

基于NVMM的存储引擎优化

2022.1.7

龚凯



目录

C O N T E N T S

- 01.** NVM介绍
- 02.** NVM对B+树的优化思路
- 03.** NVM对LSM-tree的优化思路
- 04.** 总结

目录

C O N T E N T S

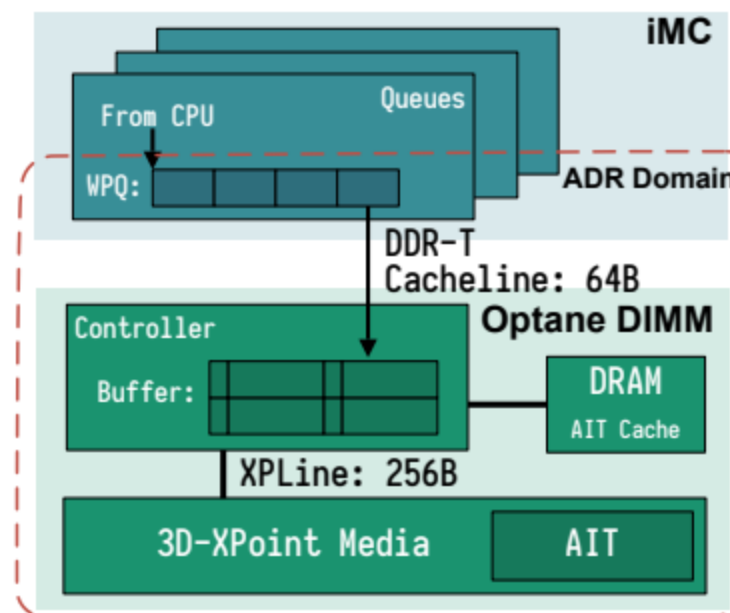
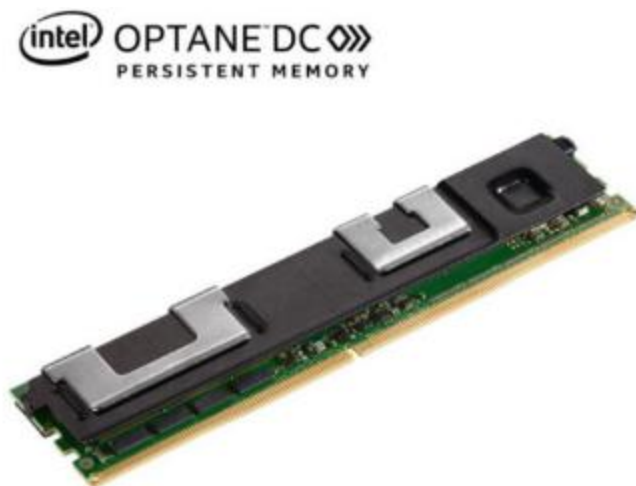
01. NVM介绍

02. NVM对B+树的优化思路

03. NVM对LSM-tree的优化思路

04. 总结

01 NVM介绍



(b) Optane DIMM Overview

1. 数据进入WPQ才会保证持久化
2. CPU与持久化内存以64B的粒度进行读写

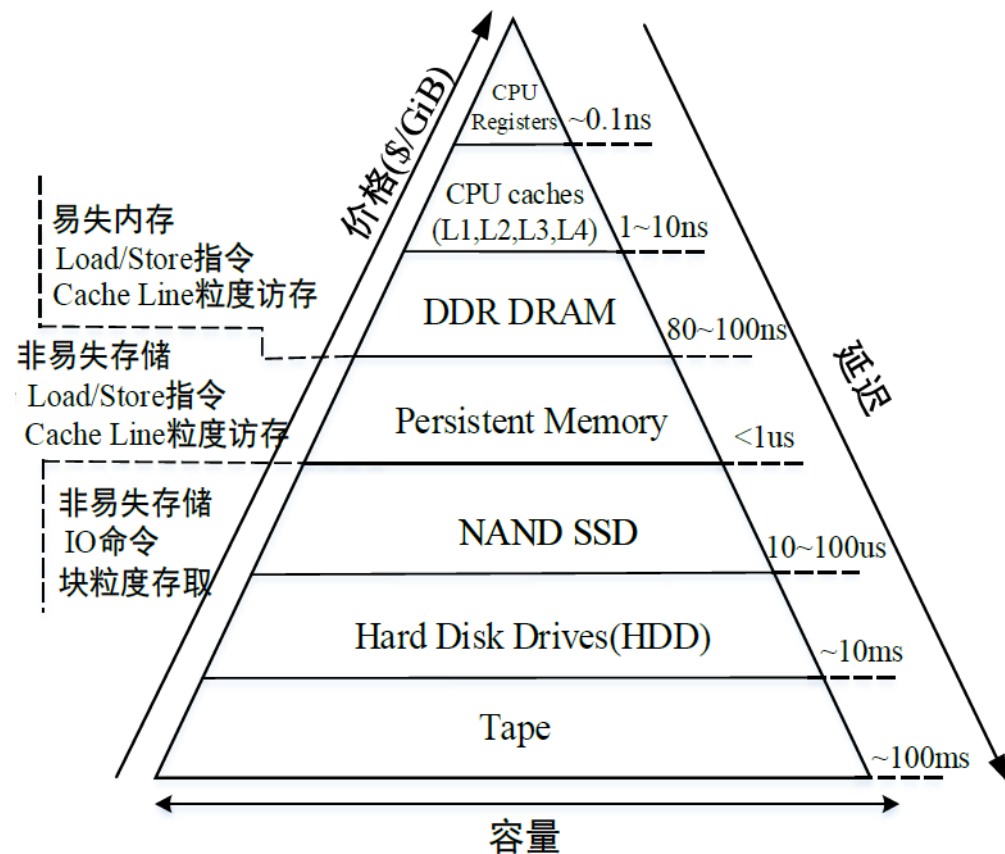
NVM介绍

Table 1 Characteristics NVM and other storage technologies

	DRAM	NVM	SSD	HDD
读延迟	60 ns	250 ns	25 us	10 ms
写延迟	60 ns	500 ns	300 us	10 ms
寻址单元	字节	字节	块	块
易失性	易失	非易失	非易失	非易失
容量	1x	4x	4x	—
写寿命	10^{16}	10^{10}	10^5	10^{16}
价格	>150x	>60x	>8x	1x

NVM特性:

1. 读写性能差DRAM几倍
2. 读写不对称
3. 字节寻址



NVM与DRAM编程的差异:

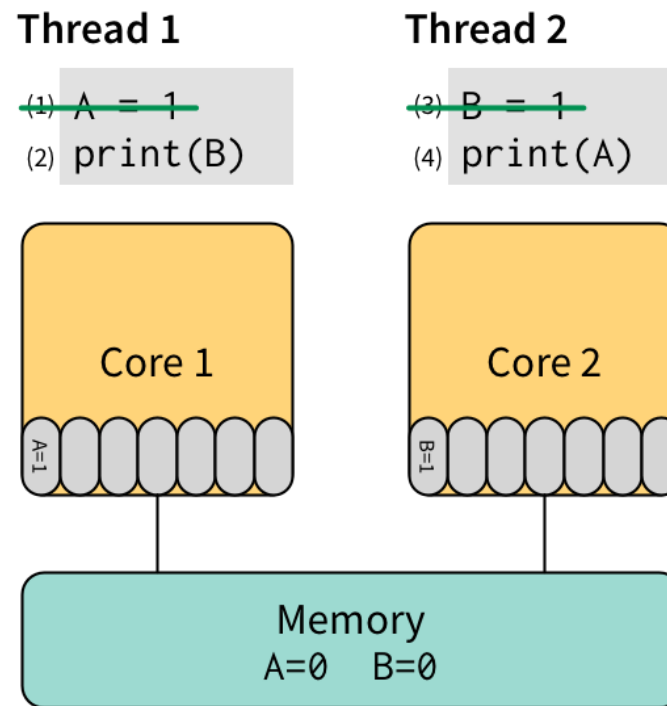
1. 数据的持久化
2. 数据的一致性

01 NVM介绍



引入NVM的挑战:

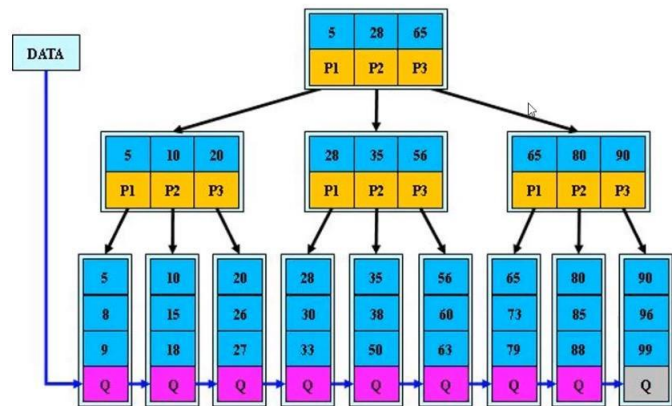
1. 缺少保证数据一致性的封装 (MFENCE、CLFLUSH)
2. 如何充分利用NVM的存储资源
3. 写慢于读, 如何对写进行优化



Algorithm 1 Insert(*key*, *value*, *prevNode*)

```
1: curNode := NewNode(key, value);
2: curNode.next := prevNode.next;
3: mfence();
4: clflush(curNode);
5: mfence();
6: prevNode.next := curNode;
7: mfence();
8: clflush(prevNode.next);
9: mfence();
```

NVM介绍



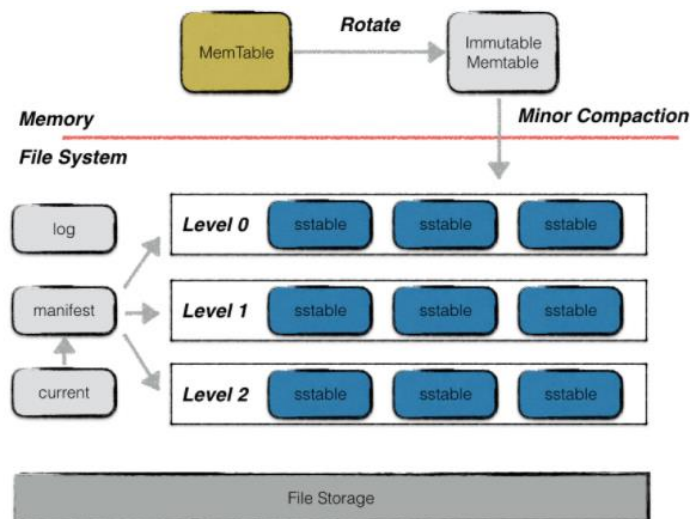
B+树:

1. 基于HDD架构而设计
2. Page大小和磁盘块大小对应

引入NVM之后:

B+树架构设计:

1. NVM
2. DRAM+NVM



LSM-tree:

1. 基于DRAM-HDD/SSD架构设计
2. 利用顺序写提高写性能

LSM-tree架构设计:

1. NVM+SSD
2. DRAM+NVM+SSD

目录

C O N T E N T S

01. NVM介绍

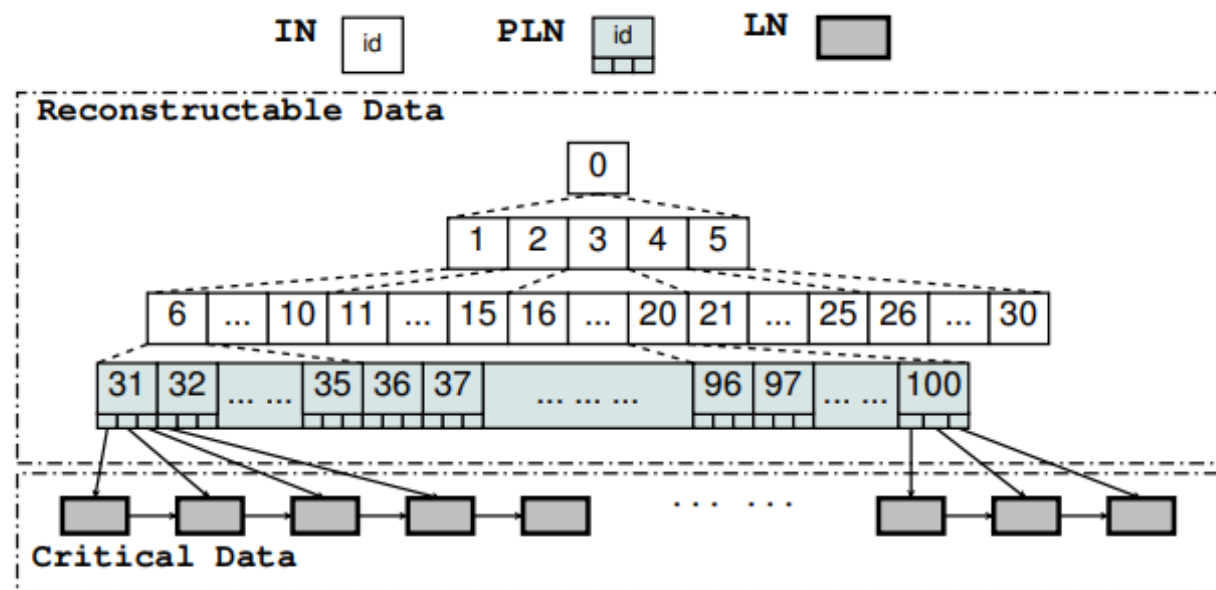
02. NVM对B+树的优化思路

03. NVM对LSM-tree的优化思路

04. 总结

NVM单级架构

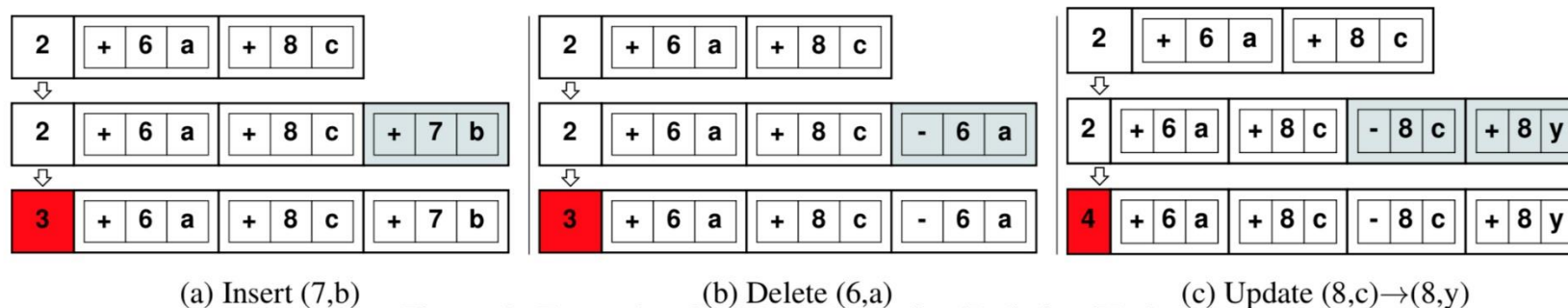
1. 只保证叶子节点的一致性
2. 叶子节点上的数据不需要保持有序
3. 使用一种处理器缓存友好的格式存储内部节点。



Node Layout

IN	nKeys	key[0]	key[1]	...	key[2m]			
PLN	nKeys	key[0]	key[1]	...	key[m]			
		LN[0]	LN[1]	...	LN[m]	LN[m+1]		
LN	nElements	flag	key	value	flag	key	value	...
		LN Element[0]			LN Element[1]			

NVM单级架构

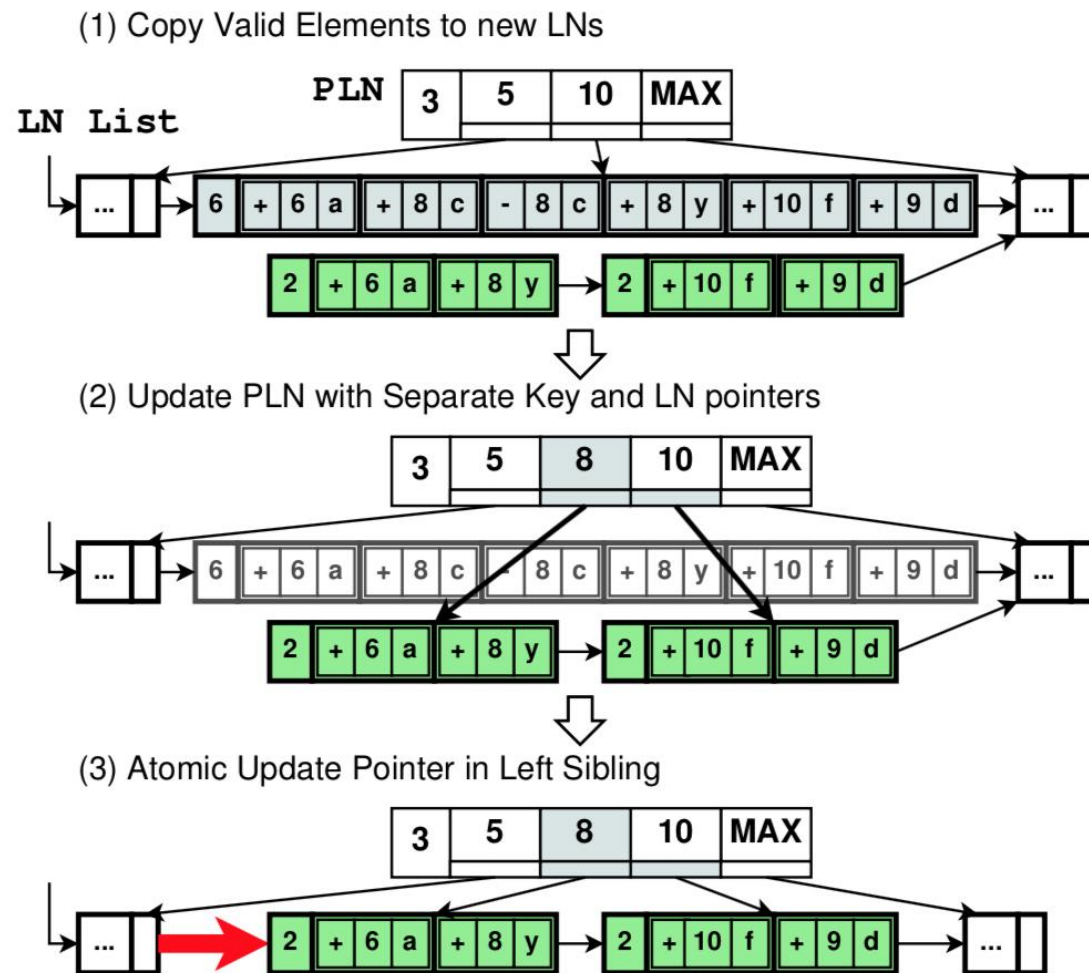


1. 所有的新数据都以append-only方式插入到新节点
2. 只有这个计数器修改成功，插入操作才算完成

NVM单级架构

当NV-Tree发生了叶节点的分裂或者合并操作时，它使用 cflush 指令控制了持久化操作的顺序

1. 拷贝合法记录到新的叶节点；
2. 更新PLN节点，插入新的叶节点。这时候没有产生任何修改原叶节点链表的操作，不影响叶节点的一致性；
3. 原子性更新分裂叶节点的前节点，指向新的叶节点。



DRAM-NVM架构

1. 采用批量写和合并,buffer tree接受的写操作批量更新到base tree
2. Buffer tree类似memtable
3. Base tree的索引部分采用radix tree

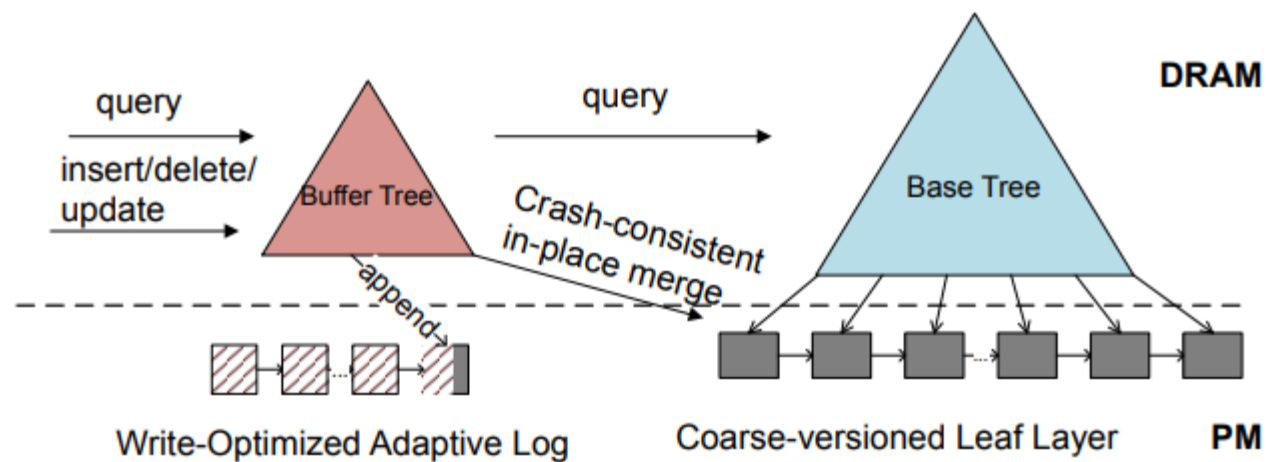


Figure 1: DP-Tree Architecture

DRAM-NVM架构

1. 叶节点数据部分保证一致性
2. 每个叶子节点包含两份元数据来进行版本控制
3. gv指明那个元数据处在激活状态
4. Bitmap用于指明当前元数据拥有的键值对
5. Fingerprint支持快速查找

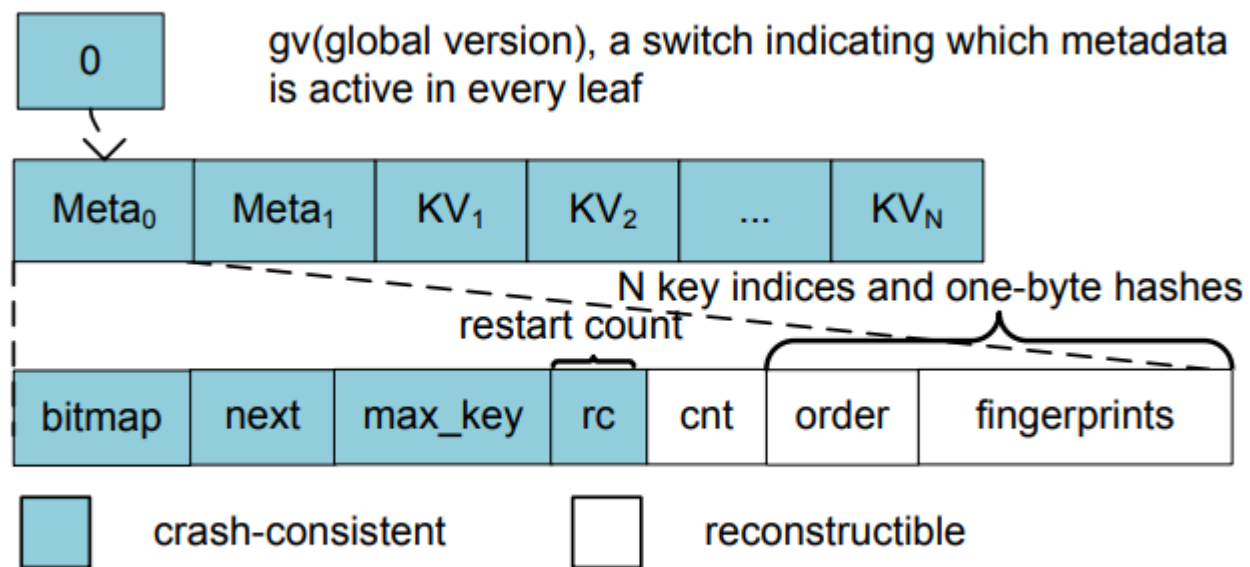
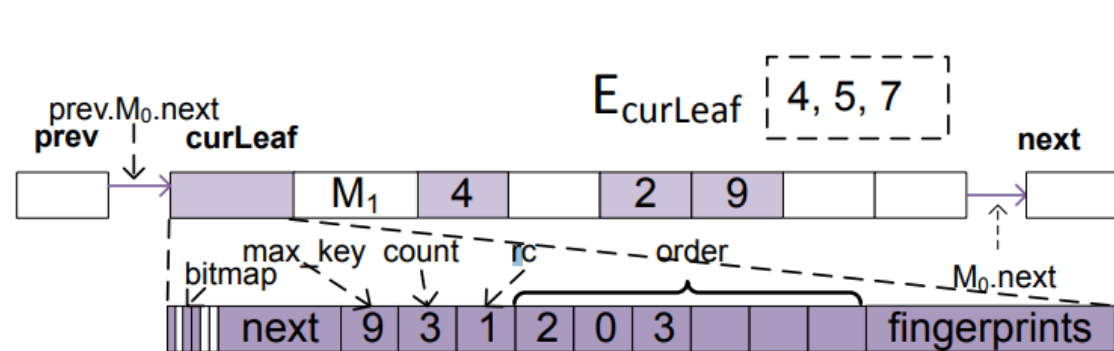
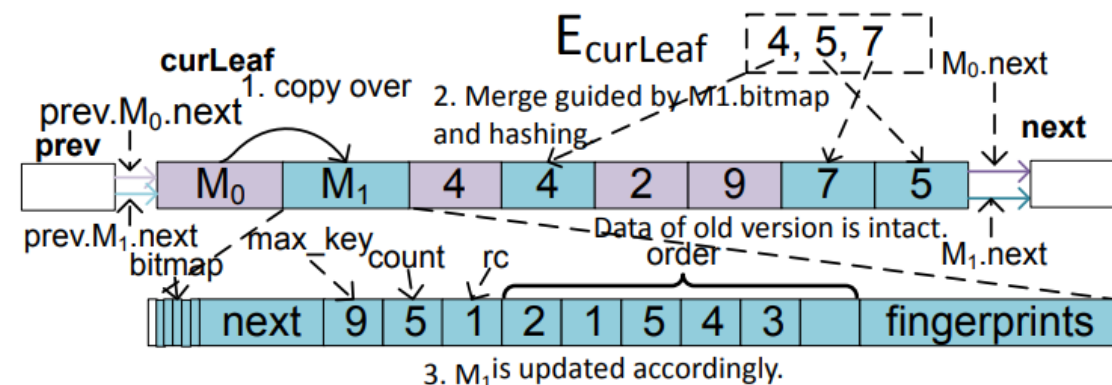


Figure 3: Base Tree Leaf Layout

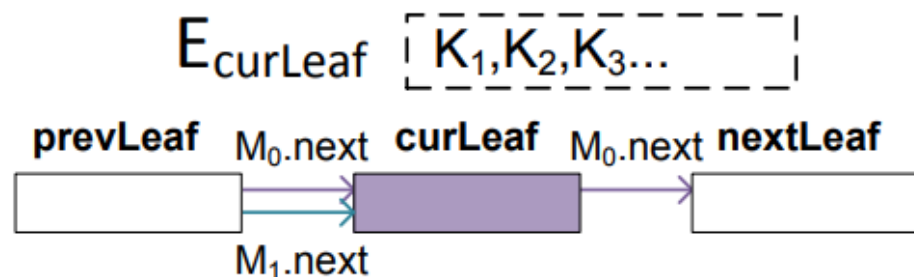


(a) Node Before Merge

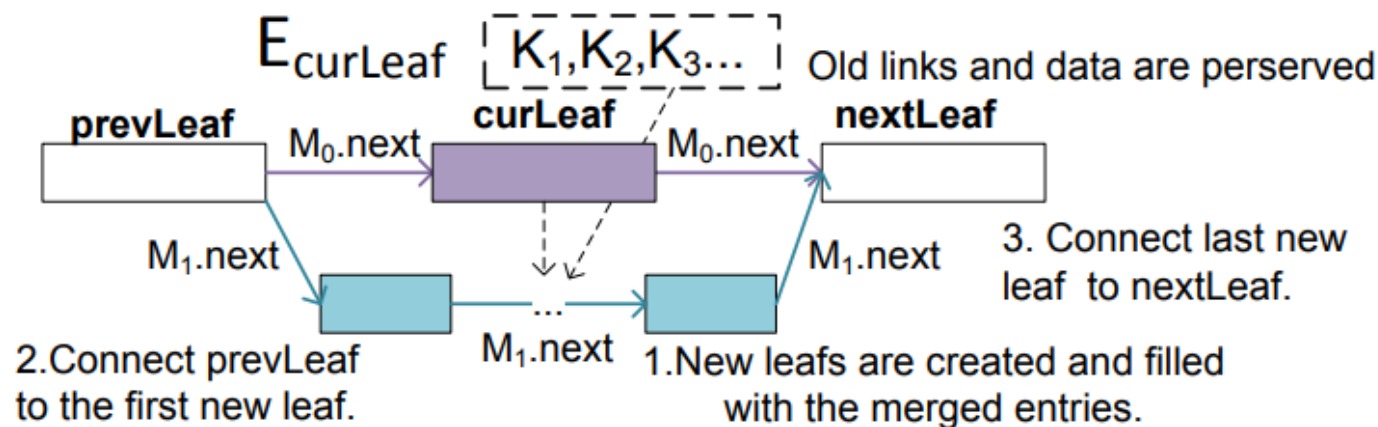


(b) Node After Merge

Figure 5: Before and After Image of Leaf Upsert Merge(No Split)



(a) Before Split



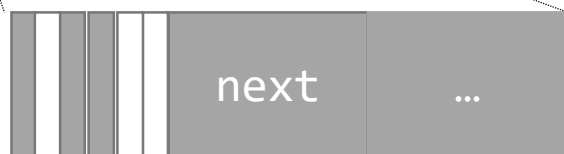
(b) After Split

Figure 4: Before and After Image of Leaf Upsert Merge(Split)

02 DP-Tree



GV=0



copy

Insert 4,5,7

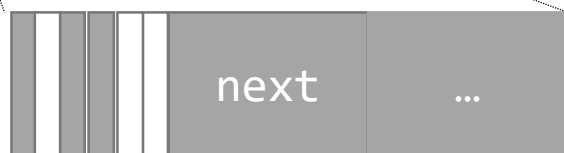


1. 不会覆盖当前gv变量所指的那一份元数据所表示的KV数据。
2. 持久化相关修改后，原子更新
3. 通过就地更新减少写放大

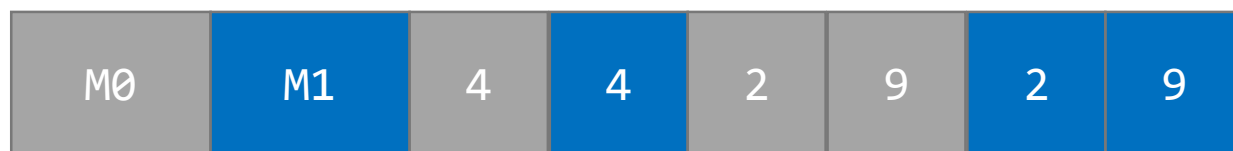
02 DP-Tree



GV=1



Insert 4,5,7



1. 不会覆盖当前gv变量所指的那一份元数据所表示的KV数据。
2. 持久化相关修改后，原子更新
3. 通过就地更新减少写放大

目录

C O N T E N T S

01. NVM介绍

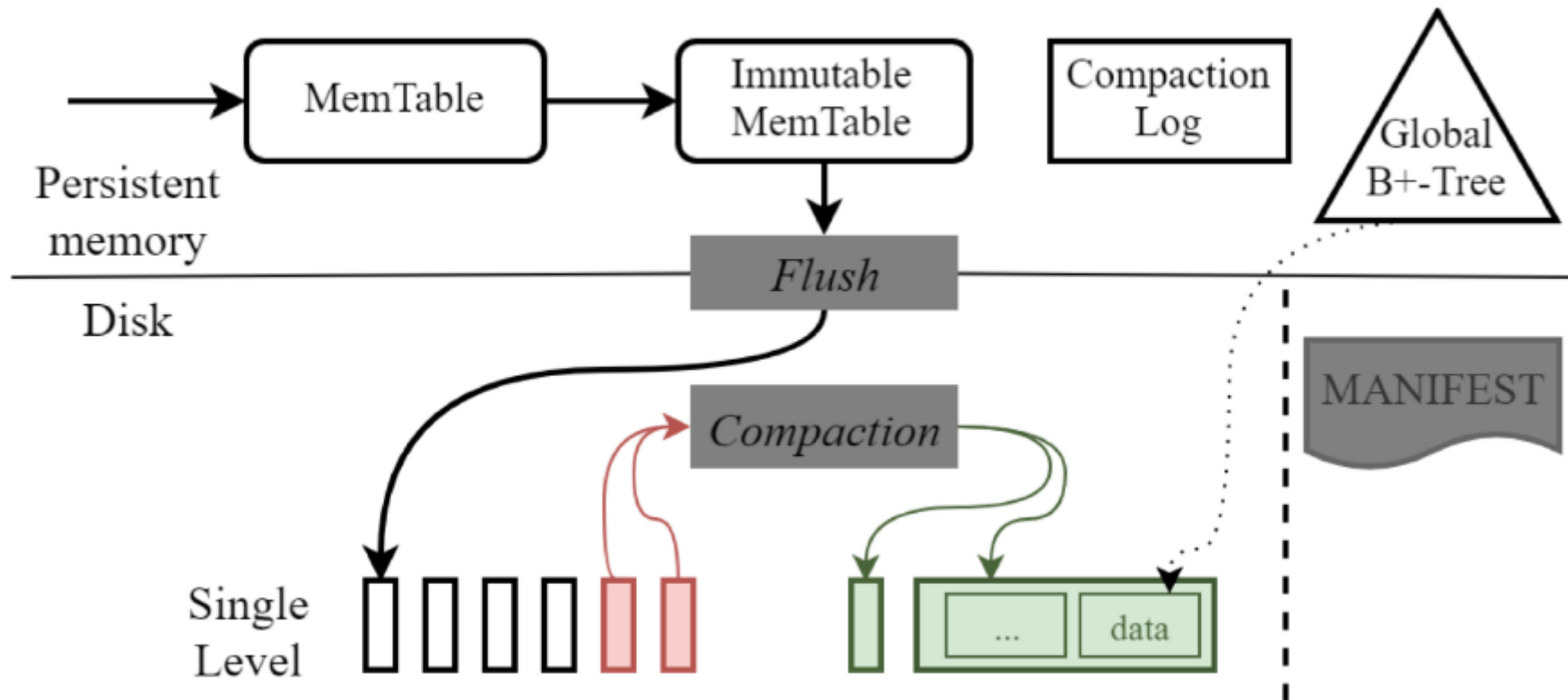
02. NVM对B+树的优化思路

03. NVM对LSM-tree的优化思路

04. 总结

NVM-DISK架构

1. 持久化的memtable
2. 持久化的B+Tree索引
3. 选择性合并



NVM-DISK架构

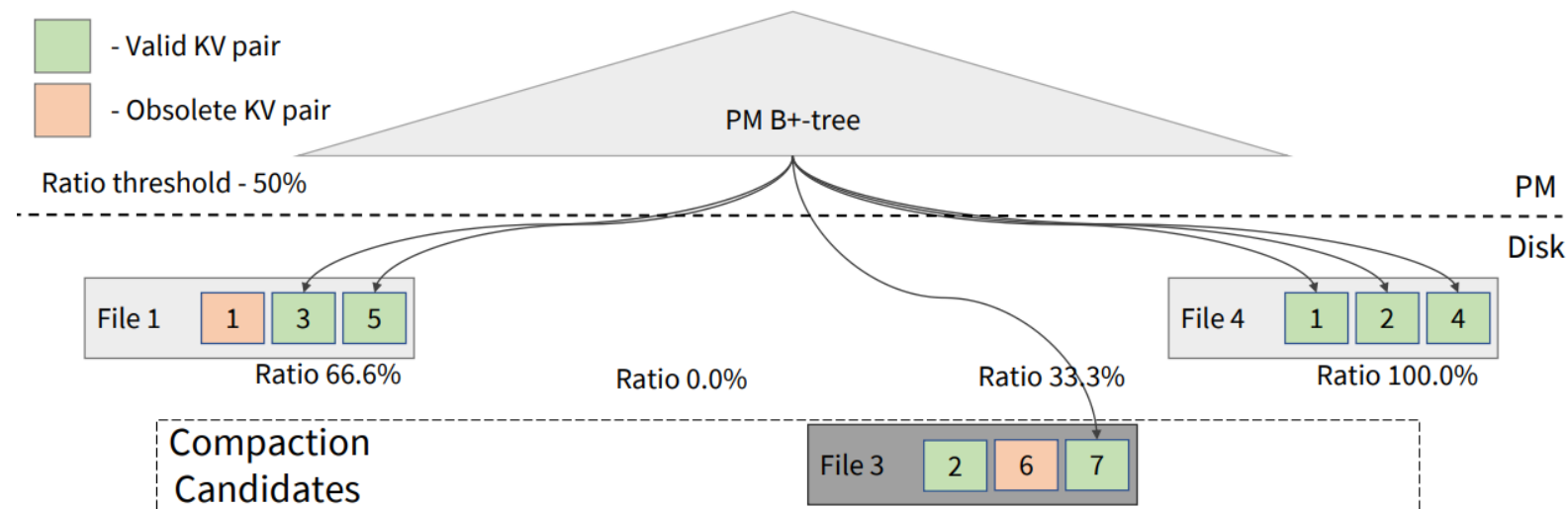
Selective compaction

目的:

1. 减少磁盘占用
2. 保持文件间的有效数据尽量有序

做法:

1. 有效key的比例
2. 叶子节点扫描
3. Range query连续性



NVM-DISK架构

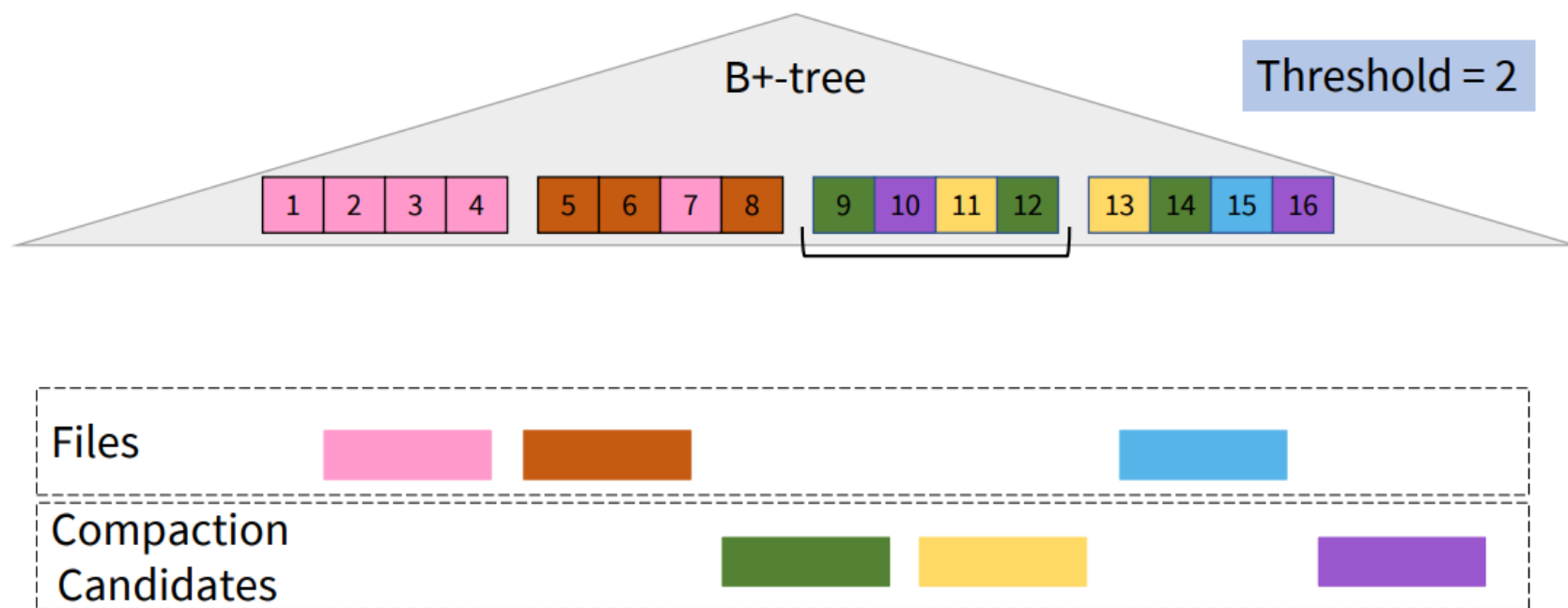
Selective compaction

目的:

1. 减少磁盘占用
2. 保持文件间的有效数据尽量有序

做法:

1. 有效key的比例
2. 叶子节点扫描
3. Range query连续性



NVM-DISK架构

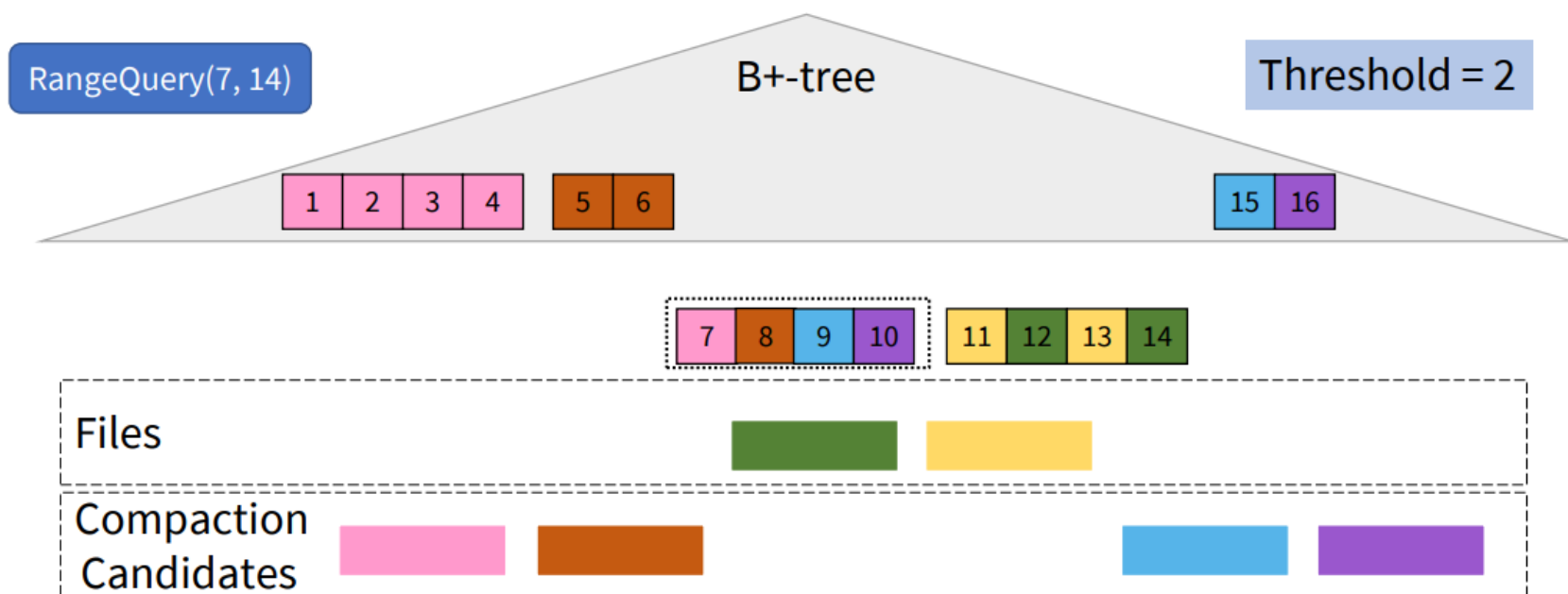
Selective compaction

目的:

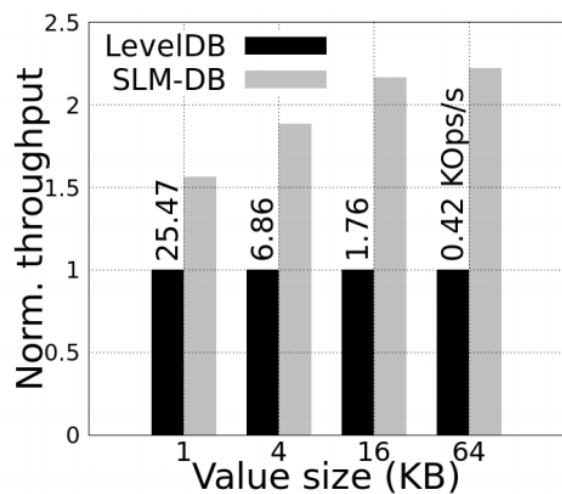
1. 减少磁盘占用
2. 保持文件间的有效数据尽量有序

做法:

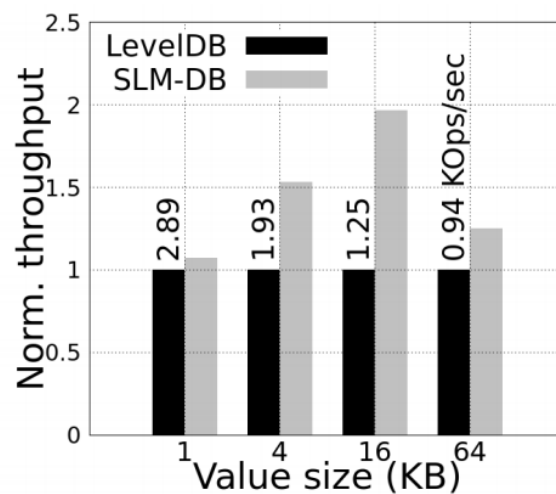
1. 有效key的比例
2. 叶子节点扫描
3. Range query连续性



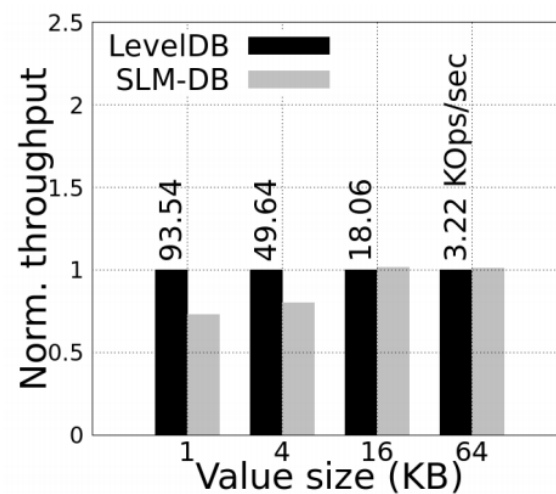
评估测试



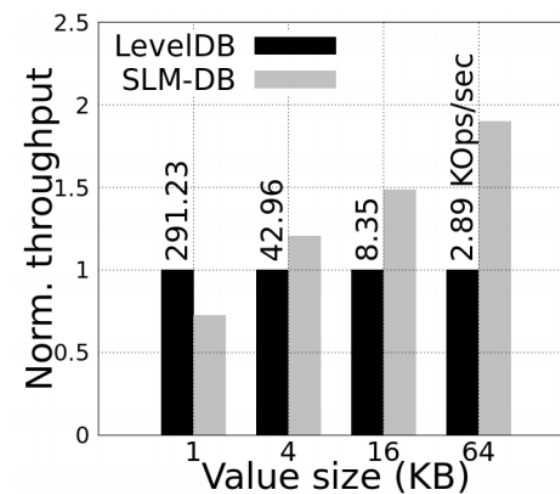
(a) Random Write



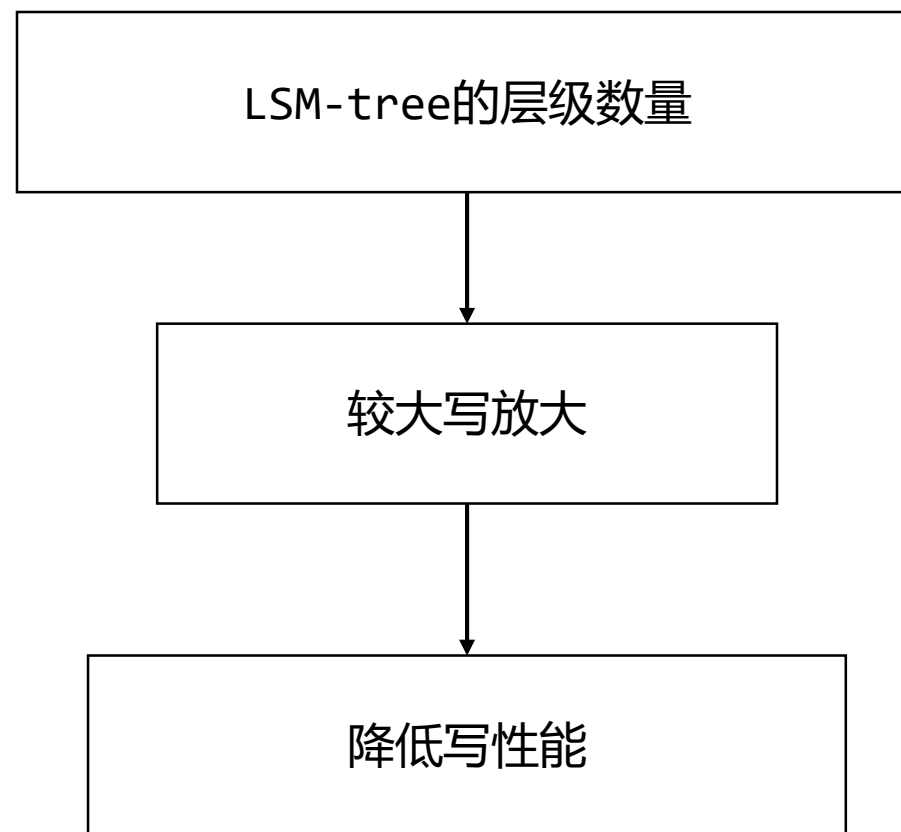
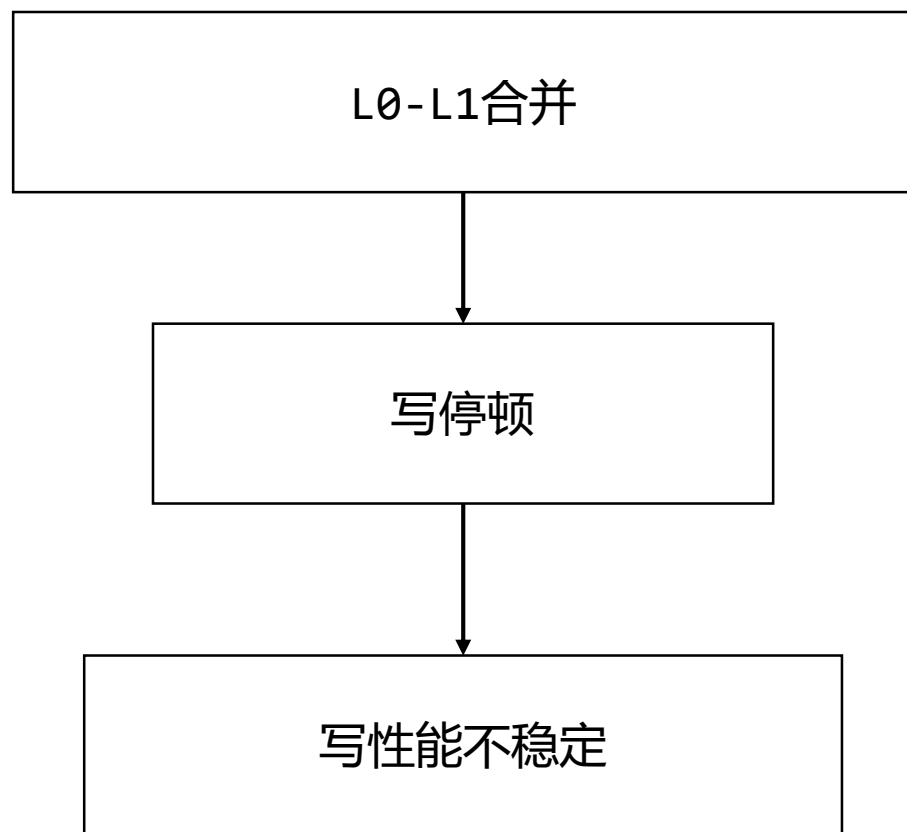
(b) Random Read



(c) Range Query

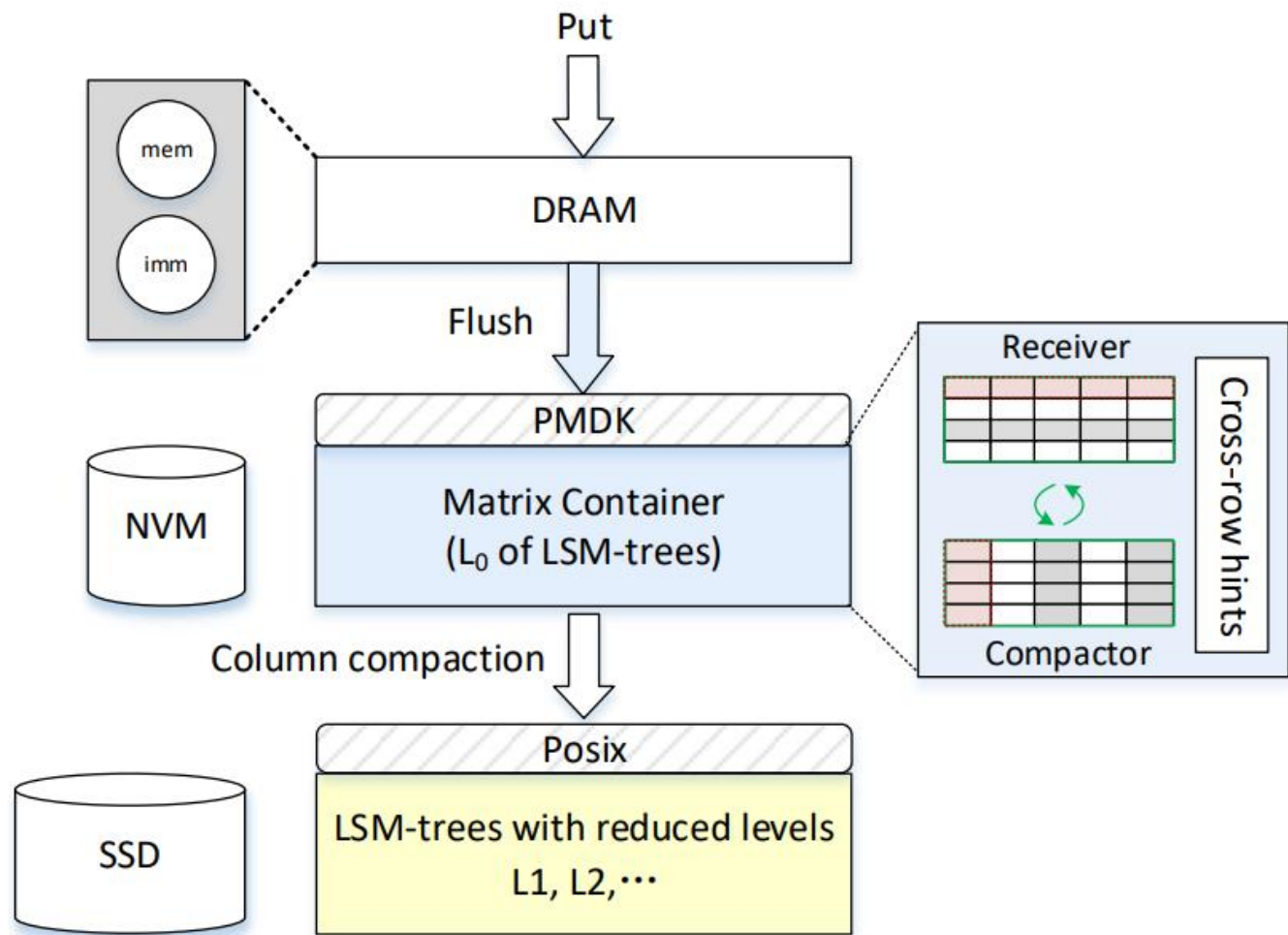


(d) Sequential Read



NVM-DISK架构

1. 重新组织L0中的数据，存放在NVM中
2. 针对L0设计了相关合并策略和查找策略
3. 扩大LSM-tree每层容量，减少层级



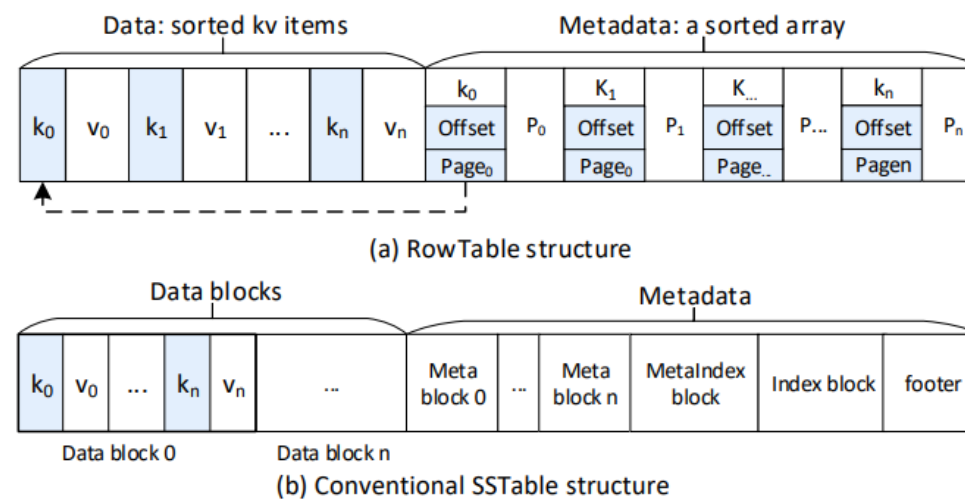
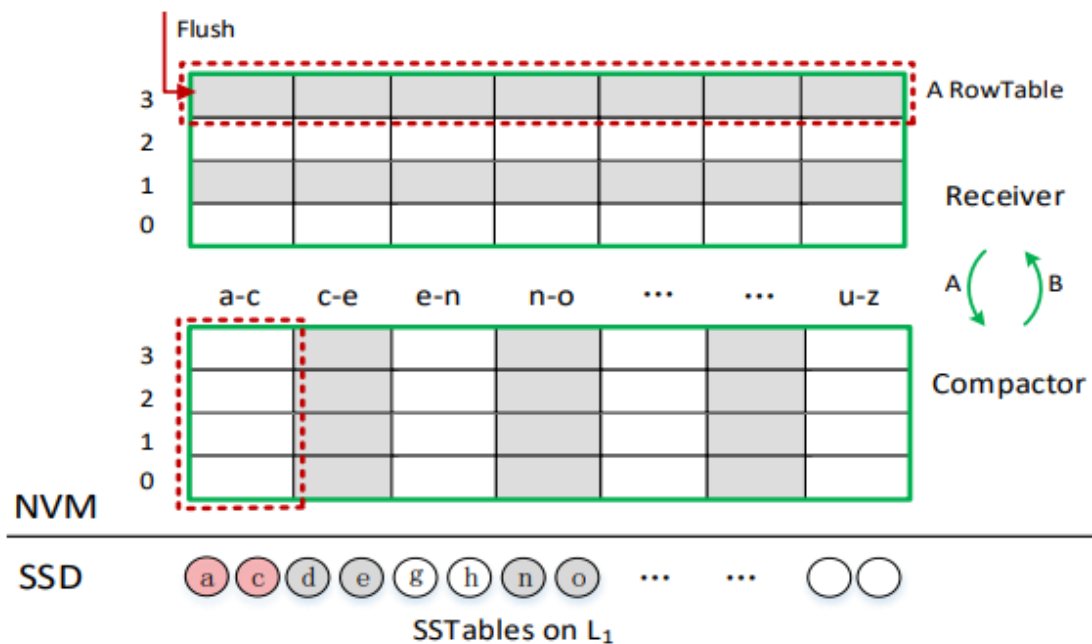
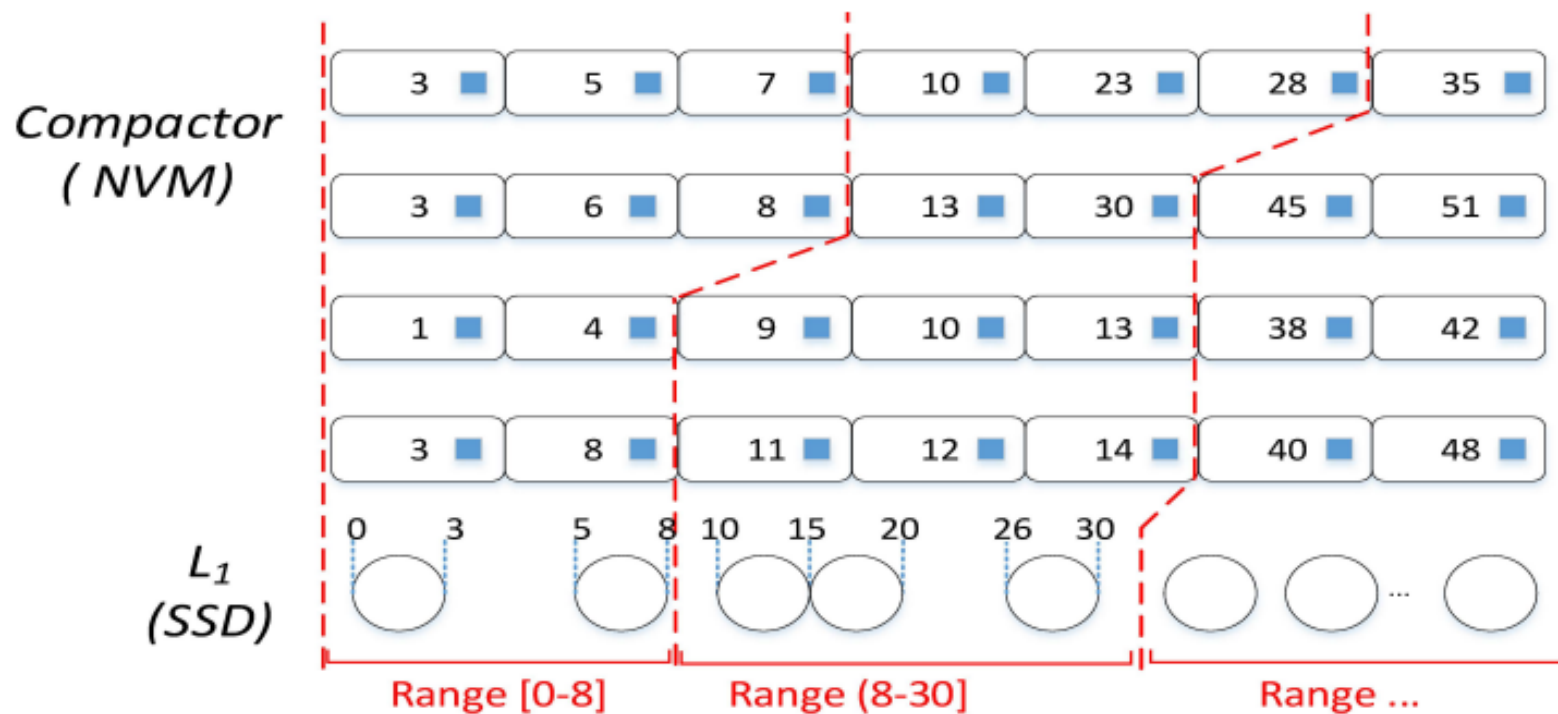
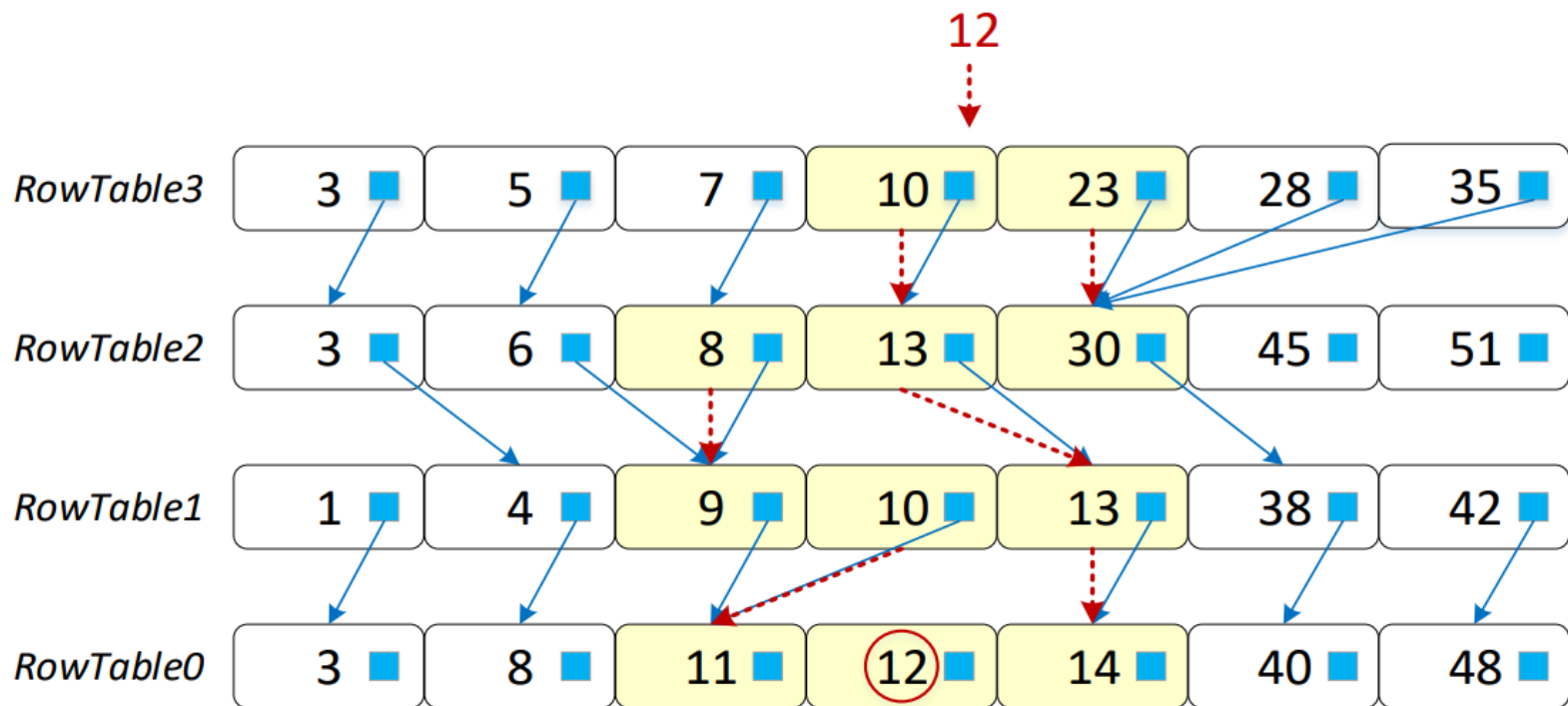


Figure 7: RowTable and conventional SSTable.

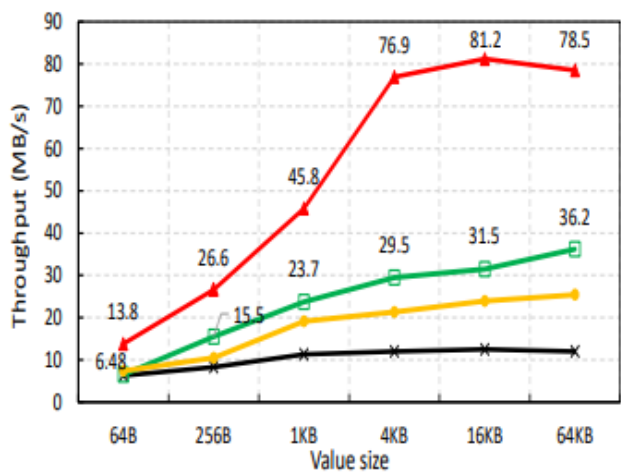
1. 将 L_0 组织存放在一个矩阵容器中
2. Receiver负责接收数据, Compactor负责合并



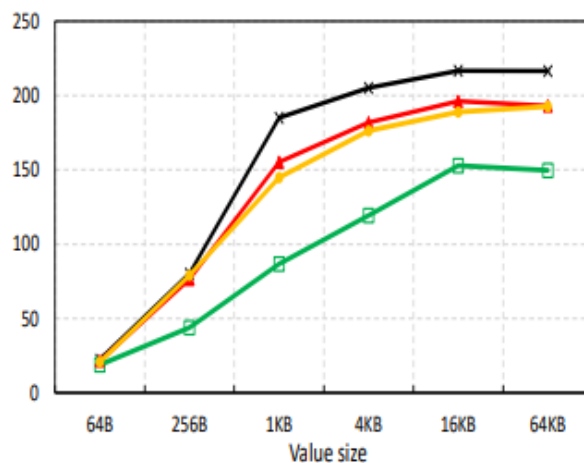
1. 列式合并，L1中的sstable键值不会重叠
2. 在创建rowtable时，创建跨行指针



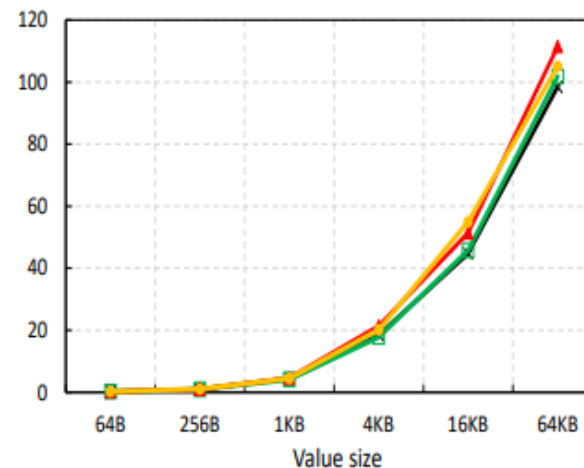
1. 列式合并，L1中的sstable键值不会重叠
2. 在创建rowtable时，创建跨行指针



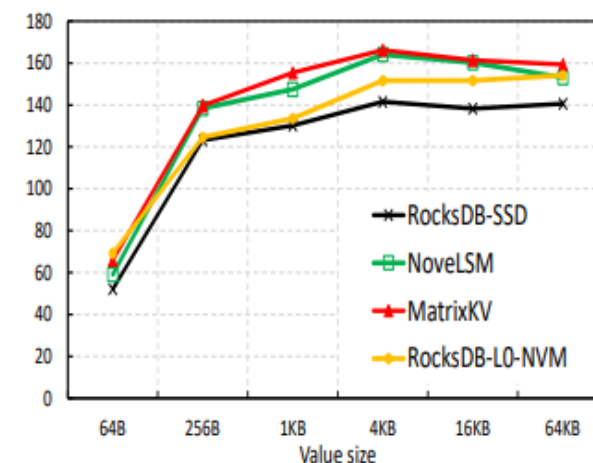
(a) Random write



(b) Sequential write



(c) Random read



(d) Sequential read

1. 随机写场景下，性能提高明显，而顺序写反而不如原本的rocksdb
2. 在读场景下性能提升不明显

目录

C O N T E N T S

01. NVM介绍

02. NVM对B+树的优化思路

03. NVM对LSM-tree的优化思路

04. 总结

1. 将NVM应用于B+树关注点在于提升其写性能和降低在NVM上的一致性
2. 将NVM应用于LSM-tree关注点在于解决LSM-tree长期以来的痛点，写停顿、读写放大等
3. 存储引擎有了更多的存储资源和计算资源，应当朝着简化的方向发展

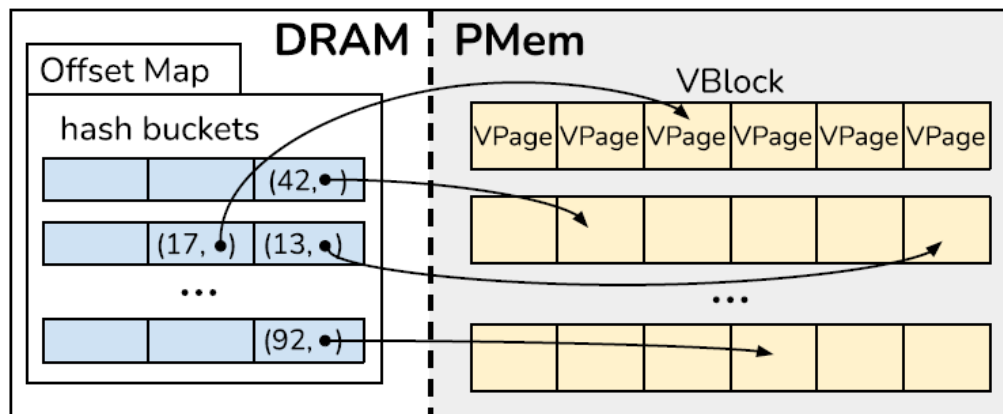


Figure 3: Viper's architecture. VPages store key-value records in PMem (right). The Offset Map stores (key, record-offset) entries in a volatile hash index (left).

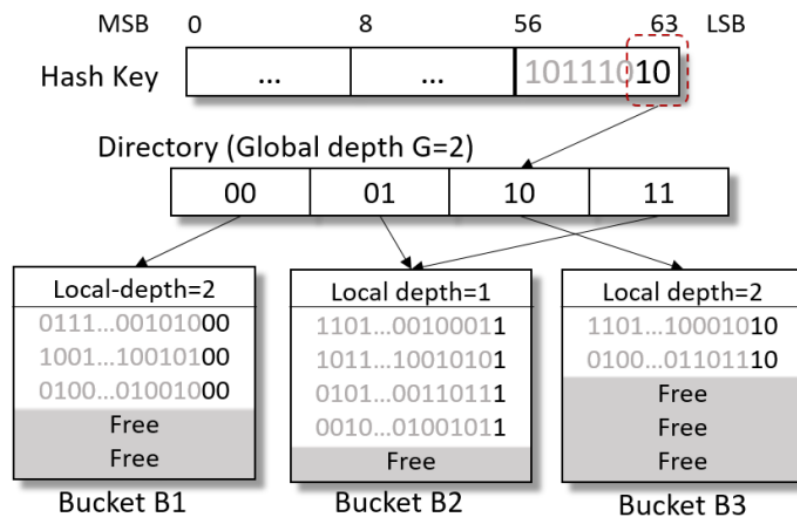



Figure 1: Extendible Hash Table Structure



谢谢