



# 基于SSD的LSM-tree键值分离

---

2022.5.13

李林峰



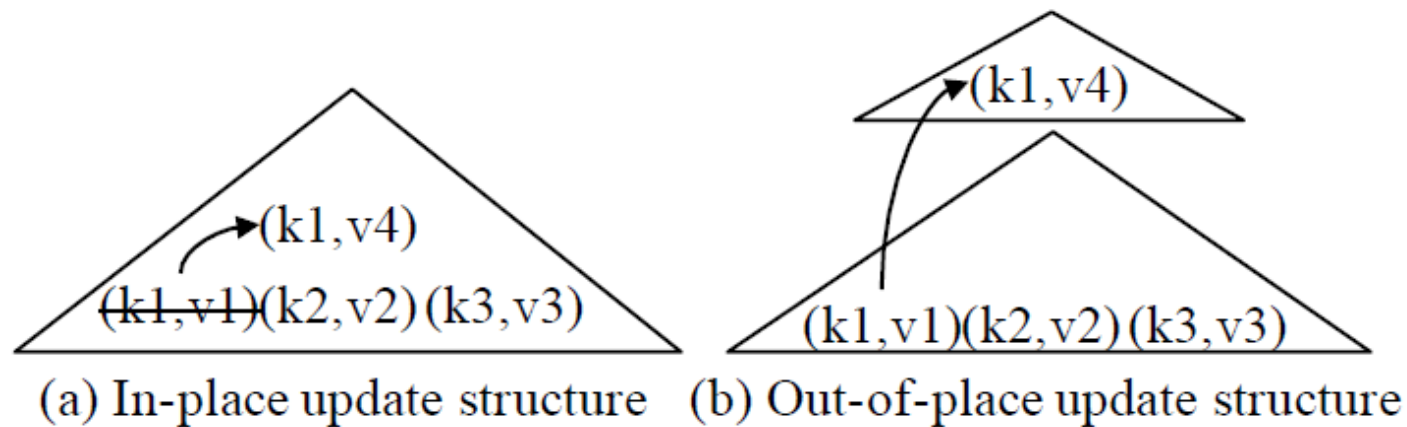
# 目录

## C O N T E N T S

1. 背景
2. KV分离策略
3. 总结

# 背景

数据存储引擎的两种存储结构。 In-place / out-of-place update



**就地更新**（例如B+树）直接**覆盖旧记录**以存储新的更新，如图a，

- 仅存储每个记录的最新版本。
- 更新会导致**随机I/O**（树形结构还涉及平衡性的维护）
- 写性能差、读性能好

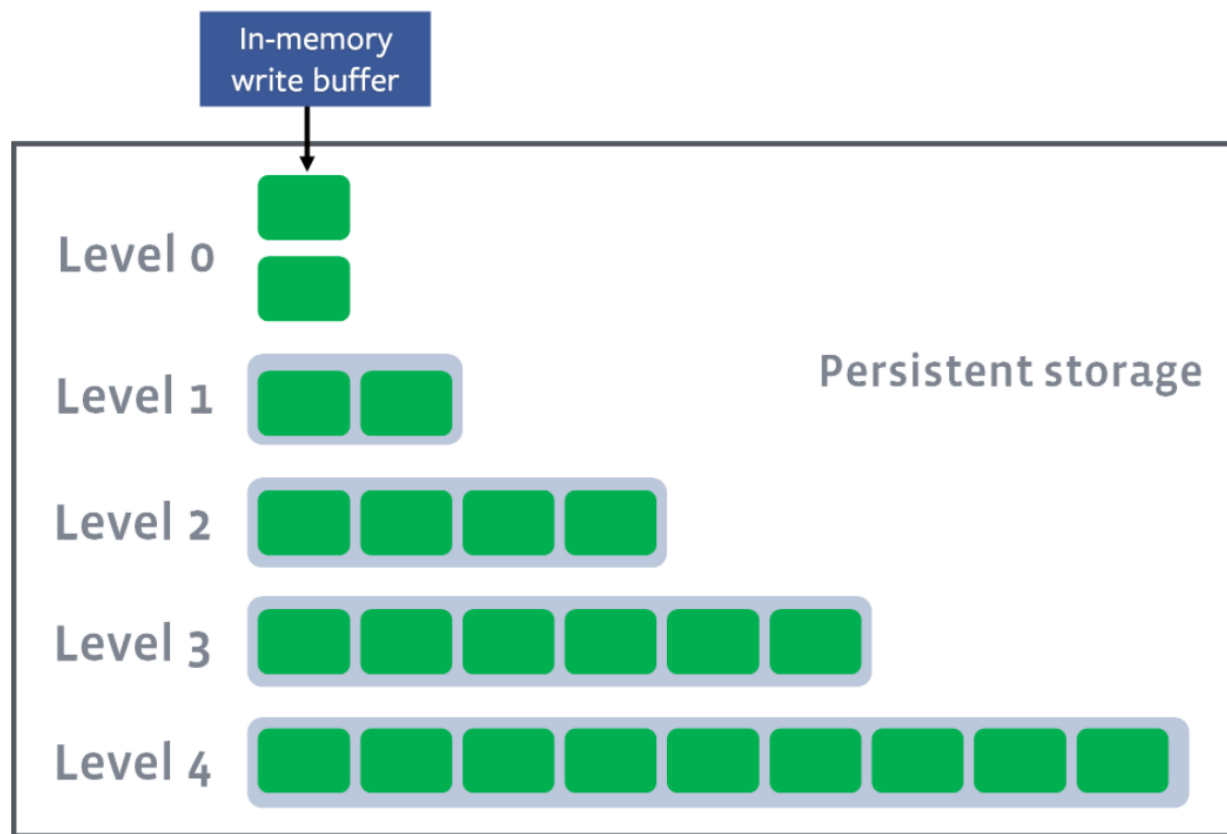
**异地更新**始终将**更新存储到新位置**，如图b，

- 利用**顺序I/O**来处理写操作。
- 不覆盖旧数据，记录可能存储在多个位置中的任何一个位置。
- 写性能好、读性能差

# LSM

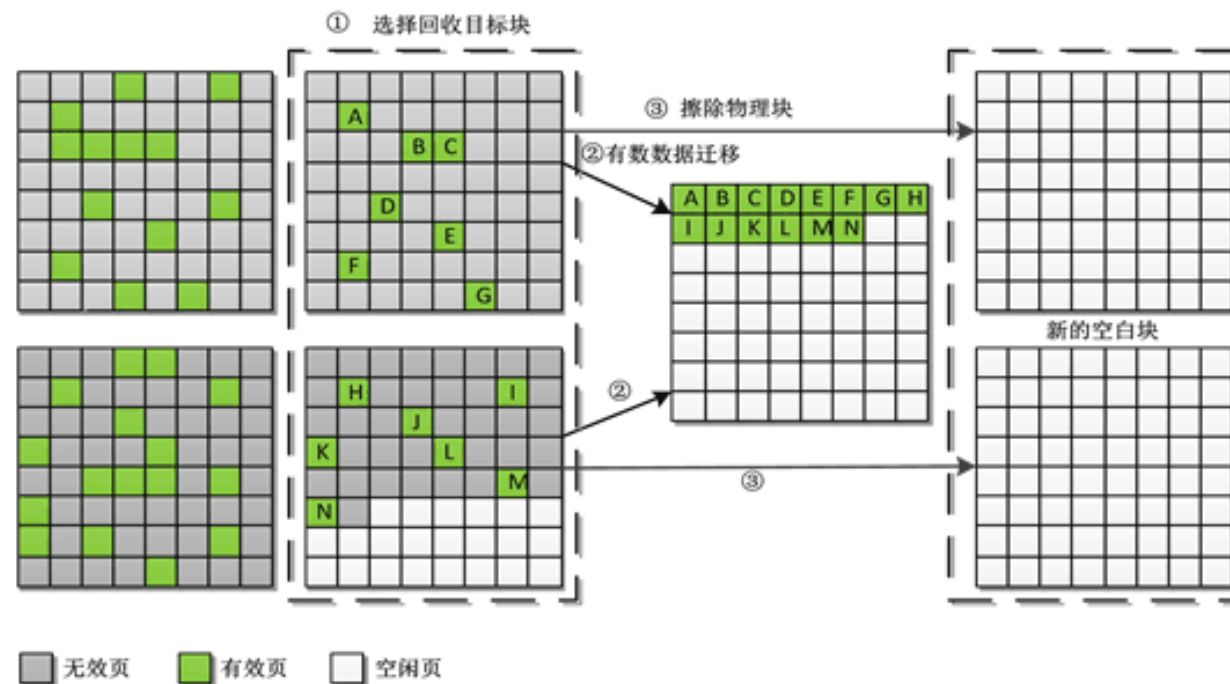
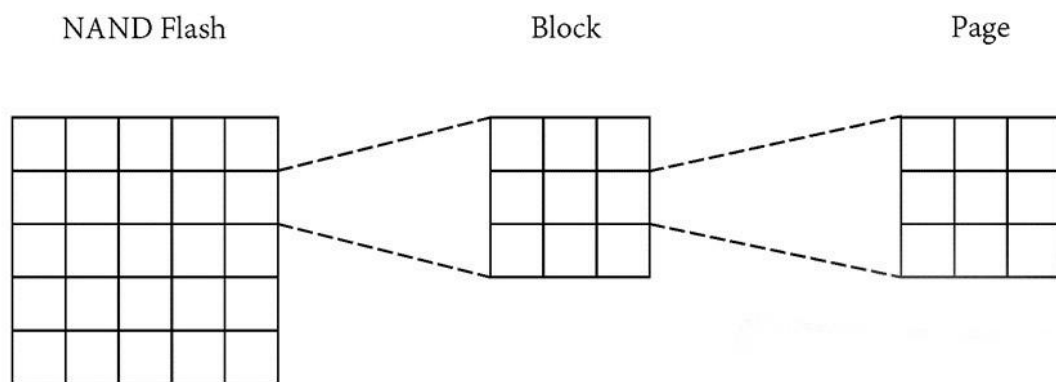
LSM-tree (log-structured merge tree) 如今被广泛用作现代 NoSQL 键值存储的存储层。  
**采用异地更新范式 (out-of-place) 以实现快速写入。**

1. HDD上顺序写的性能远远好于其随机写性能。LSM Tree 将随机写转化为顺序写，提升在HDD上的性能
2. 存在严重的读写放大、空间放大



1. SSD随机和顺序性能之间的差异远没有硬盘那么大,执行大量顺序I/O以减少后来的随机I/O可能会不必要地浪费带宽
2. SSD具有很大程度的内部并行性
3. SSD可能会通过重复写入而磨损, LSM tree中的高写入放大会显著降低设备寿命
4. SSD特有的写入擦除循环和昂贵的垃圾回收, 过量的随机写有害

	顺序读	随机读	顺序写	随机写
Samsung SM951 NVMe	1973.5MB/s	49.4MB/s	1155.3MB/s	140.4MB/s
		40倍		8倍
Seagate ST30000DM 001	200.7MB/s	0.6MB/s	135.1MB/s	1.0MB/s
		334倍		135倍



1. NAND Flash分成多个block, block分成多个page
2. 擦除的最小单位为block
3. 多个Block废弃page达到一定的阈值, 就启动垃圾回收
4. 随机写会造成写放大和空间放大、I/O放大

垃圾回收

1. Key 放在 LSM 树里面，Value 放在分离开的 Value log 中
2. compaction 时，只会对 Key 进行排序和重写，不影响 Value log，显著降低了写放大
3. Value 分离后，LSM-Tree 本身大幅减小，可以减少查询时从磁盘读取的次数，并且可以更好的利用 cache
4. Value log 需要定期进行 GC，开销较大



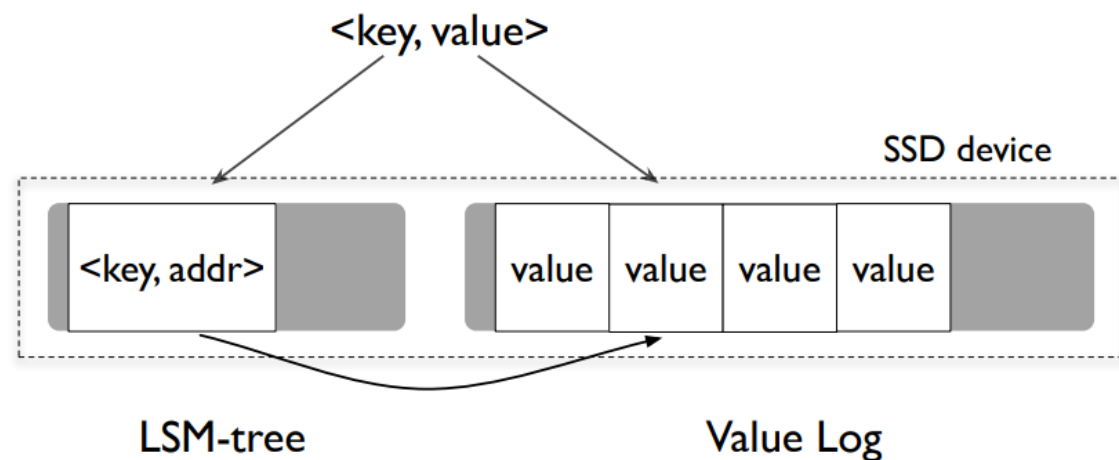
# 目录

## C O N T E N T S

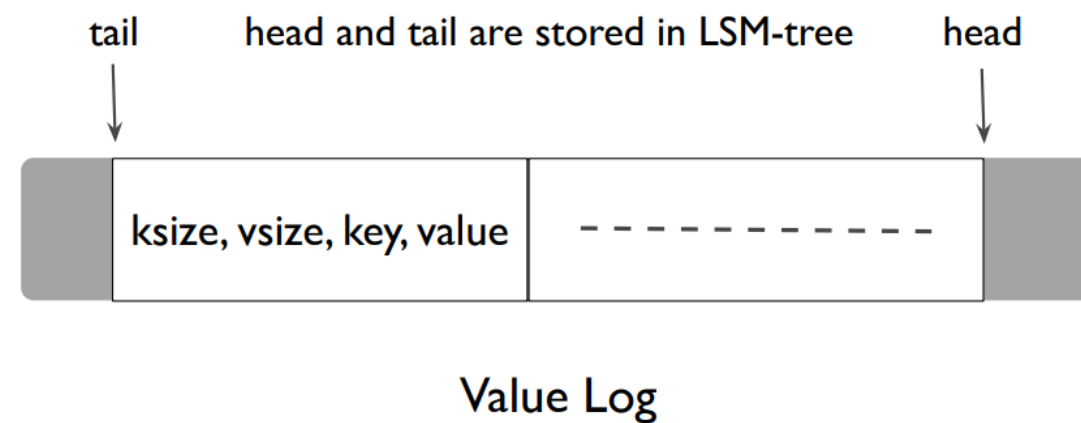
1. 背景
2. KV分离策略
3. 总结



1. value放在value-log 文件中
2. 不写WAL, 以  $\langle \text{vLog-offset}, \text{value-size} \rangle$  形式插入 LSM 树中
3. Compaction:只需将无效key从 LSM 树中移除, 不需要修改vLog
4. 范围查询:利用内部线程池并行读取 value 数据。



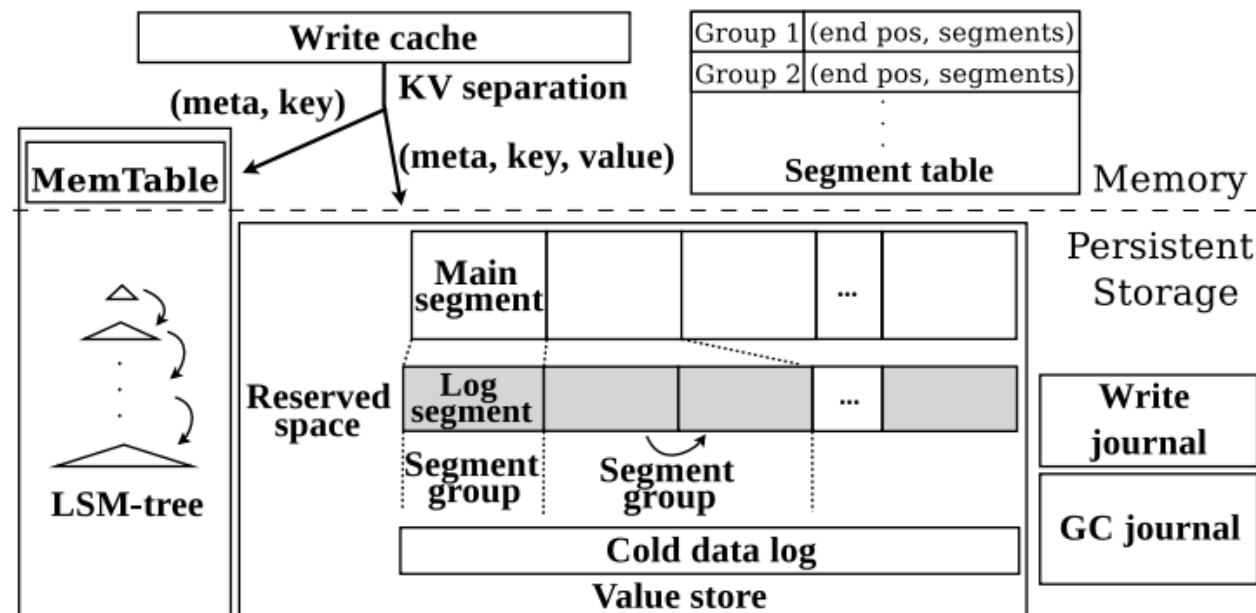
1. Value log: 循环队列, tail至head之间部分包含所有value
2. GC: 依次读取一个键值对块, 返回 LSM 中检查value有效性, 将有效值写回vLog 的head
3. GC完成后将新的 value 地址和 tail 指针地址写回到 LSM-Tree 中

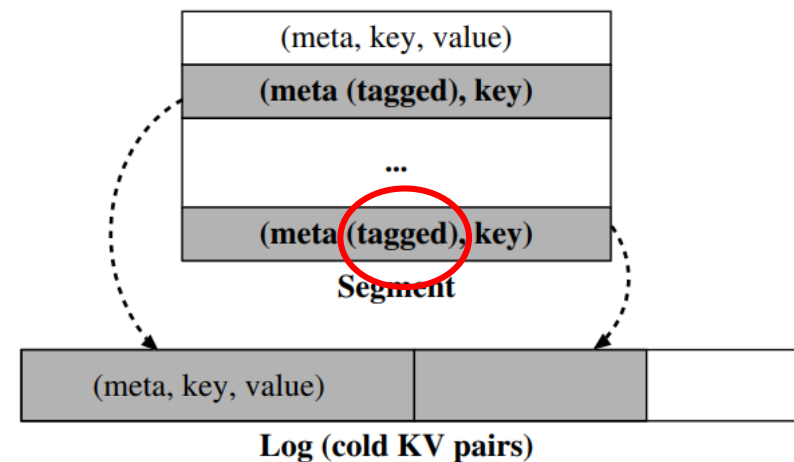
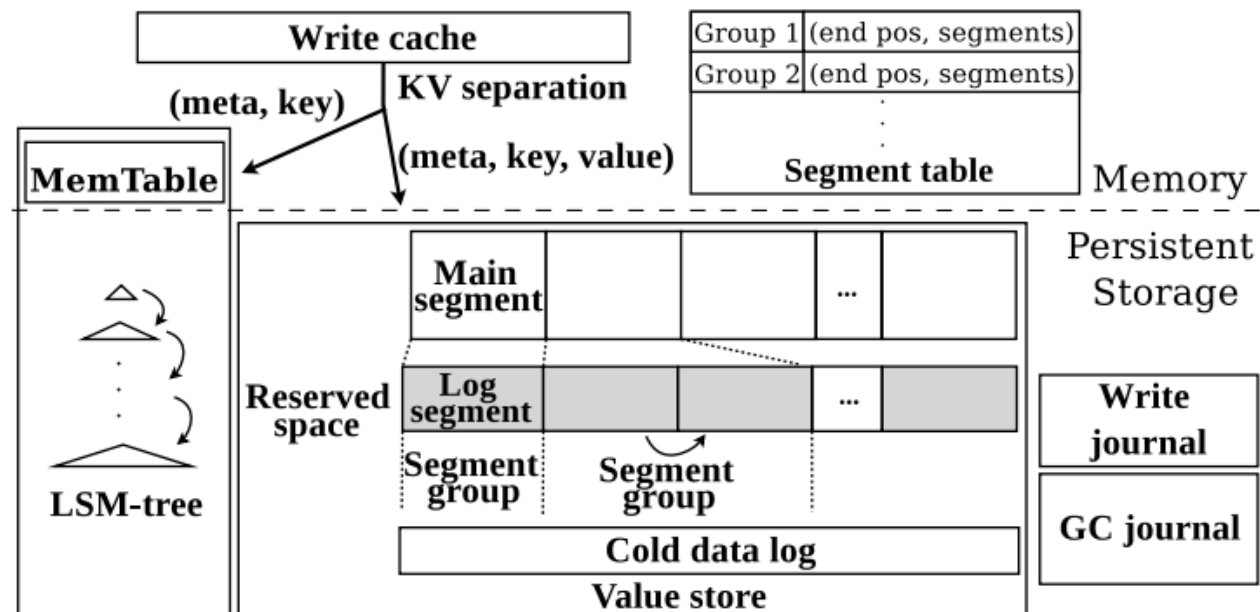


1. 严格的 gc 顺序, 从 head -> tail, 造成很多不必要的写放大
2. GC时通过LSM-tree 判断value 有效性, 查询开销不容忽视。
3. GC完成后还需要将最新的value指针更新到LSM-tree, **导致大量不必要的数据重定位**
4. 对于小value的范围查询性能较差

## 部分KV分离

1. 小value放在LSM中，大value写入value log
2. 小value对整个系统的平均写放大都影响很小
3. 改善它的一个查询性能
4. 减小GC开销



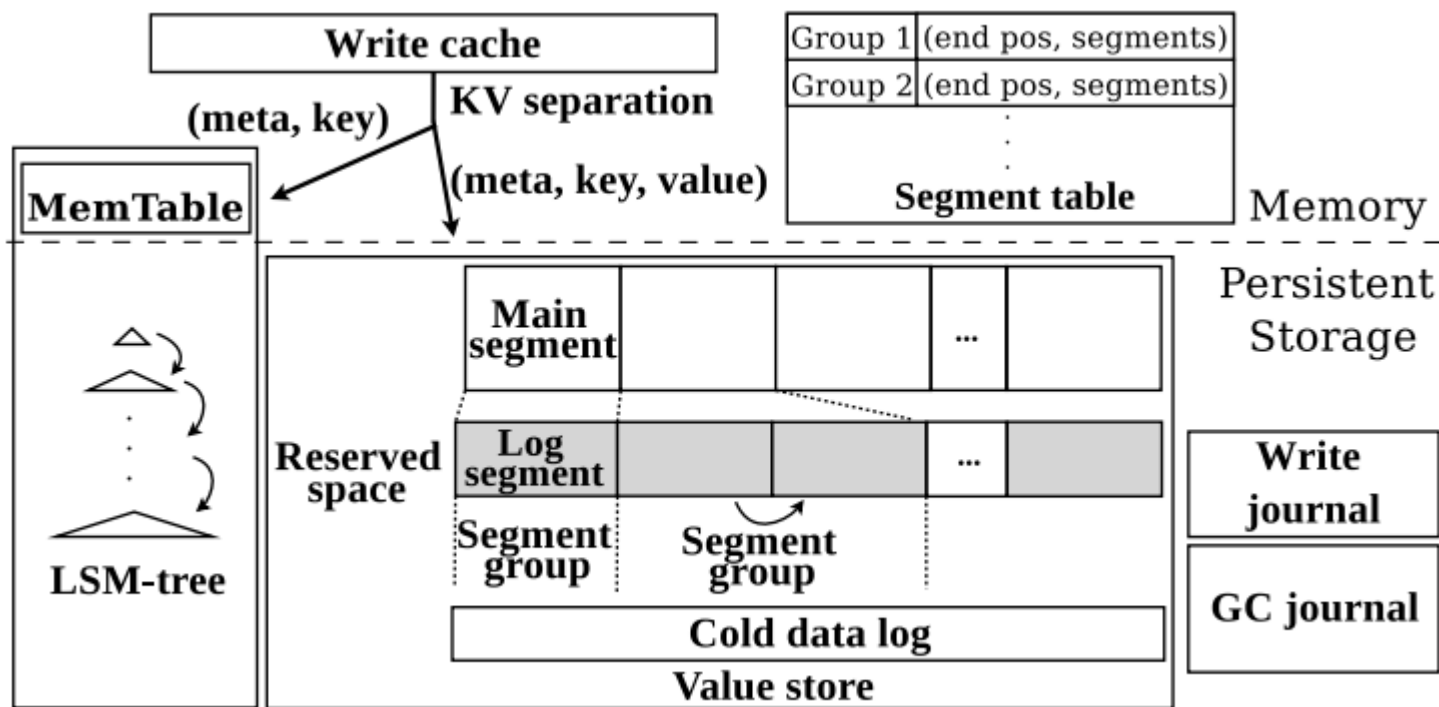


## 减小写放大方法:

1. 对key进行hash将value放在不同partition
2. 对value进行冷热分离，最后一次插入以来至少更新过一次的KV对视为热数据

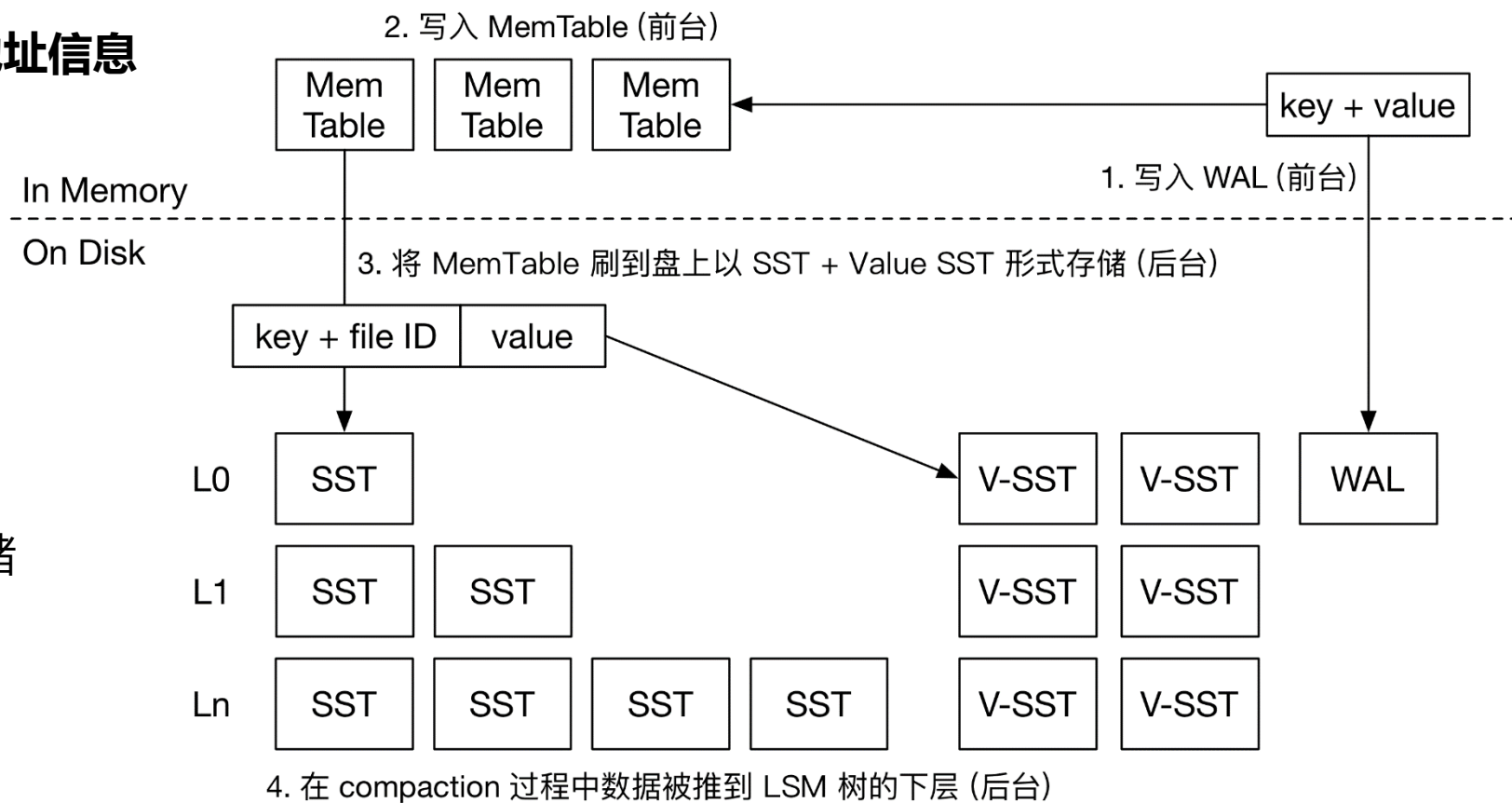
## Garbage Collection

1. 以 segment group 为单位, 选择写入量最大的segment
2. 检查KV有效性: 不需要查找 LSM-tree
3. 构造临时内存哈希表(按键索引)来缓冲在段组中找到的有效KV对的地址
4. Cold data log进行GC, 与wiskey一致
5. 更新 LSM-tree中的地址信息



## 设计亮点: GC时无需更新LSM-tree地址信息

1. value 需要在前台被写入 WAL
2. 部分KV分离: 大 value写入 v-SST 中
3. V-SST 按 key 排序, LSM tree仅存储 <key, fileno>

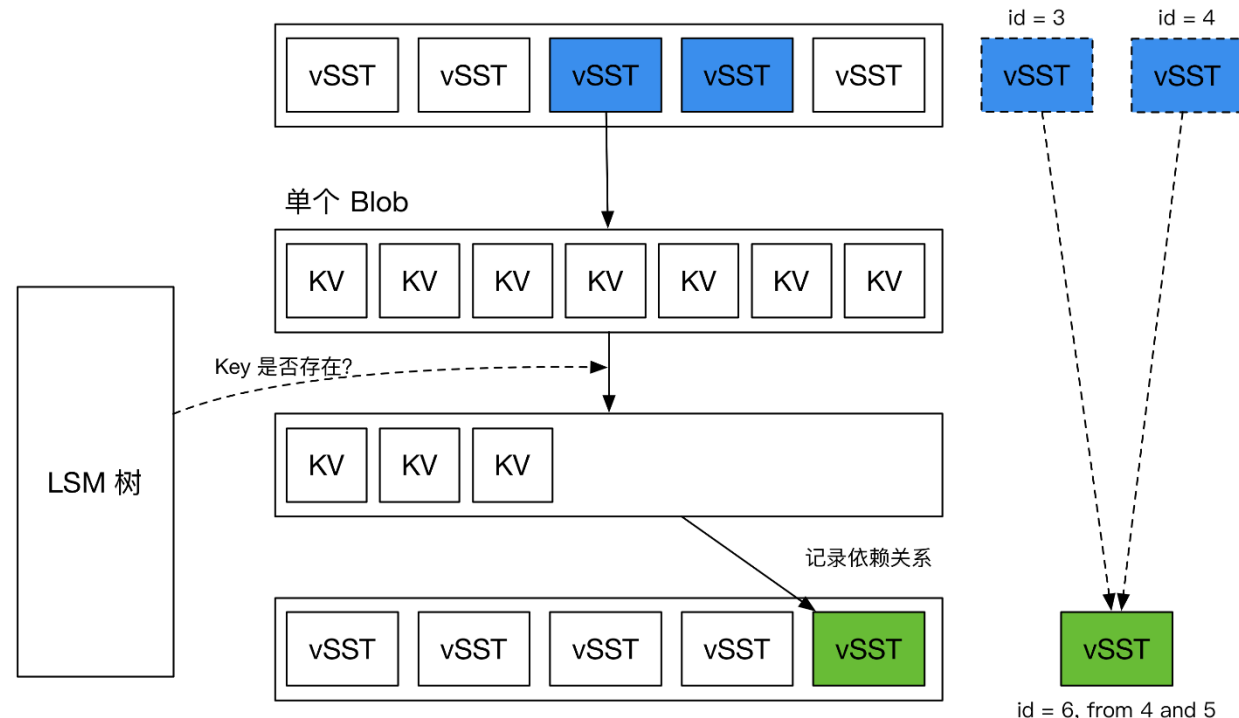


1. GC 无需更新LSM tree, 在 MANIFEST 中记录 v-SST 之间的依赖关系, 降低了 GC 对用户前台流量的影响
2. compaction 过程中顺便更新 value 位置信息

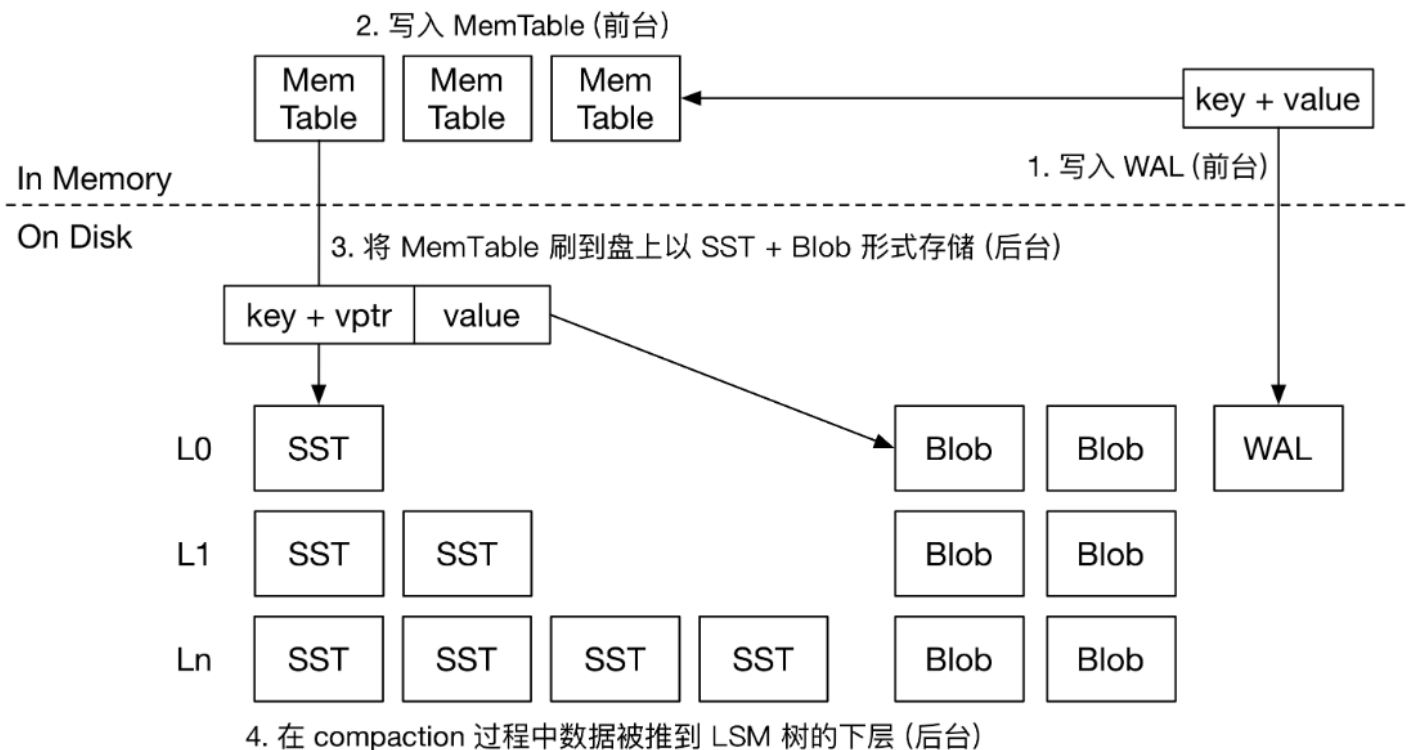
1. 选择需要回收的 vSST

2. 扫描 LSM 树并写新的 Blob

3. 记录 vSST 之间的依赖关系  
MANIFEST







## 设计亮点：特有的GC方式level merge

1. 部分KV分离，存储形式：  
<key, <fileno, offset>>
2. Blob file中的value有序

vptr	
File No.	Offset

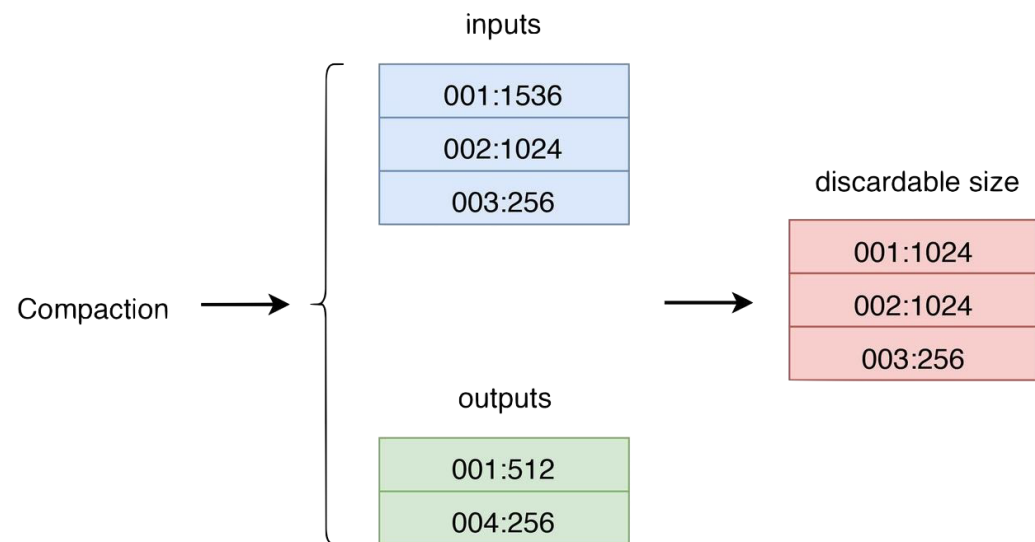
SST

key - index {001:0000:512}
key - index {002:0000:1024}
key - value
key - index {001:2873:1024}
key - index {003:1752:256}

BlobFileSizeProperties

001:1536
002:1024
003:256

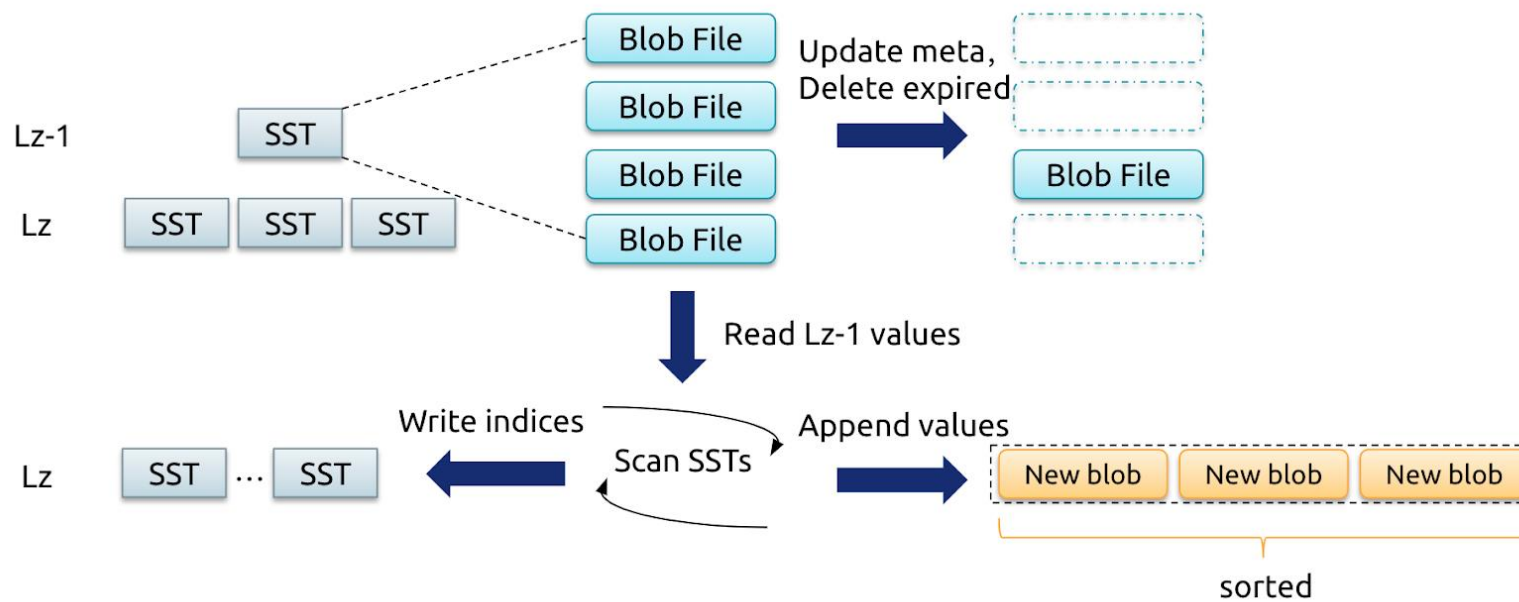
BlobFileSizeCollector



## 传统GC

1. 使用BlobFileSizeProperties和discardable size来收集 GC 所需信息
2. 每个BlobFile在内存中维护一个discardable size 变量
3. 需要去LSM-tree查询value有效性, GC完成后需要写回SST

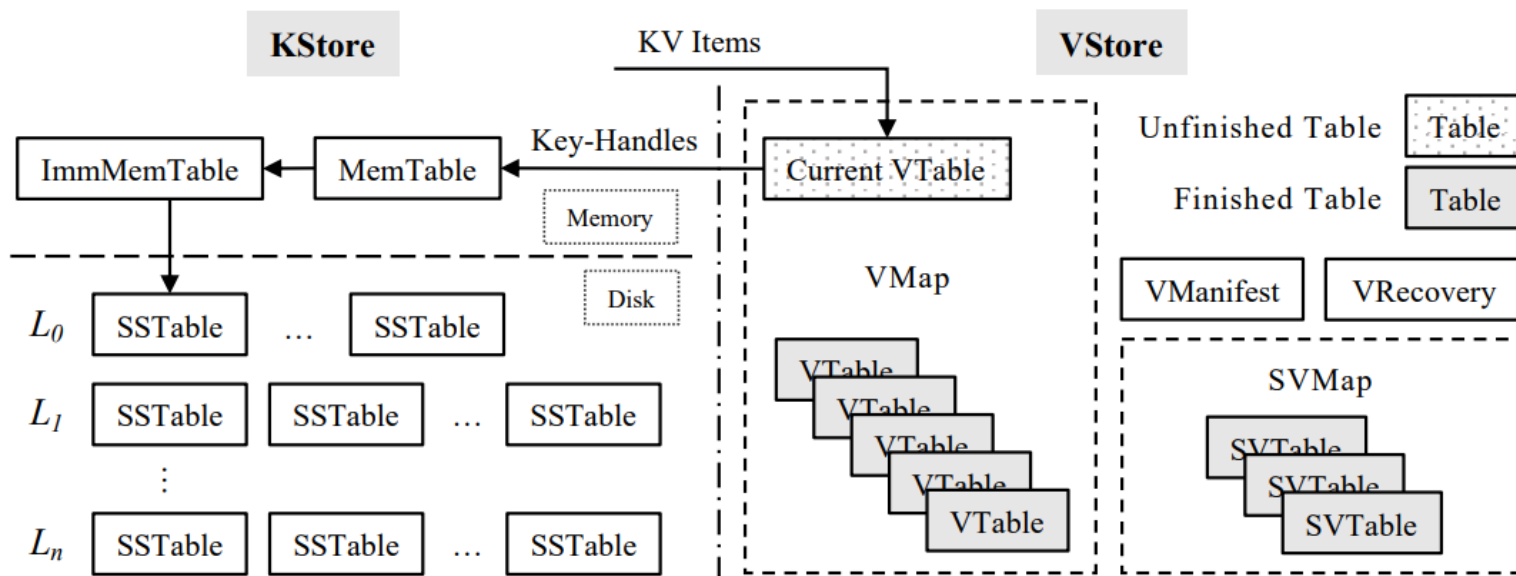
## Level merge



1. Compaction的同时进行GC
2. 省去了有效性检查和更新SSTable
3. 造成写放大, 仅对 LSM-tree 中 Compaction 到最后两层数据对应的 BlobFile 进行 Level Merge

## 设计亮点:

1. Vstore分为Vtable和Svtable, 新写入的数据写入Vtable
2. GC的时候, 利用DropKeys来检查kV的有效性
3. KV被读取的时候再更新GC后的 epoch, file offset, item size



SSTable:

Key:key+sequence number

Value:< file num, epoch, file offset, item size >

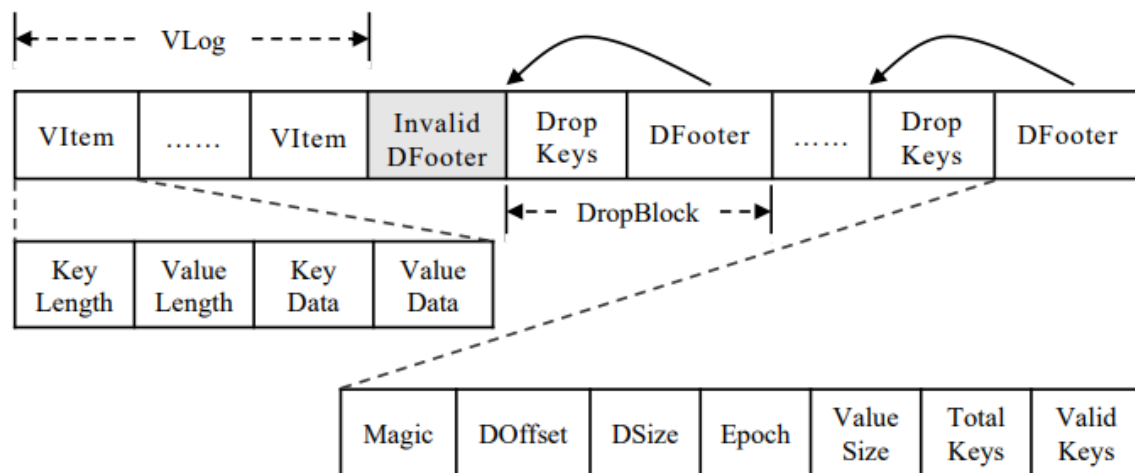


Fig. 2. The VTable Format.

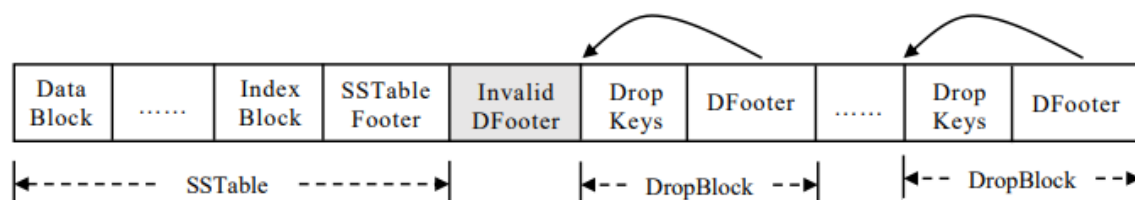


Fig. 3. The SVTable Format.

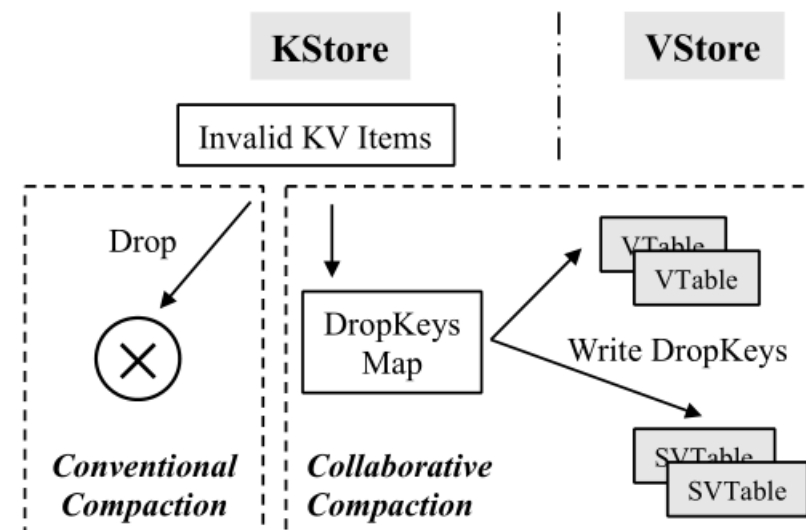


Fig. 5. Collaborative Compaction.

1. VTable: 顺序写入  
SVTable: 便于查询
2. 在key的尾部添加一个sequence number
3. Compaction: 创建一个map  
map key : file number;  
map Value : DropKeys组成的vector

## GC流程:

1. 选择valid rate最小的VTable或者SVTable进行GC
2. 从后往前读取Drop keys,并将其全部插入hash set
3. 遍历KV Item, 与hash set做对比, 将valid value写入新的SVTable (memtable)
4. 将memtable刷盘 (SSTable), 并在末尾添加一个invalid Dfooter
5. GC后的结果当这个KV被读取的时再更新 (通过epoch判断)

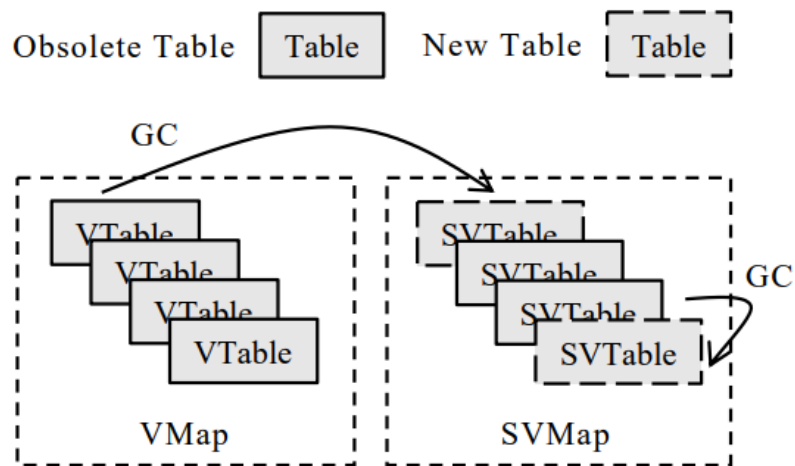


Fig. 6. The VStore Garbage Collection.

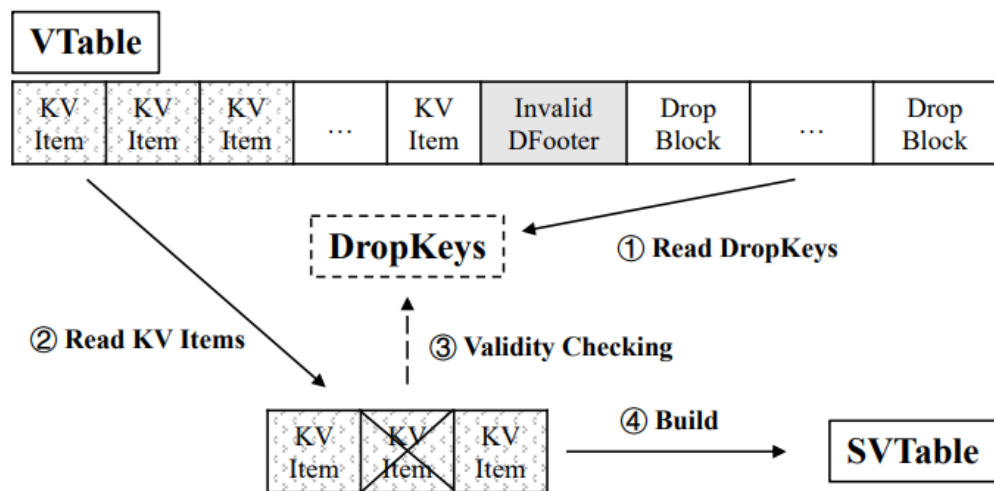
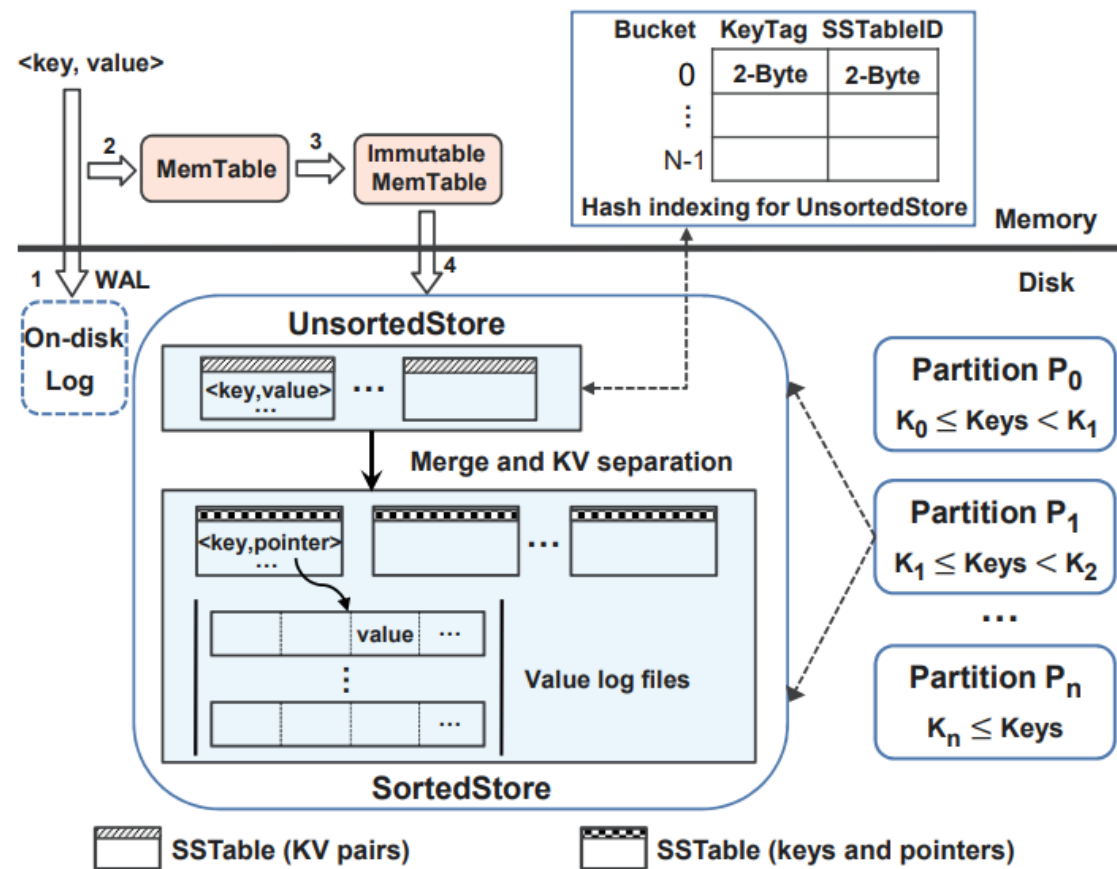


Fig. 7. The VTable Garbage Collection.

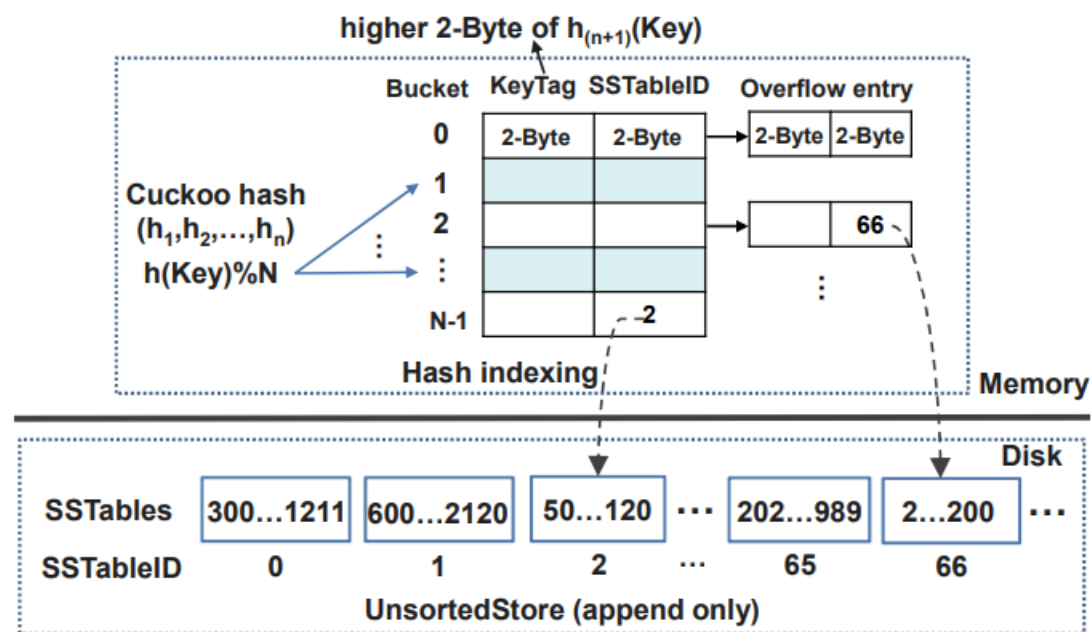
优化点：采用分层体系结构实现差异化数据索引  
保证 read/write/scan/Scalability 的高效

1. UnsortedStore: 内存刷回的数据, 无序  
SortedStore: Unsorted Store合并后的有序的数据
2. 内存 HASH 索引来支持快速的读写
3. SortedStore采用KV分离



## Hash Indexing

1. 轻量级的两级 HASH:  
cuckoo hashing+linked hashing
2. 索引项:<keyTag, SSTableID, pointer>
3. 写入: 从  $h_1$  一直到  $h_n$ , 直到找到空的bucket, 否则生成一个overflow entry添加到对应的桶  $h_n(\text{key})\%N$  后
4. 读取: 从  $h_n$  一直到  $h_1$





## KV分离

1. 元信息:  $\langle \text{partition}, \text{logNumber}, \text{offset}, \text{length} \rangle$
2. 动态范围分区: 根据Key大小进行拆分, 改善范围查询
3. GC: 需要遍历Sortedstore, 判断有效性。之后还需要将元信息写回sortedstore、

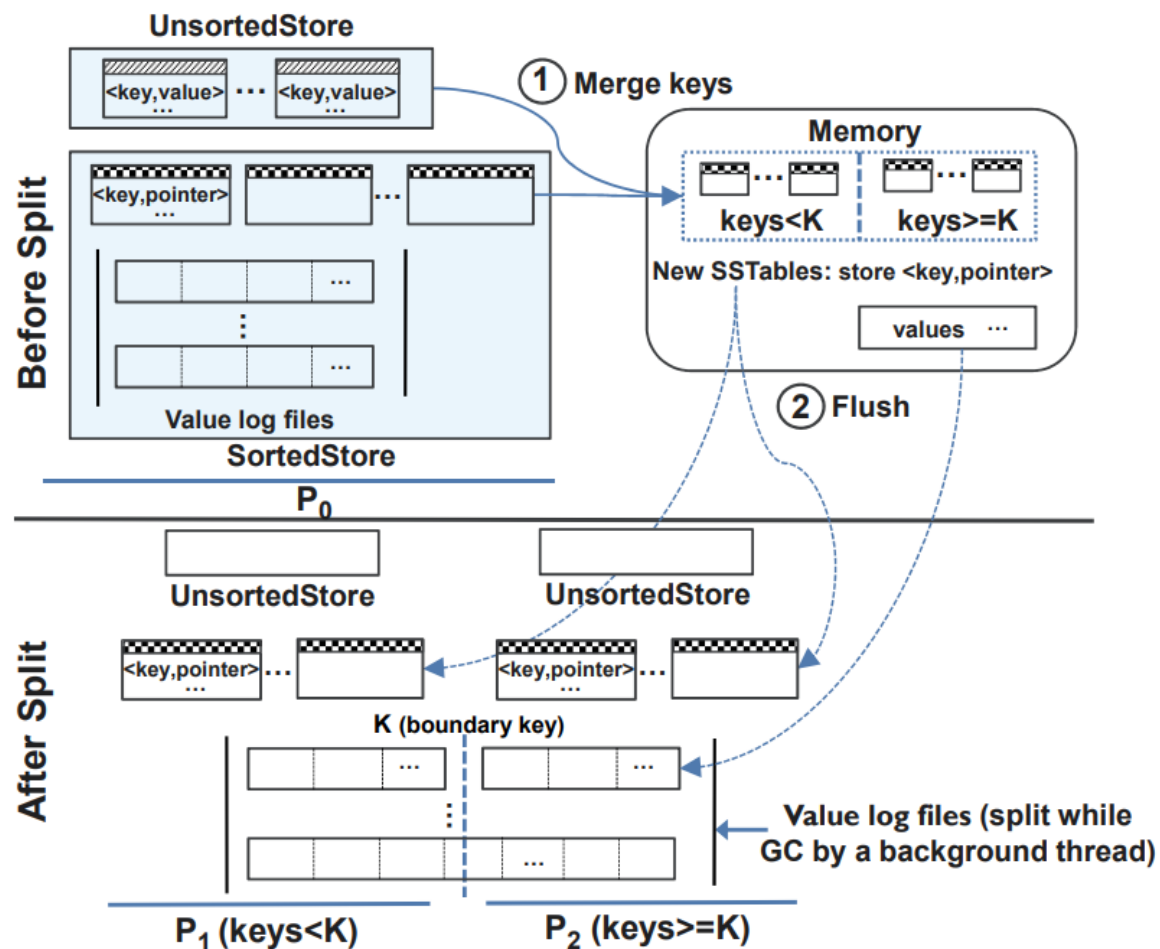


Fig. 8: Dynamic range partitioning.



# 目录

## C O N T E N T S

1. 背景
2. KV分离策略
3. 总结

## KV分离改进措施:


1. 部分KV分离, 在不影响系统整体写放大的情况下, 减少GC开销, 提高范围查询性能 (HashKV)
2. 对key进行hash、或按key的大小将value放在不同partition, 减小写放大 (HashKV, UniKV)
3. 冷热数据分离, 减小冷数据GC造成的写放大 (HashKV)
4. 通过 MANIFEST 记录GC前后文件之间的依赖关系, 推迟至compaction 过程中更新value 位置信息 (TerarkDB)
5. 在最后1~2level, Compaction的同时进行GC, 省去了有效性检查和更新SSTable (Titan)
6. Compaction时将Dropkey写入VStore, GC时利用DropKeys来检查kV的有效性 (NovKV)
7. KV被读取时再更新SSTable (NovKV)

[01] - Lanyue Lu, Thanumalayan Sankaranarayanan Pillai, Hariharan Gopalakrishnan, Andrea C. Arpaci-Dusseau, and Remzi H. Arpaci-Dusseau. 2017. WiscKey: Separating Keys from Values in SSD-Conscious Storage.

[02] - Helen H. W. Chan , Yongkun Li , Patrick P. C. Lee, 2018. HashKV: Enabling Efficient Updates in KV Storage via Hashing

[03] - Chen Shen, Youyou Lu , Fei Li, Weidong Liu, Jiwu Shu . 2020. NovKV: Efficient Garbage Collection for Key-Value Separated LSM-Stores.

- [04] - Qiang Zhang; Yongkun Li; Patrick P. C. Lee; Yinlong Xu; Qiu Cui; Liu Tang. 2020. UniKV: Toward High-Performance and Scalable KV Storage in Mixed Workloads via Unified Indexing
- [05] - Giorgos Xanthakis, Giorgos Saloustros, Nikos Batsaras, Anastasios Papagiannis, and Angelos Bilas. 2021. Parallax: Hybrid Key-Value Placement in LSM-based Key-Value Stores.
- [06] - Yongkun Li and Zhen Liu, Patrick P. C. Lee, Jiayu Wu, Yinlong Xu, Yi Wu, Liu Tang, Qi Liu, and Qiu Cui. 2021. Differentiated Key-Value Storage Management for Balanced I/O Performance



谢谢