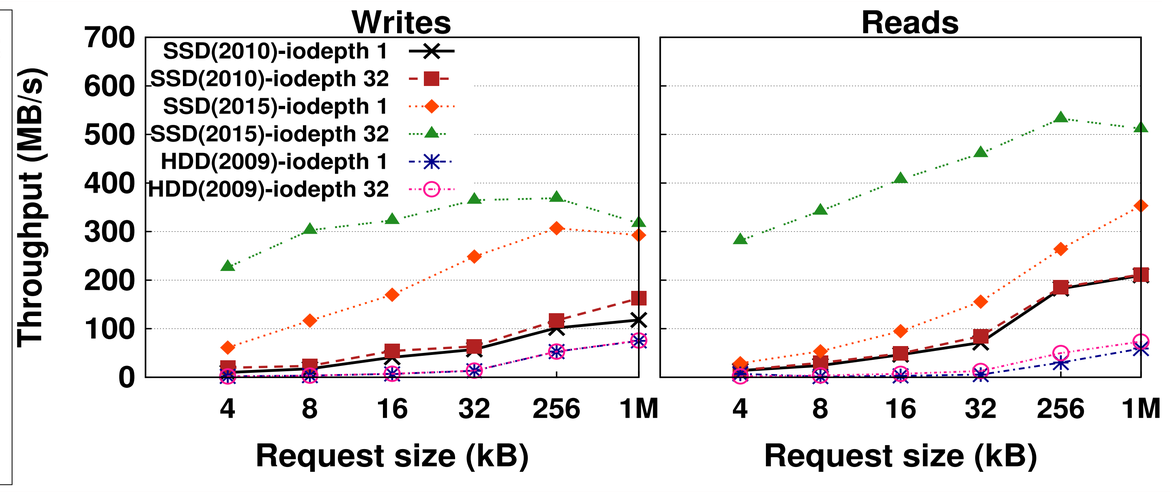
Tucana: Design and Implementation of a Fast and Efficient Scale-up Key-value Store

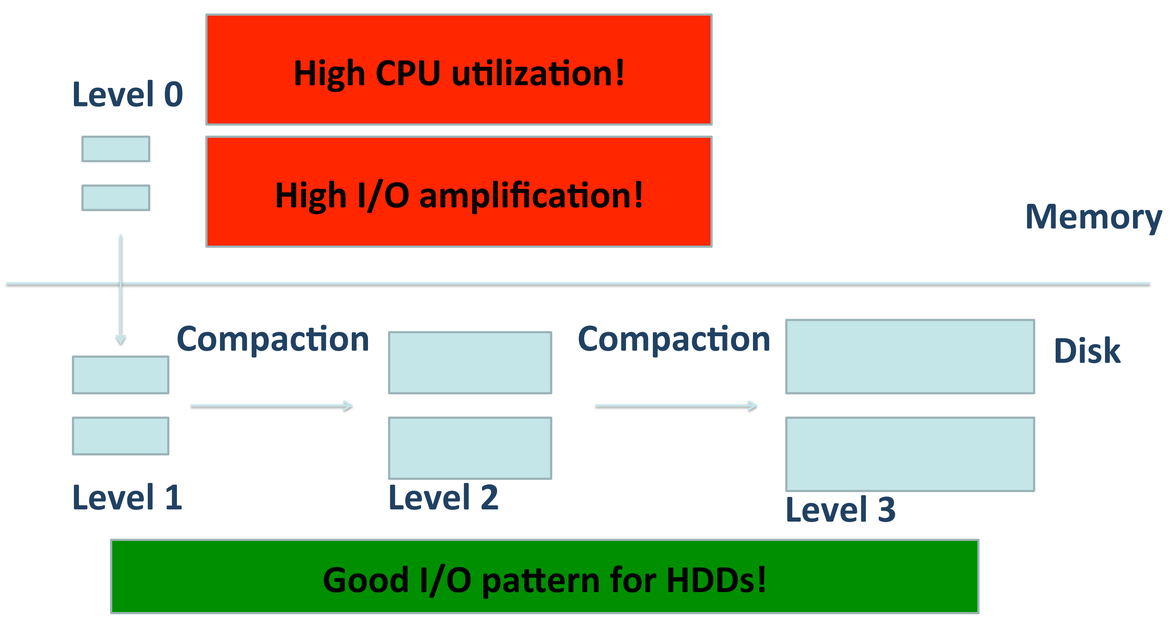
Anastasios Papagiannis, Giorgos Saloustros, Pilar González-Férez, Angelos Bilas
{**apapag**, gesalous, pilar, bilas}@ics.forth.gr
Foundation for Research and Technology - Hellas (FORTH), Greece
University of Crete, Greece University of Murcia, Spain

Key-Value Stores Today

- Key-value store: Dictionary that stores **arbitrary** key-value pairs
  - Used **extensively**: web indexing, social networks, data analytics
  - Supports inserts, point and range queries (scan), and deletes
- Today key-value stores are **inefficient**
  - **Consume a lot of CPU cycles -** mainly designed for HDDs
- Our goal: **Improve efficiency of key-value stores**
  - Reconsider design of key-value stores for fast storage (SSDs)



Issues with current approaches

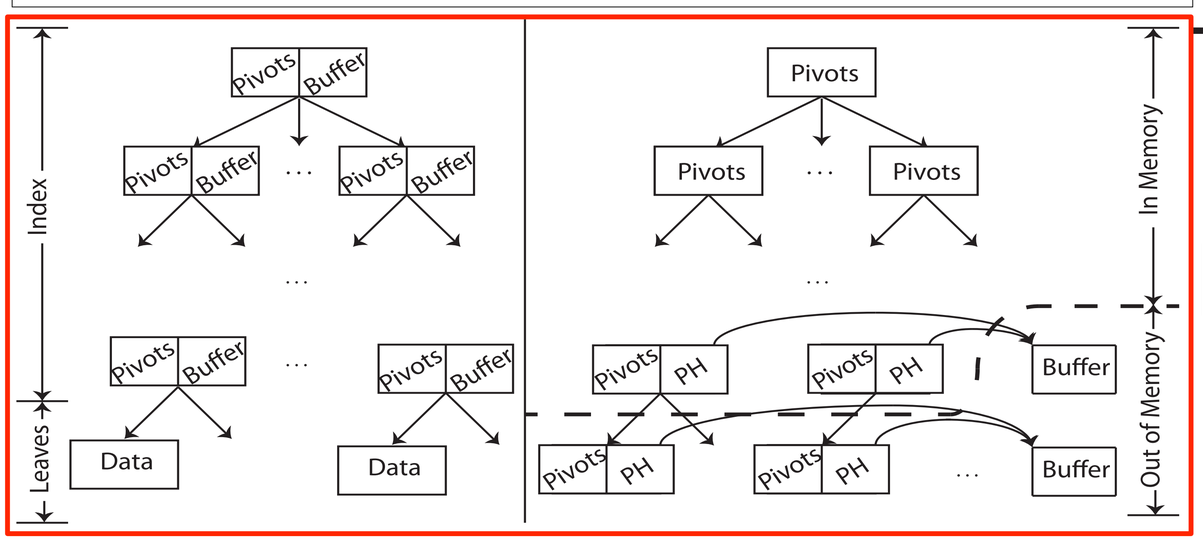


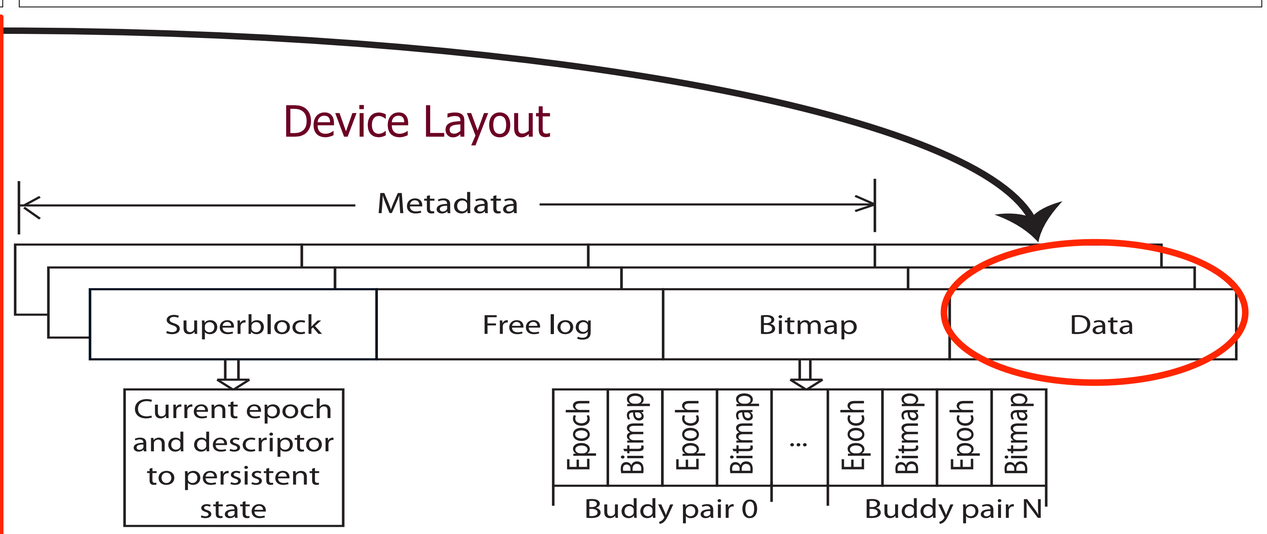
- **Data caching and I/O**
  - Key-value stores require a user-space DRAM cache
  - Explicit I/O using read()/write()
  - Hits in DRAM require lookup – significant overhead
  - Both for index and data, in every traversal
  - Alternative: **mmap()**
- **Failure atomicity via Write-ahead-log**
  - Sequential on device – BUT: requires additional I/O
  - Alternative: **Failure atomicity via Copy-on-Write**

Tucana Design

- **Modified Bᵋ-Tree**
- Buffer at last level of tree that **fits** in memory and below
- Single **append-only** buffer per level
- Typical key, value sizes: 10% internal nodes - 90% leaves
- Similar to DRAM-SSD cost ratio
- Practically: Keep **full index** in DRAM - same cost as SSD

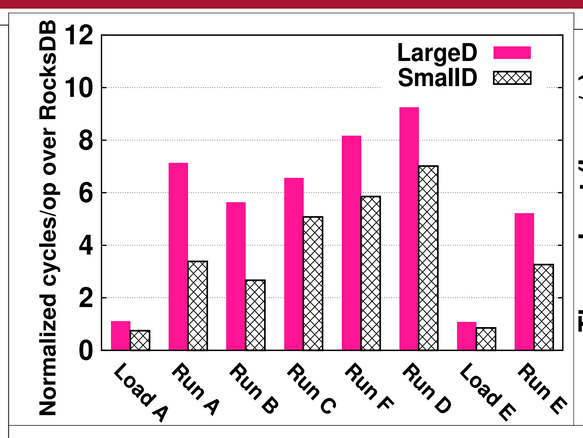
- **Hashes** and **prefixes** in leaf nodes to reduce I/Os
- We use **mmap** instead of explicit I/O
  - No overhead for hits in DRAM – page-faults only for misses
  - Several, non-trivial issues to address
- **Copy-On-Write** instead of Write-ahead-Log for persistence
  - Use with mmap and versions to provide failure atomicity



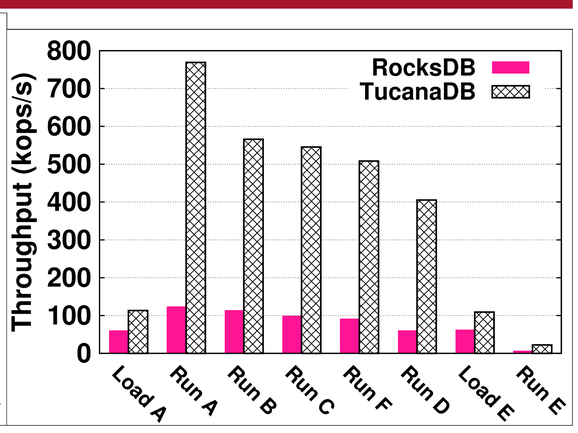


Experimental Analysis

- YCSB, Small (memory) and Large (device) Dataset
- **Efficiency:** RocksDB - up to 9.2x in cycles/op (HBase – up to 8x, Cassandra – up to 22x)
- **Throughput:** RocksDB – up to 7x in ops/sec (Hbase – up to 5.4x, Cassandra – up to 10.7x)



Efficiency



Small Dataset

With the support of the European Commission through the CoherentPaaS (FP7-ICT-611068) and LeanBigData (FP7-ICT-619606) projects and the NESUS COST Action IC1305.

Computer Architecture & VLSI Systems Laboratory (CARV)
http://www.ics.forth.gr/carv