MUTANT: Balancing Storage Cost and Performance in LSM-Tree Data Stores

Hobin Yoon¹, Juncheng Yang²
Sveinn Kristjansson³, Steinn Sigurdarson⁴
Ymir Vigfusson²,⁵, Ada Gavrilovska¹

¹Georgia Institute of Technology, ²Emory University
³Spotify, ⁴Takumi, ⁵Reykjavik University
Why Dave, a Database Engineer, Quit

Hey Dave, our DB costs $30 M/year. Can you make it less expensive?

No problem, Carol!

• Find a new storage type
• Live data migration: backup, replicate new data, validate data, migrate applications. Could take months [Netflix].

(After 2 months)

Here is a new database. It’s a bit slower, but costs only $20 M!

Dave, the budget is getting tighter. Can you make it $10 M?

(After 2 months)

Here is a $10 M database. I was lucky to find a right storage device for the budget.

Actually, it’s too slow now. Can you make it a bit faster? I fired 5 people and we have more budget now.

Still there?
Seamless Cost-Performance Trade-offs

Wouldn’t it be nice if
- You can get any cost-performance trade-off?
- DB does migrations by itself?

**Mutant**, a database storage layer with seamless cost-performance trade-offs!
Problem Formulation

Organize DB storage blocks into fast, expensive storage, and slow, inexpensive storage.

With **cost** constraint:

“I’d like to pay no more than $0.03 /GB/month, while keeping the latency minimum.”

With **latency** constraint:

“I’d like the latency no higher than 40 ms, while keeping the cost minimum.”
NoSQL DBs

- LSM (Log-Structured Merge) tree

  - Write a record
  - Read a record

  - Commit log
  - MemTable
  - SSTable
  - Merge
  - SSTable

- Read optimization

  - Key
  - Keyspace

  - L0: 71
  - L1: 60, 64
  - L2: 50, 51, 52, 35, 36, 37, 40

  - O(log n)

  - 10x more SSTables
Organizing SSTables ...

Web workloads have a strong temporal locality.
Problem Formulation

Constraint
I’d like to pay no more than $0.03 / GB/month,

Optimization goal
while keeping the latency minimum

I’d like to keep the total SSTable size in the fast storage no more than 50 GB,

while maximizing the SSTable accesses in the fast storage

Hard to formulate:
• No storage latency model
• Parallel accesses

😊
SSTable Organization

• “Store more frequently accessed SSTables into the fast storage of a limited size.”

• **0/1 Knapsack** problem!
  • $O(nW)$ time and space with dynamic programming
    • with $n$ SSTables and a $W$-byte storage

• Greedy algorithm!
  • Using SSTable access freq / size
  • Faster: $O(n)$
    • Almost optimal! The item sizes are a lot smaller than $W$ (64 MB or 160 MB vs. TBs)

• Now, how do you migrate SSTables between storages?
SSTable Migration

- Copy SSTable → Redirect reads
  → Delete old SSTable

- Use SSTable compaction!
- SSTable migration = Single SSTable compaction to a different storage
SSTable Compaction

Level n

Level n+1

Level n

Level n+1

SSTable compaction
SSTable Compaction

Output SSTable temperature
= Average of the input SSTable temperatures
System Architecture
Implementation

- Mutant in [RocksDB](https://rocksdb.org) with 658 lines of C++ code and 110 lines for the integration.

- Minimal API

  **Clients:**

  ```
  void Open(Options);
  void SetCost(target_cost);
  ```

  ```
  Options opt;
  opt.storages.Add(
    "mnt/local-ssd1/mu-rocks-stg", 0.528,
    "mnt/ebs-st1/mu-rocks-stg", 0.045);
  DB::Open(opt);
  DB::SetCost(0.2);
  ```

  **Database:**

  **SSTable temperature monitor**

  ```
  void Register(sstable);
  void Unregister(sstable);
  void Accessed(sstable);
  ```

  **SSTable migration**

  ```
  void SchedMigr();
  sstable PickSstToMigr();
  sstable GetTargetDev();
  ```
Evaluation

• Cost Adaptability?

• Cost-Performance Spectrum?

• System Overhead?
Evaluation Setup

- Fast storage: Local SSD (EC2 instance store). $0.528/GB/month
- Slow storage: Remote HDD (EBS Magnetic volume). $0.045

- Workloads: YCSB ”read latest” and QuizUp
Cost Adaptability

Fast: $0.528, Slow: $0.045

Time for SSTable temperature stabilization

Target cost ± ε

Storage cost ($/GB/month)

Target cost ($/GB/month)

Initial value

Changes

0.4

0.2

0.3

00:00

00:15

00:30

00:45

01:00

Time (HH:MM)
Latency
**Summary**

**Cost-performance trade-offs** in DBs were manual and limited in options.

**Mutant**: Automatic, seamless cost-performance trade-offs by (a) carefully monitoring SSTable temperatures and (b) organizing them into different storages.

Dave’s life made easy! 😊