DeuceDB: Smarter categorizing and scheduling to amortize background IO costs in LevelDB

Omkar Desai, Xiangqun Zhang, Ziyang Jiao

Oana Balmau, Deigo Didnona, Rachid Guerraoui, Willy Zwanenepoel, Huapeng Yuan, Aashray Arora, Karan Gupta, and Pavan Konka

Goal

Decrease background ops overhead to increase user throughput

Dynamically adapt key-value stores to changing workload characteristics in runtime

A recap - DeuceDB_v1 (Project 2)

- TRIAD, the paper we implemented, aims to decrease background work overhead in the presence of a skewed workload (80/20).
- TRIAD, in order to avoid unnecessary background I/O, does hot and cold key separation in the memtable and also delays compaction when there isn’t enough overlap to amortize the cost of the background compaction.
- These efforts help to reduce the overall compute spent in background I/O work and helps improve LevelDB performance.

Evaluations - DeuceDB_v1 (Project 2)

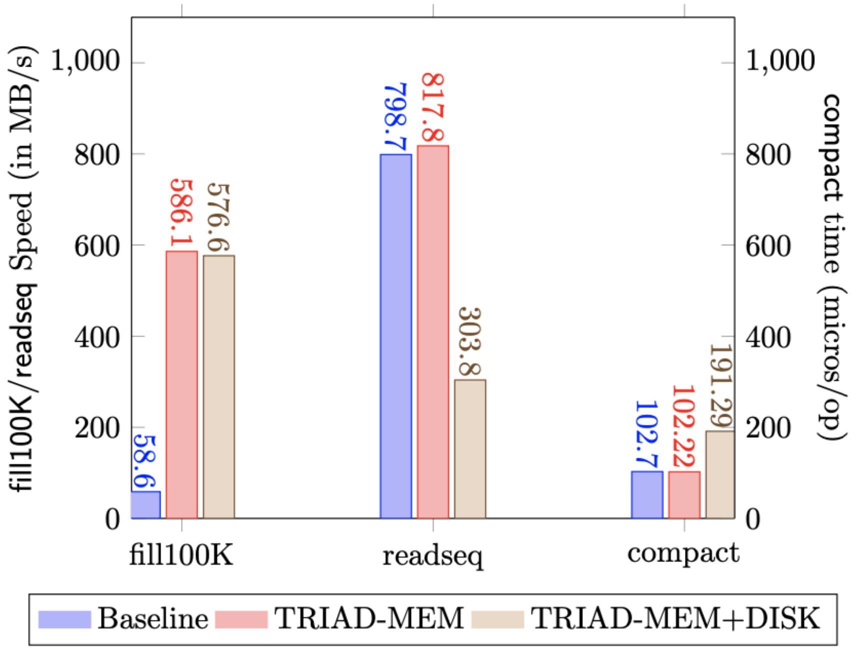


Figure 6: Evaluation on 1 million keys

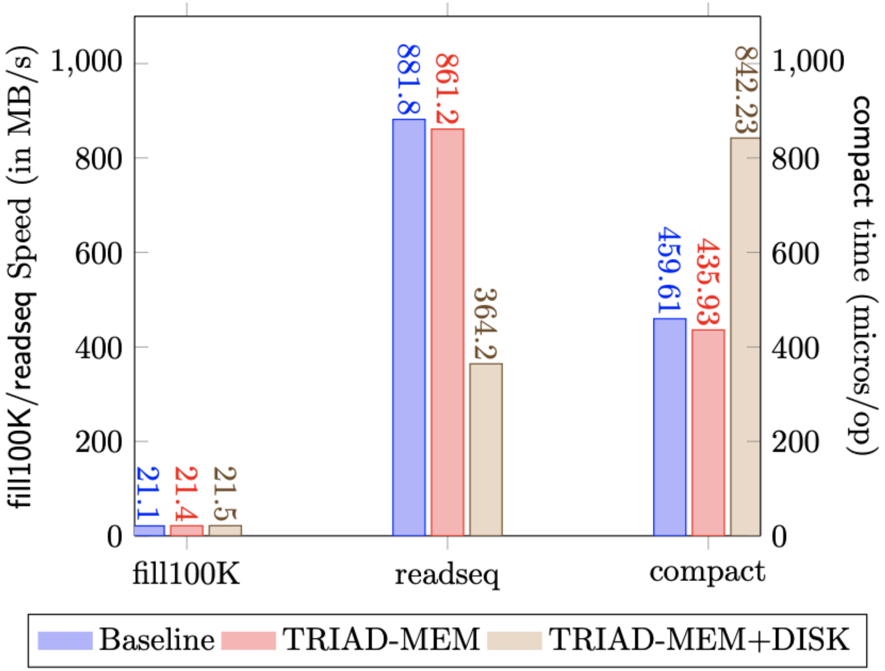


Figure 7: Evaluation on 5 million keys

What’s with the name?

- We implemented **2** algorithms to delay background work when it’s not worth it
- “Deuce” for **2**
- We wanted to sound cooler than we actually are

Problem statement

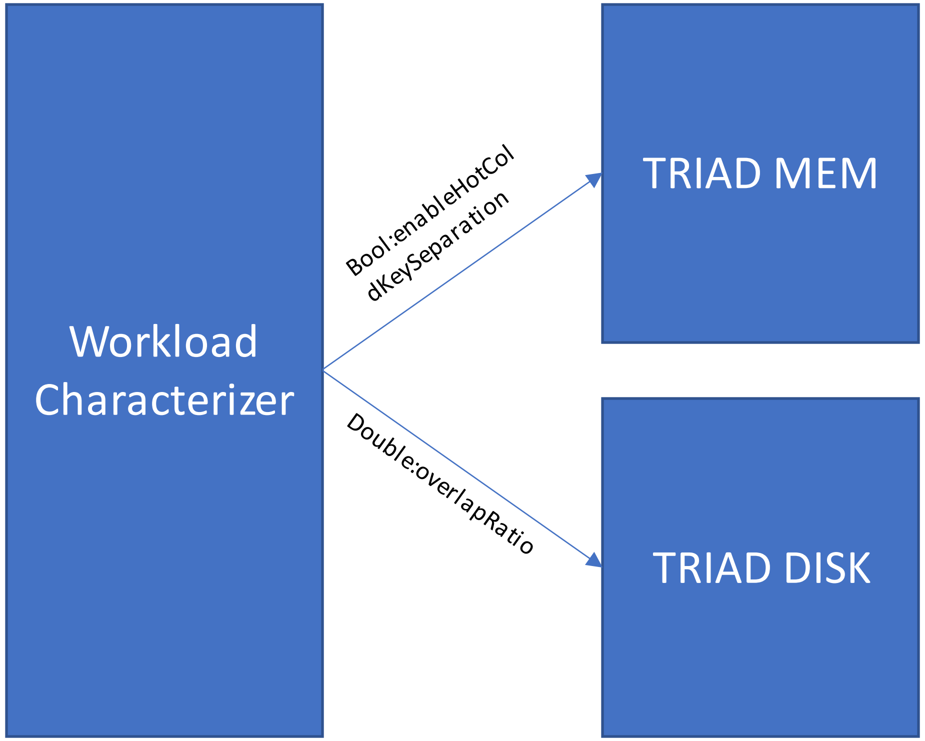
- Databases are usually built keeping in mind a specific type of workload.
- Databases based on LSM are suitable for very fast write throughput
- Databases based on b-tree like structures are built for faster reads.
- Databases, in production could be facing a wide variation of workloads including the worst case for the database in question.

Proposal

- The drawback of TRIAD is that the system is implemented to work well on a skewed workload only.
- In case of uniform workloads, or read only workloads, the performance suffers
- We argue that workload characteristics change over time and databases should be adaptable to the changing workload patterns.
- For this reason, we propose an improvised version of DeuceDB.

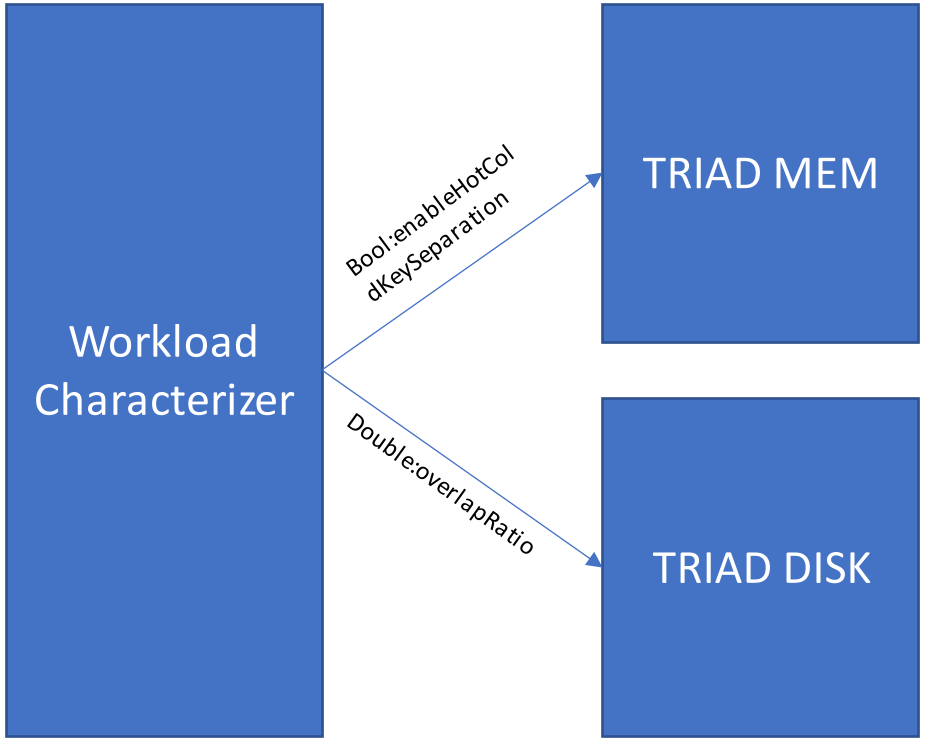
Workload Characterization

- We propose a solution to analyze the current workload condition and characterize it as:
  - Read Heavy
  - Write Heavy
  - Skewed
  - Uniform
- We implement a WorkloadCharacterizer as a new util class to understand the workload characteristic and dynamically adapt parameters of the database at runtime.



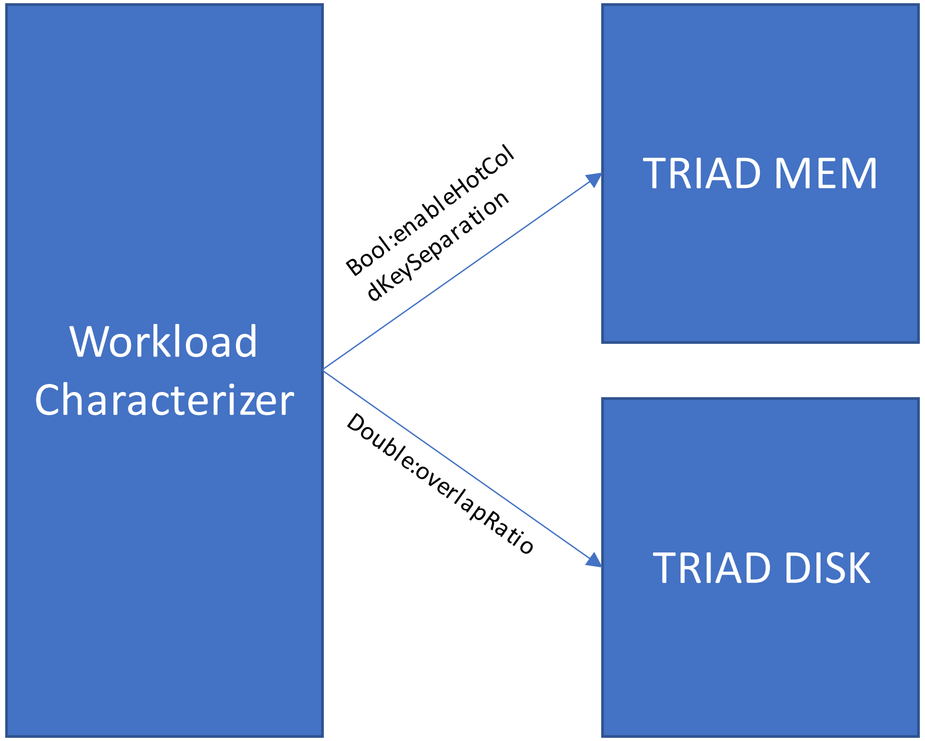
Workload Characterization

- We want to dynamically understand the read to write ratio being experienced by the database.
- For every Put and Get operation, we record the current get count and put count.
- We calculate the ratio in windows of 10k operations to ensure freshness



Workload Characterization

- We use this read to write ratio to modify the overlap ratio which was a hardcoded value in our previous implementation
- If the database is experiencing high reads, we reduce the overlap ratio to make sure we flush data from L0 to L1 faster to make reads faster.
- Else, we increase the overlap ratio threshold to reduce BG work.

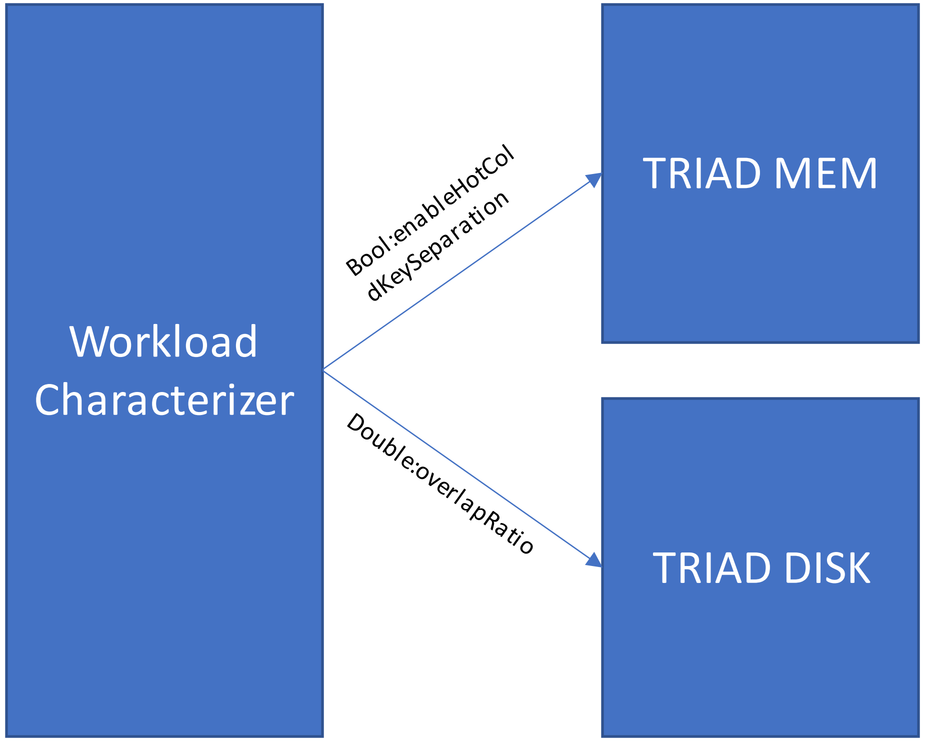


Workload Characterization

- TRIAD MEM improvements
  - For write heavy
    - If skewed

    => enable HotColdKeySeparation
    - If uniform

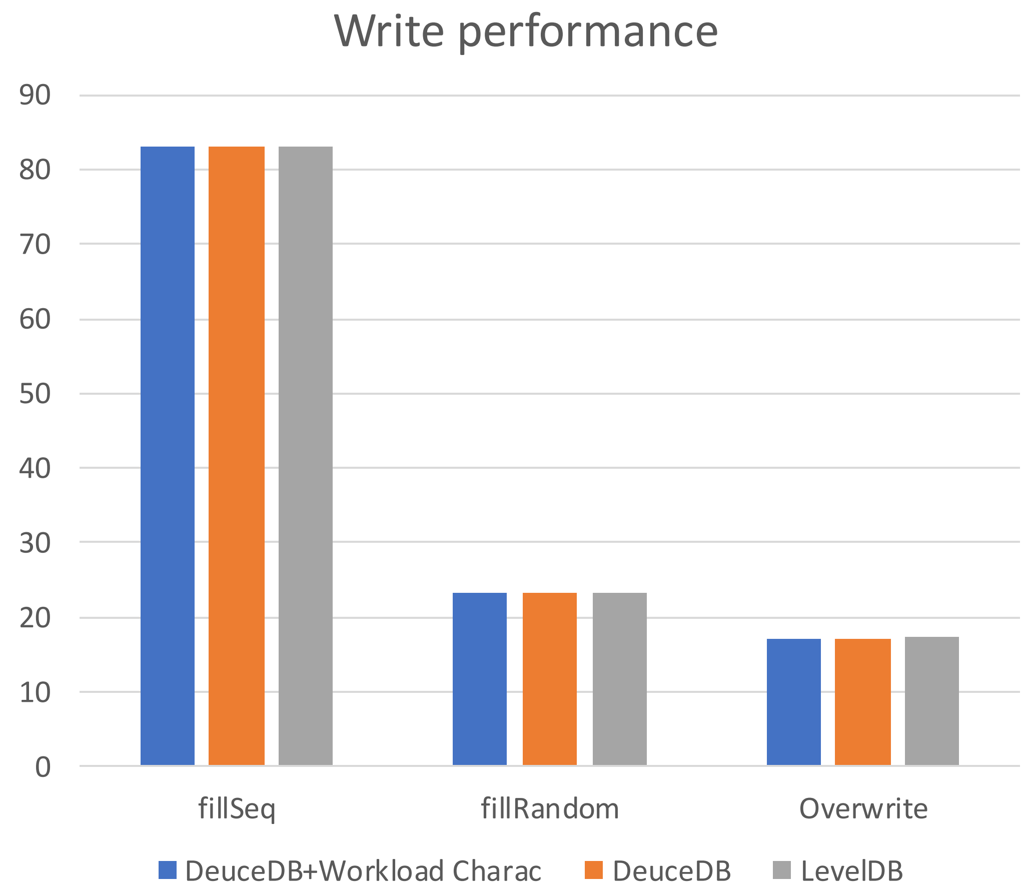
    => disable HotColdKeySeparation



Evaluations

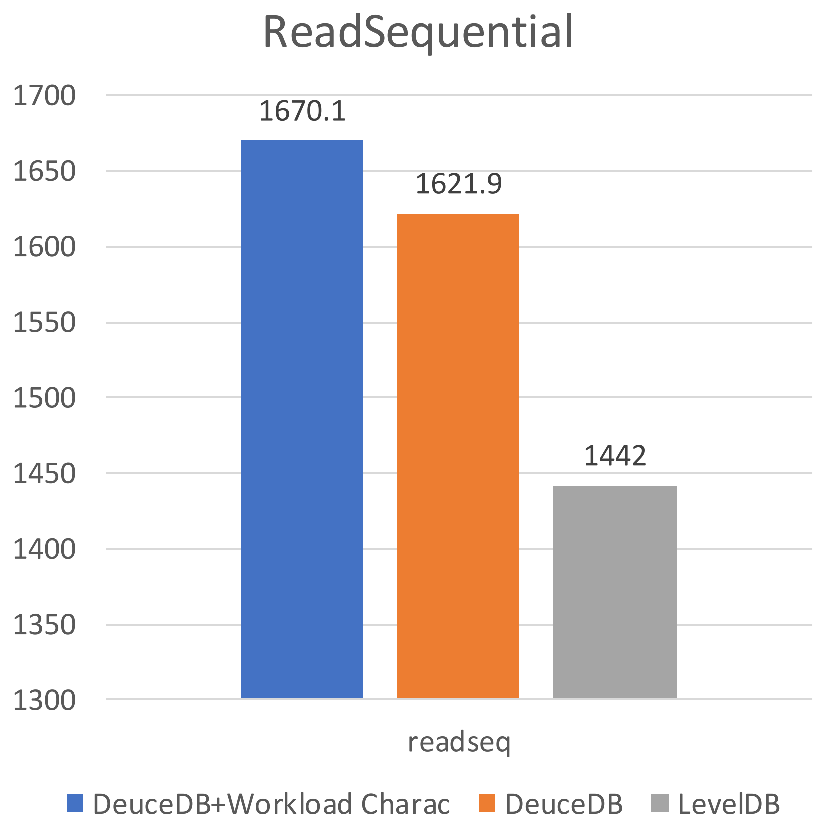
- Hardware
  - CPU: 8 * Intel(R) Core(TM) i7-9700 CPU @ 3.00GHz
  - CPUCache: 12288 KB
  - Keys: 16 bytes each
    Values: 100 bytes each
  - DRAM: 32GB
  - Storage: Samsung 840 evo 1TB M.2 SATA SSD

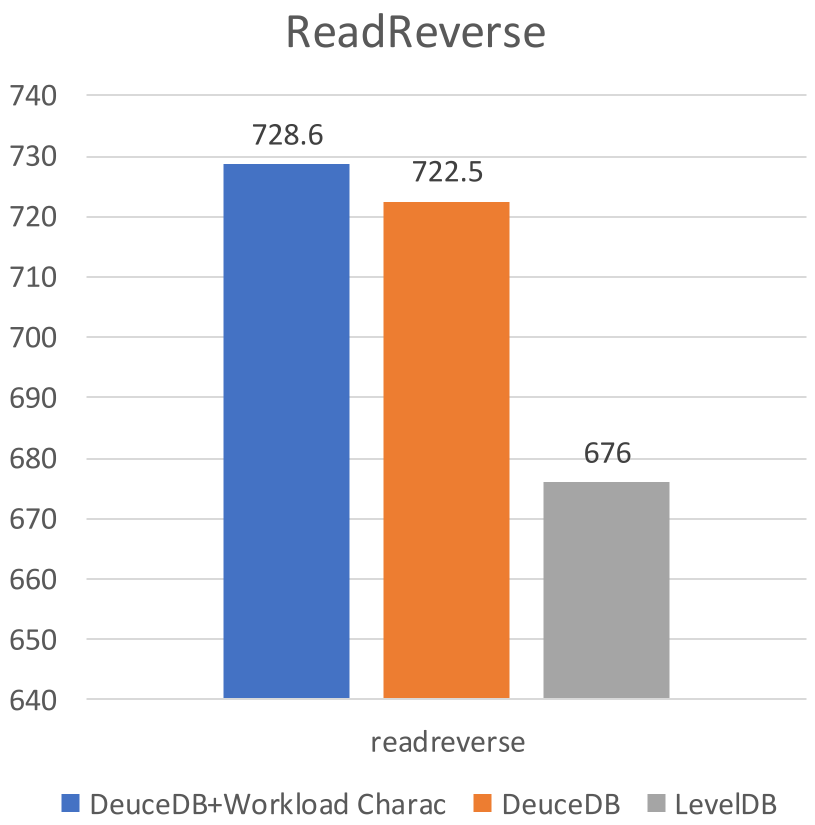
Evaluations



*Evaluations on 5M KV pairs

Evaluations





*Evaluations on 5M KV pairs

Evaluations

- ReadSeq:
- We were able to extract better read performance. An improvement of 3% over our DeuceDB implementation
- This is a 15% improvement over baseline LevelDB.

- Readreverse:
- We were able to extract better read performance. An improvement of 1% over our DeuceDB implementation
- This is an 8% improvement over baseline LevelDB.

- We were able to achieve this without having any impact on Write performance

Q & A

DeuceDB_v2