SDC
STORAGE DEVELOPER CONFERENCE
SNIA SANTA CLARA, 2017

Workload Analysis of Key-Value Stores on Non-Volatile Media

Vishal Verma (Performance Engineer, Intel)
Tushar Gohad (Cloud Software Architect, Intel)

Outline

- KV Stores – What and Why
- Data Structures for KV Stores
- Design Choices and Trade-offs
- Performance on Non-Volatile Media
- Key Takeaways

SDC
STORAGE DEVELOPER CONFERENCE
SNIA SANTA CLARA, 2017

Intro to KV stores

What are KV Stores

- Type of NoSQL database that uses simple key/value pair mechanism to store data
- Alternative to limitations of traditional relational databases (DB):
  - Data structured and schema pre-defined
  - Mismatch with today's workloads. Data: Large and unstructured, lots of random writes and reads.
- Most flexible NoSQL model as application has complete control over what is stored inside the value

What are KV Stores

- Key in a key-value pair must (or at least, *should*) be unique. Values identified via a key, and stored values can be numbers, strings, images, videos etc
- API operations: **get**(key) for reading data, **put**(key, value) for writing data and **delete**(key) for deleting keys.
- **Phone Directory** example:

Key	Value
Bob	(123) 456-7890
Kyle	(245) 675-8888
Richard	(787) 122-2212

KV Stores benefits

- **High performance**: Enable fast location of object rather than searching through columns or tables to find an object in traditional relational DB
- **Highly scalable**: Can scale over several machines or devices by several orders of magnitude, without the need for significant redesign
- **Flexible** : No enforcement of any structure to data
- **Low TCO**: Simplify operations of adding or removing capacity as needed. Any hardware or network failure do not create downtime

Types of KV stores

Types of KV Stores: B-Trees

- Classic disk based structure for indexing records based on an ordered key set
- Writes, Updates and Deletes are far more expensive than Reads
- Suffers fragmentation
- B-Trees generally support good read and write concurrency
- DB engines like MySQL InnoDB, MongoDB Wild Tiger etc. use B-Trees to store data

B-Trees

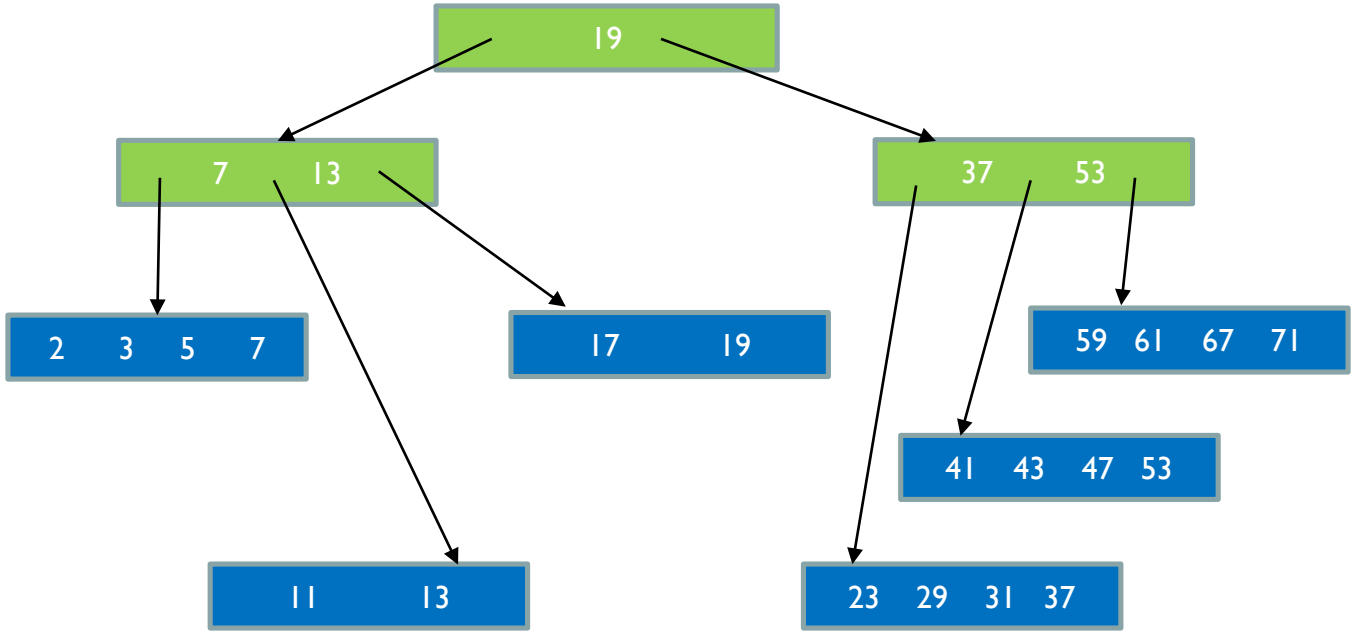


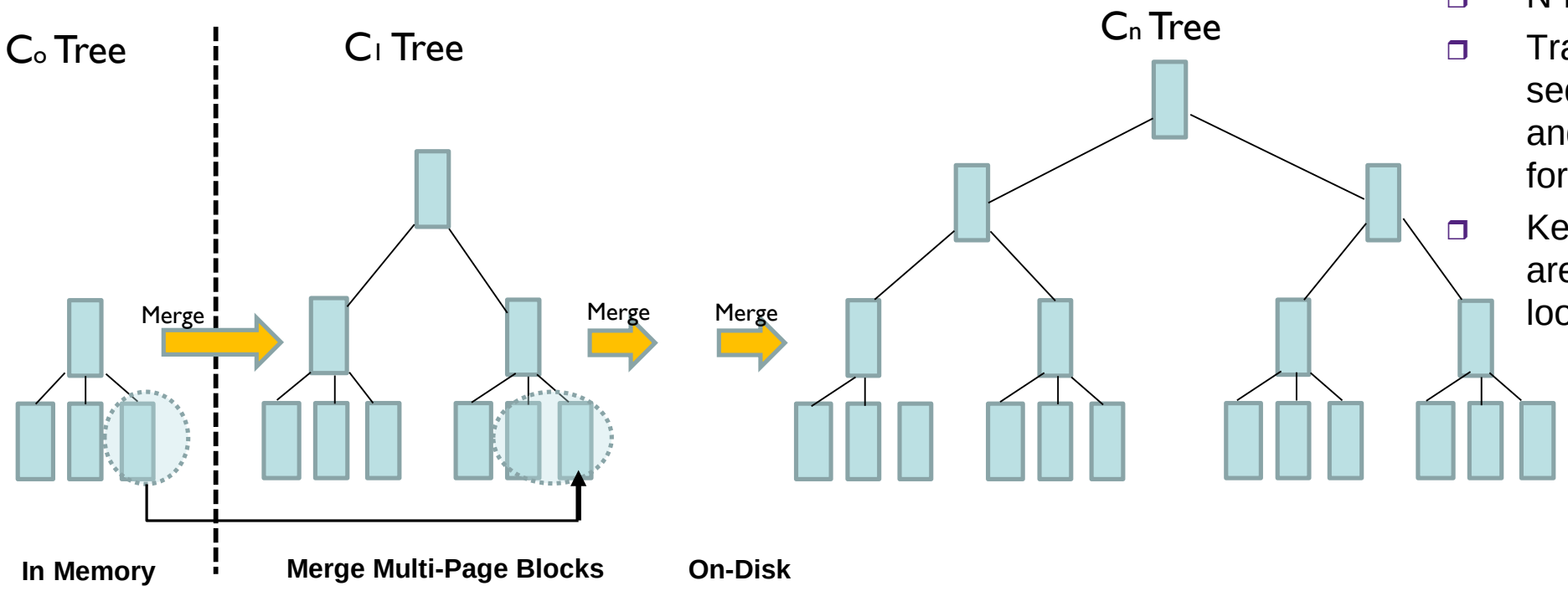
Figure 1: A B-Tree containing first twenty prime numbers

- **Internal nodes** containing pivot keys has children and are shaded green
- **Leaf nodes** containing only data records are shaded blue
- Search takes O(log n) time
- Insertion/Deletion may need to rebalance the tree
- For worst-case insertion, every insertion requires writing leaf block. Incurs small block random I/Os to disk.
- Write amplification worse

Types of KV Stores: LSM Trees

- The log-structured merge-tree (LSM-tree) are designed to provide better write throughput than traditional B- tree approaches
- Converts random writes to sequential writes to get better database performance
- No fragmentation
- Numerous DB engines like MyRocks, MongoDB RocksDB, Cassandra, CouchBase, Hbase, LevelDB use LSM Trees to store data

Types of KV Stores: LSM Trees



- N-level Merge tree
- Transform random writes into sequential writes using logfile and memtables. Optimized for insertions by buffering
- Key value items in the store are sorted to support faster lookups

Figure 2: LSM Tree high level diagram

Types of KV Stores: Fractal Trees

- Merges features from B-trees with LSM trees
- A Fractal Tree index has buffers at each node, which allow insertions, deletions and other changes to be stored in intermediate locations
- Fast writes slower reads and updates
- Better than LSM for range reads on cold cache, but the same on warm cache
- Commercialized in DBs by Tokutek

Fractal Trees

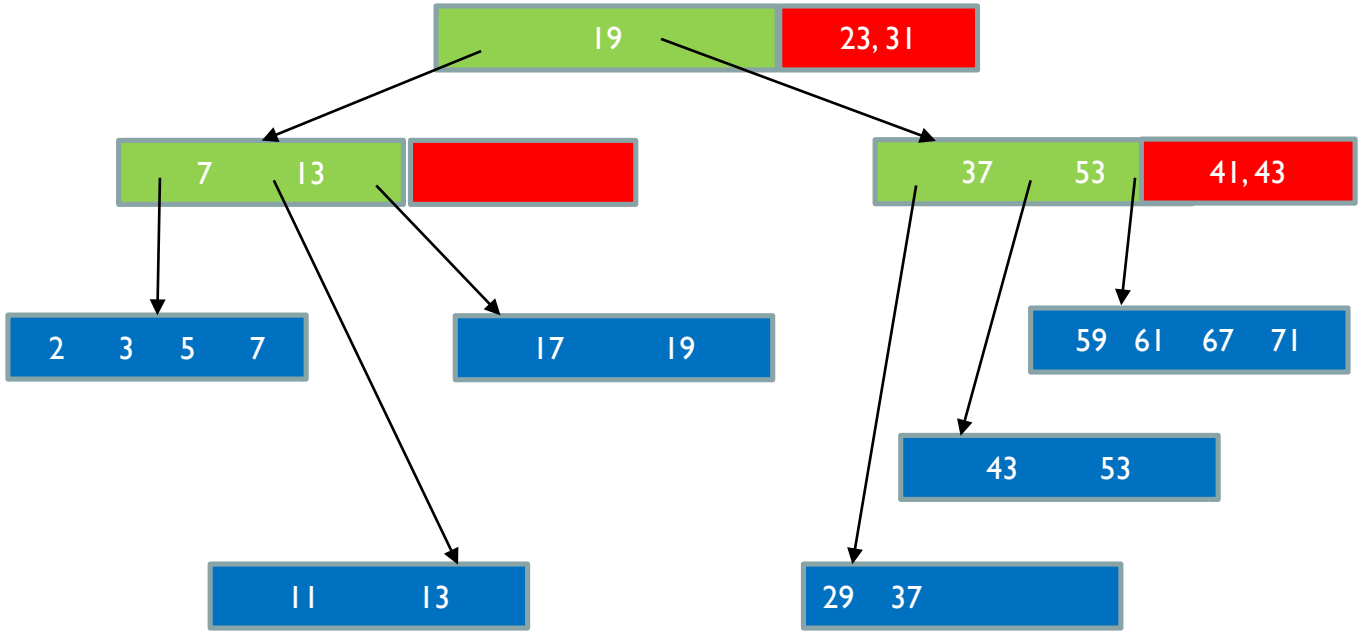


Figure 3: A FT index after inserting 31, and tree has already moved 41 and 42 into the child of root

- Maintains a B tree in which each **internal node** contains a **buffer,** shaded RED
- To insert a data record, simply insert it into the buffer at the root of the tree vs. traversing entire tree (B-Tree)
- When root buffer full, move inserted record down a level until they reach leaves and gets stored in leaf node like B-Tree
- Several leaf nodes are grouped in turn accounting for large I/O (~1-4MB) sizes
- Excellent write amplification

Performance Comparisons

System, Dataset & Workload Configuration

System Configuration	OS Configuration	SSD Details
Processor: Intel(R) Xeon(R) CPU E5-2618L v4 @ 2.20GHz **Total Physical CPU** (HT disabled): 20 **Total Memory**: 64GB	**Distro:** Ubuntu 16.04.1 LTS **Kernel:** 4.12.4 (built) **Arch:** x86_64 • Intel ® SSD DC P4500 Series • Intel ® Optane™ SSD DC P4800X	**SSD:** 1x Intel ® Optane™ SSD DC P4800X 375GB
Linux Kernel 4.12.0 Tuning parameters		
Page Cache Size: 30GB (using cgroups)		
Bytes per sync: 64KB XFS filesystem, agcount=32, mount with discard		

Configuration may change

Dataset:

242GB (250 million uncompressed records)

- Dataset size kept higher (4:1) than main memory size
- Fills ~70% of disk capacity

Key_size: 16 Bytes

Value_size: 1000 Bytes

Db_bench:

Test Duration: 5 minutes

No. of Runs for each test: 3

Compression: None

Workloads:

- ReadWrite
- Overwrite
- Randread

Workload # 1: Readwrite
Intel Nand SSD

Compare B-Tree vs. LSM readwrite performance on Intel Nand SSD

Workload # 1: Readwrite
Intel Optane SSD

Compare B-Tree vs. LSM readwrite performance on Intel Optane SSD

Workload # 2: Overwrite Intel Nand SSD

Compare B-Tree vs. LSM overwrite performance on Intel Nand SSD

Workload # 2: Overwrite
Intel Optane SSD

Compare B-Tree vs. LSM overwrite performance on Intel Optane SSD

Workload # 3: Randread
Intel Nand SSD

Compare B-Tree vs. LSM randread performance on Intel Nand SSD

Workload # 3: Randread
Intel Optane SSD

Compare B-Tree vs. LSM randread performance on Intel Optane SSD

Key Takeaways

Types of KV Store	Amplification		ReadWrite Performance		Overwrite Performance		Randread Performance	
	Read	Write	Nand SSD	Optane SSD	Nand SSD	Optane SSD	Nand SSD	Optane SSD
B- Tree								
LSM Tree								
Fractal Tree								

Table 1: Comparing different KV stores on Intel Non-Volatile media

Summary