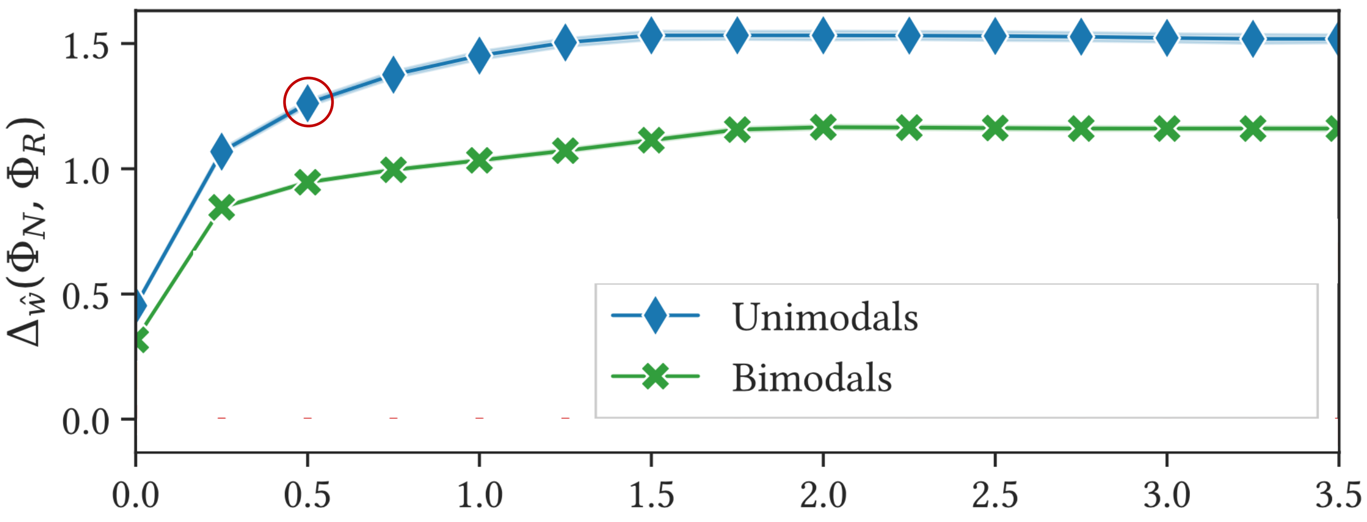
Normalized delta throughput

$$\Delta_{\mathbf{w}}(\Phi_1, \Phi_2) = \frac{1/C(\mathbf{w},\Phi_2) - 1/C(\mathbf{w},\Phi_1)}{1/C(\mathbf{w},\Phi_1)}$$

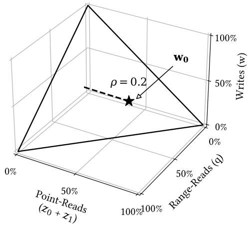
Nominal vs Robust: > 0 is better

1 means 2x speedup

Impact of Workload Type



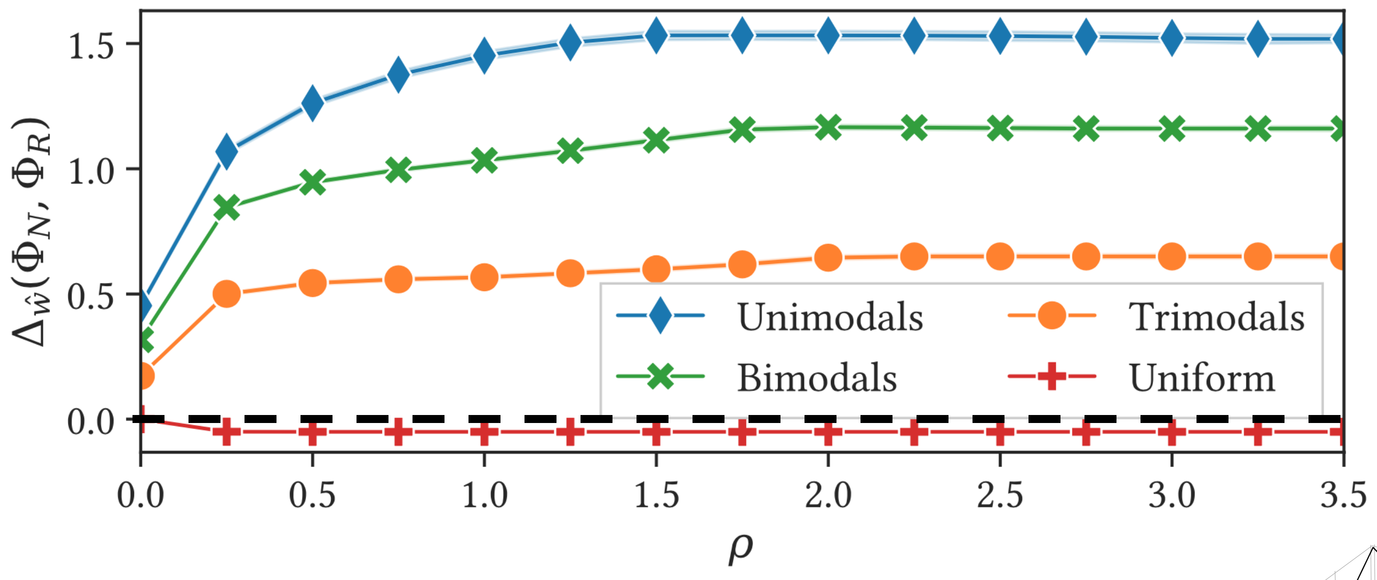
ρ ⟵ Given to ENDURE



Index	(z_0, z_1, q, w)				Type
0	25%	25%	25%	25%	**Uniform**
1	97%	1%	1%	1%	**Unimodal**
2	1%	97%	1%	1%	
3	1%	1%	97%	1%	
4	1%	1%	1%	97%	
5	49%	49%	1%	1%	**Bimodal**
6	49%	1%	49%	1%	
7	49%	1%	1%	49%	
8	1%	49%	49%	1%	
9	1%	49%	1%	49%	
10	1%	1%	49%	49%	
11	33%	33%	33%	1%	**Trimodal**
12	33%	33%	1%	33%	
13	33%	1%	33%	33%	
14	1%	33%	33%	33%	

Unbalanced workloads result in overfitted nominal tunings

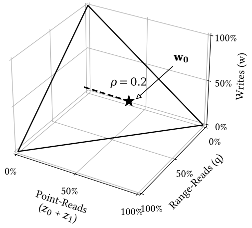
Impact of Workload Type



Index	(z_0, z_1, q, w)				Type
0	25%	25%	25%	25%	**Uniform**
1	97%	1%	1%	1%	**Unimodal**
2	1%	97%	1%	1%	
3	1%	1%	97%	1%	
4	1%	1%	1%	97%	
5	49%	49%	1%	1%	**Bimodal**
6	49%	1%	49%	1%	
7	49%	1%	1%	49%	
8	1%	49%	49%	1%	
9	1%	49%	1%	49%	
10	1%	1%	49%	49%	
11	33%	33%	33%	1%	**Trimodal**
12	33%	33%	1%	33%	
13	33%	1%	33%	33%	
14	1%	33%	33%	33%	

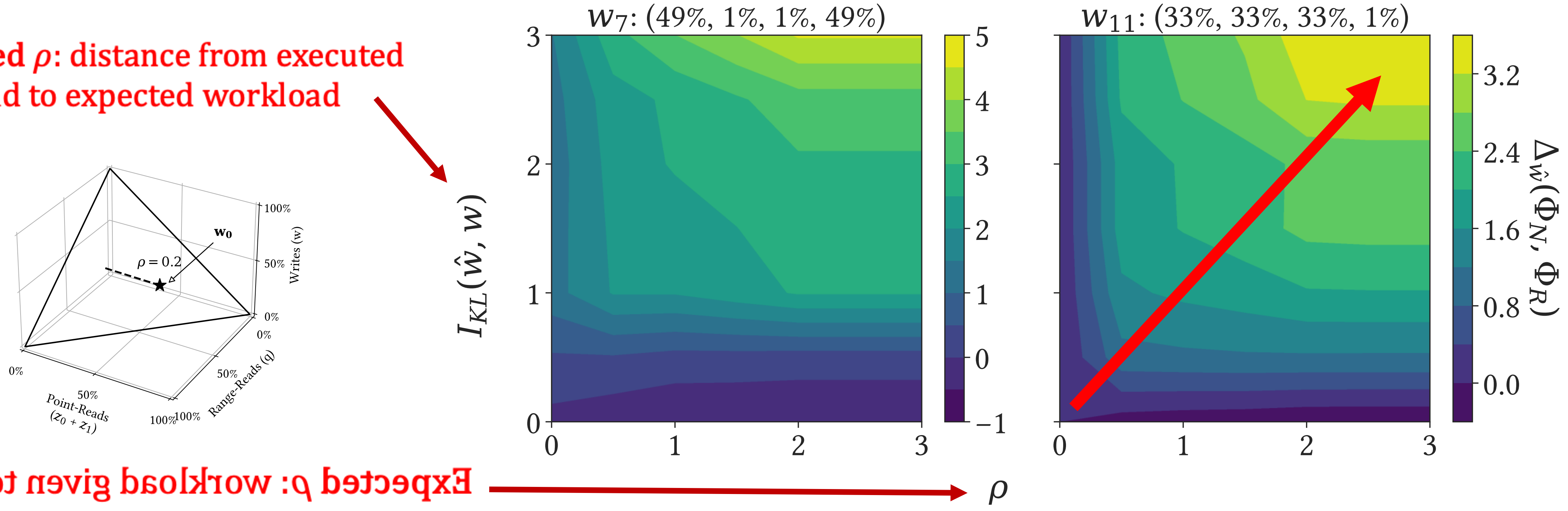
<u>Unbalanced</u> workloads result in overfitted nominal tunings

Tuning with uncertainty ($\rho > 0.5$) provides benefits



Relationship of Expected and Observed ρ

Observed ρ: distance from executed workload to expected workload

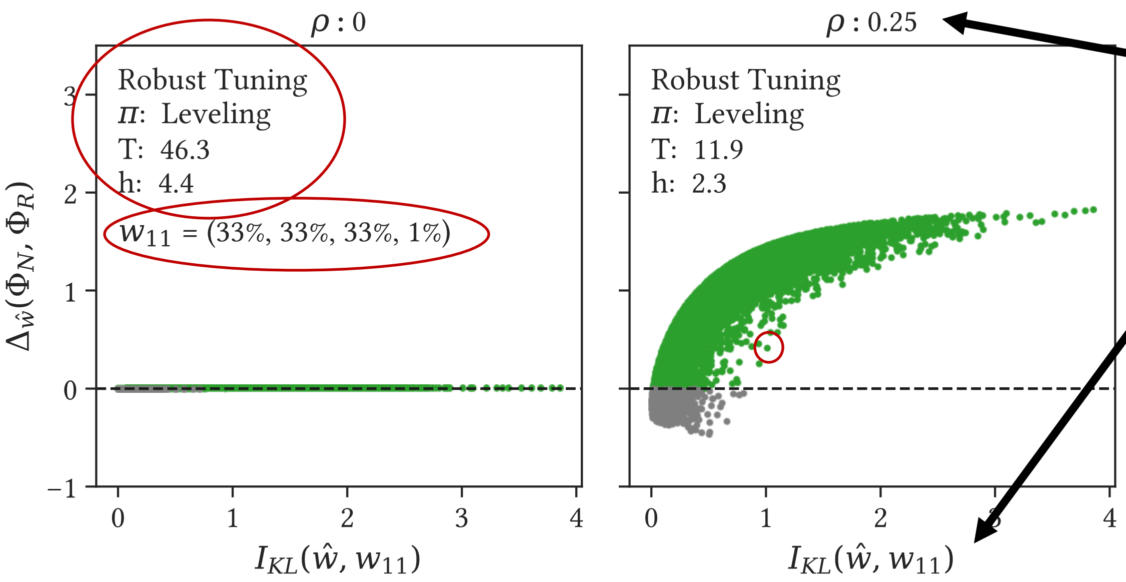


Expected ρ: workload given to tuner

Highest throughput when observed and expected ρ match

Lowest throughput when ρ is mismatched

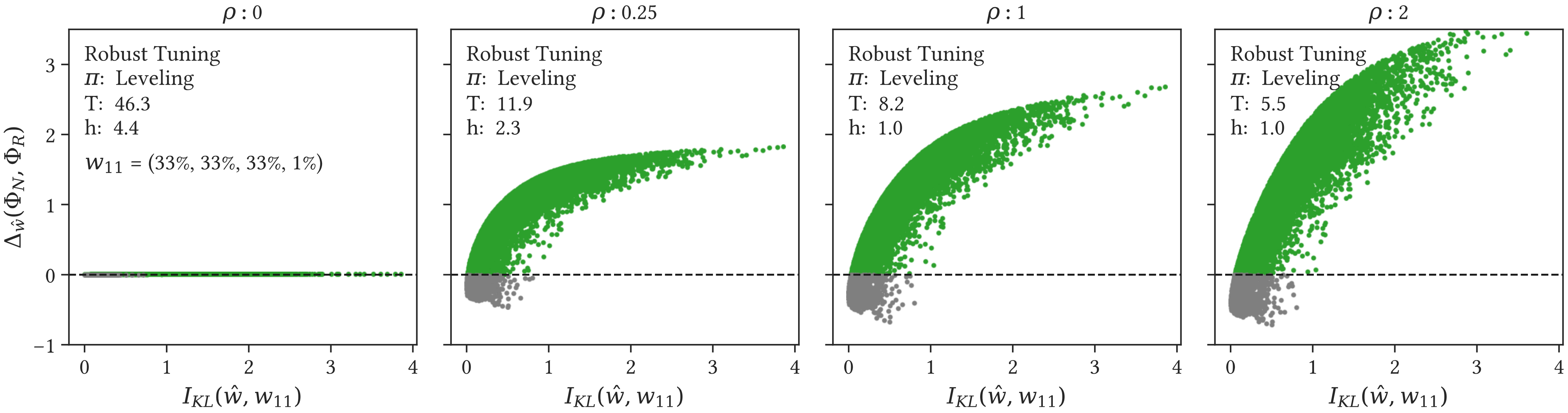
Impact of Observed vs Expected ρ



Expected ρ: given to tuner

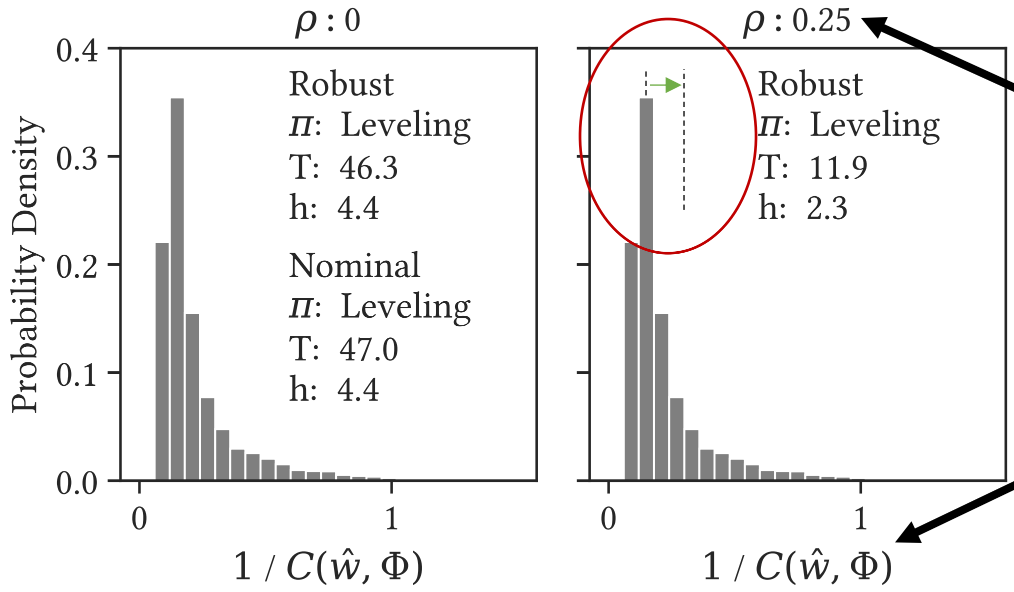
Observed ρ: distance from executed workload to expected workload

Impact of Observed vs Expected ρ



- Higher expected ρ accounts for more uncertainty
- Potential speed up of 4x
- Higher expected ρ → anticipates writes → shallow tree

ρ and Performance Gain Distribution

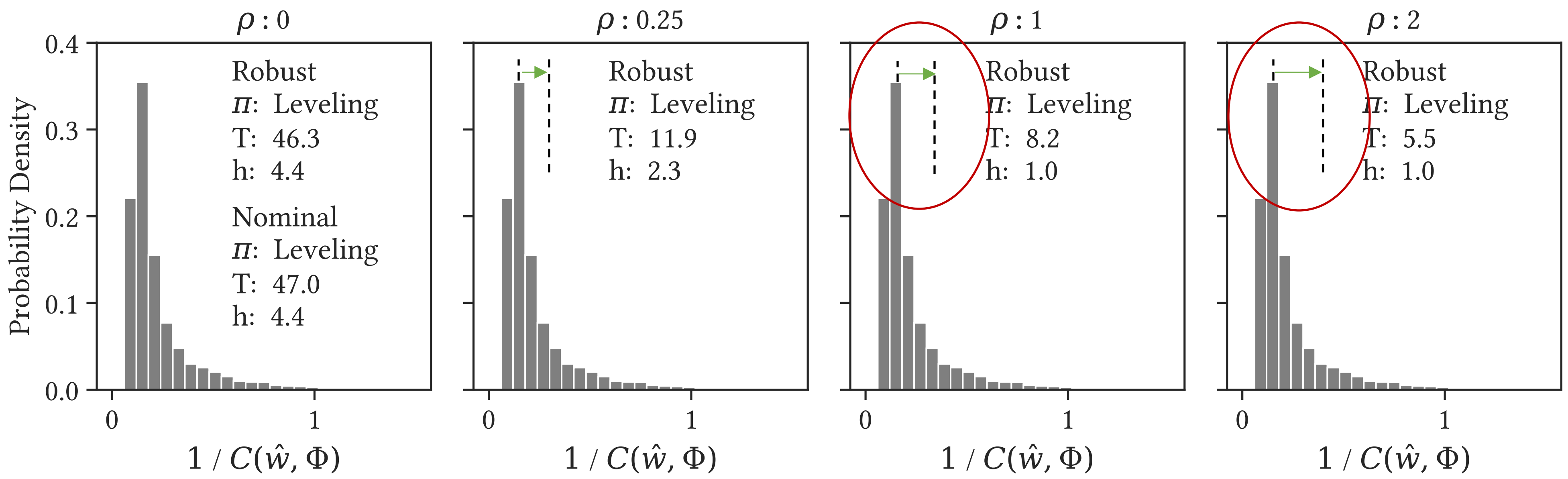


Expected ρ: given to tuner

Throughput

$w_{11} = (33\%, 33\%, 33\%, 1\%)$

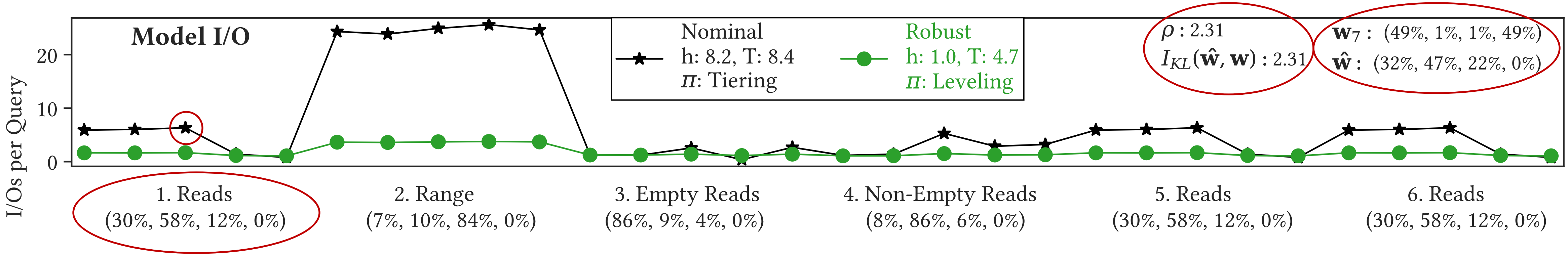
ρ and Performance Gain Distribution



$$w_{11} = (33\% , 33\%, 33\%, 1\%)$$

Peak of the distribution moves towards higher throughput as we consider higher uncertainty

Workload Sequence on RocksDB

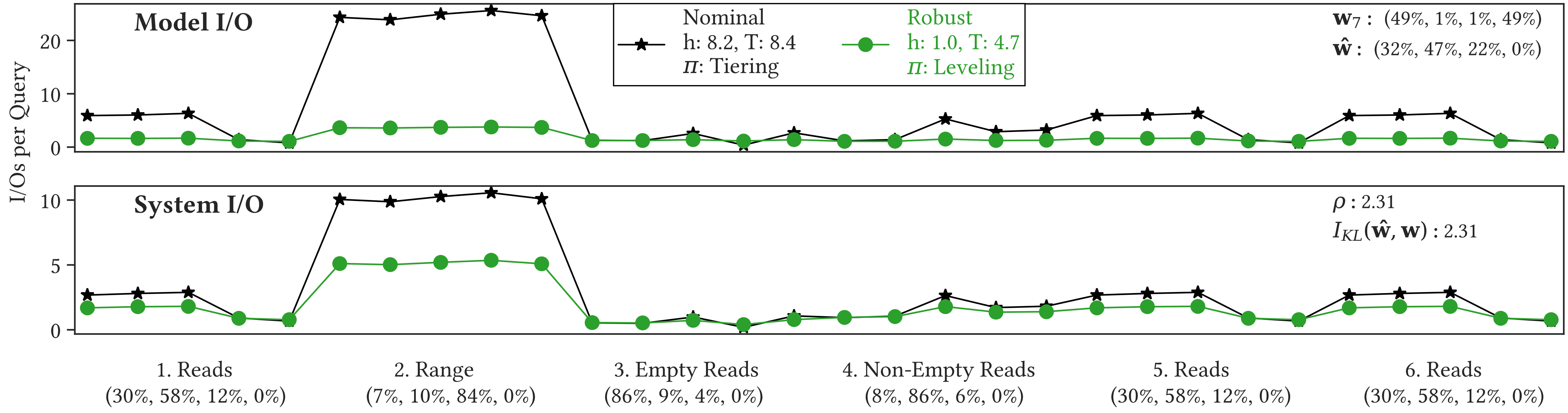


RocksDB instance setup with 10 million unique key-value pairs of size 1KB

Each observation period is 200K queries, with 5 observations per session 6 million queries to the DB

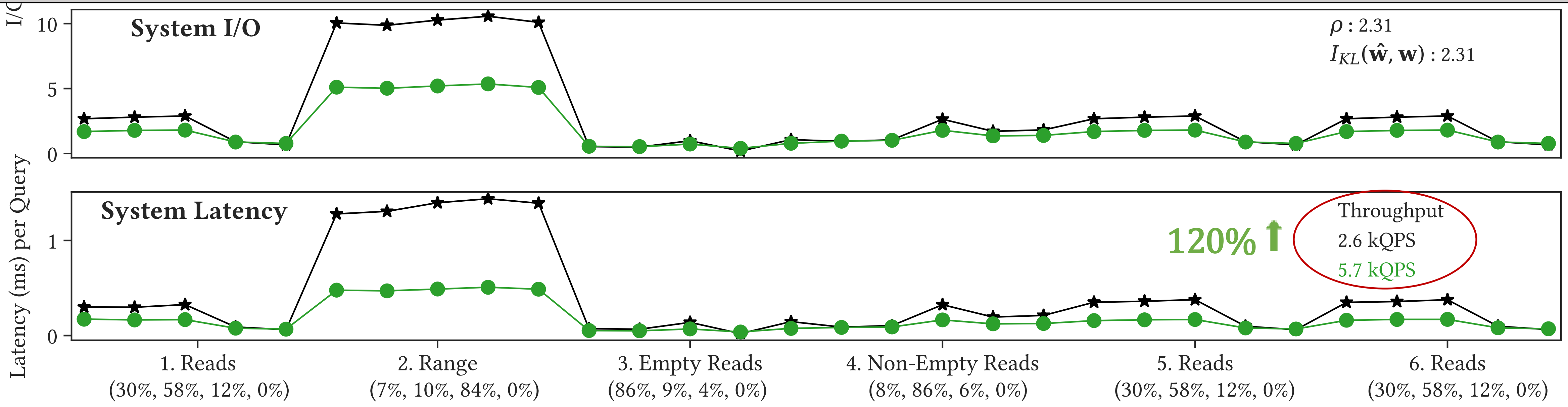
Writes are unique, range queries average 1-2 pages per level

Workload Sequence



Workload Sequence

Small subset of results! Take a look at the paper for a more detailed analysis

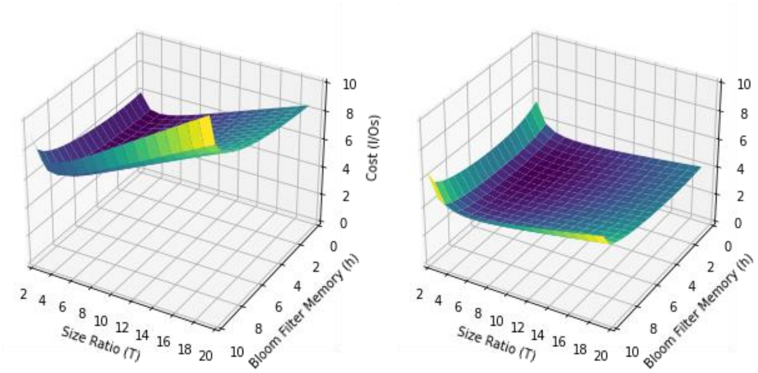


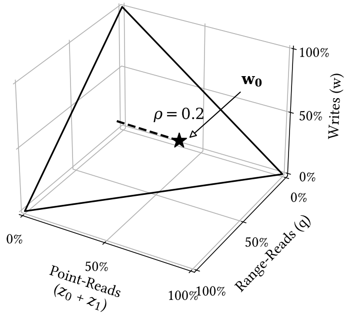
Thanks!

Workload uncertainty creates suboptimal tunings

ENDURE: robust tuning using neighborhood of workloads

Deployed ENDURE on RocksDB





disc.bu.edu/

www.ndhuynh.com/

@nd_huynh