LSM-DB

前言

LSM-DB是我大二下做的一个课程项目，在当时的我来看非常的困难（~~迟交子整整三周~~），Debug更是De到神情恍惚，吃不下睡不好。一个二进制文件的bug折磨了我好几天，我从几百万行日志里面一行一行地找错误，最后发现是打开方式的错误，没加 `std::ios::binary` 。真的，这辈子再也不想经历一遍了😇

值得一提的是，LSM Tree的实验报告是我用LaTeX写的，而这也是我第一次使用LaTeX这种高大上的工具（原来一直以为只有写论文才需要用到）。在写实验报告的过程中，LaTeX丝滑的排版体验让当时正在code rush的暴躁的我也忍不住啧啧称奇，感慨这世界上居然有这么好用的文档工具。

LSM-DB是我很难忘的一段项目经历，各种意义上。

现在把它翻出来是因为我想找日常实习了，心仪的工作岗位和Cloud、DB强相关。2023以来我基本一直在做Cloud相关的项目，Minik8s和Satellite-SDN都拿得出手，可以放在简历里面讲。但是，我发现自己几乎没有DB相关的项目，除了一年半以前做的LSM-DB🤡（好好好，最终还是你院的项目救了我的狗命），于是就只能把它翻出来深挖了。

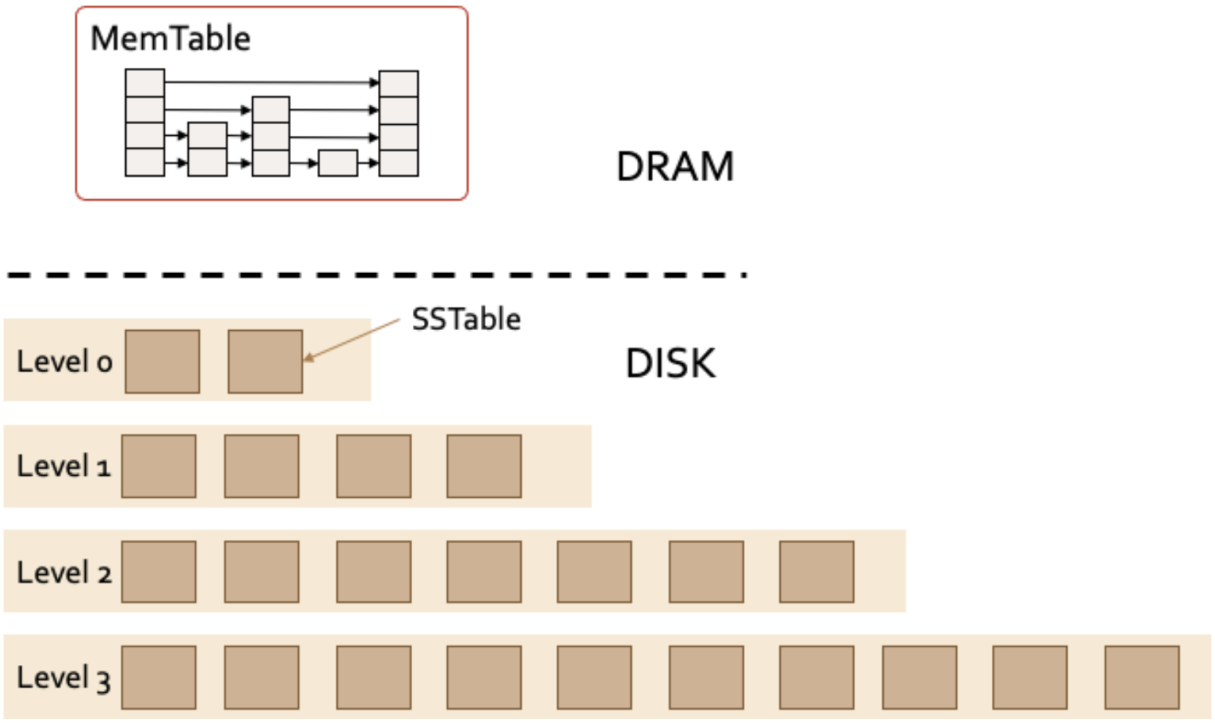
在这里先祈愿一个面试顺利，复习LSM-DB之后offer多多，大厂独角兽公司抢着要我🙏🙏🙏

Overview

LSM- DB是一个键值存储系统，支持GET、PUT、DEL、SCAN操作。

LSM-DB的架构如下所示，主体由DRAM中的MemTable和Disk中的SSTable组成。

此外，SSTable的元数据、索引、布隆过滤器在DRAM中有缓存，用于加速查找等操作。



MemTable

MemTable可以通过跳表、AVL树、红黑树等数据结构实现，我使用了跳表实现了MemTable

每个键值对的层数由课程组给出的随机数生成函数给出

当MemTable的容量将要超过2MB时，会触发Compaction操作。Compaction的模式有很多，我们这里使用的Level Compaction就是其中的一种，这个我们后面再做介绍

我们先来看一下跳表··

跳表（Skiplist）是一个特殊的**链表**，相比一般的链表，有更高的查找效率，可比拟二叉查找树，平均期望的**查找、插入、删除时间复杂度都是O(logn)**，许多知名的开源软件（库）中的数据结构均采用了跳表这种数据结构。

- Redis中的有序集合zset
- LevelDB、RocksDB、HBase中Memtable
- ApacheLucene中的TermDictionary、Posting List

跳表的高度控制策略有以下两种：

1. 限制最大高度：`h=max{10, 3*RoundUp(logn)}`

2. 不限制最大高度：插入算法中不限制跳表的最大高度，以随机函数的值来确定是否继续在Tower上Append Entry

时间与空间分析：

最坏场景是每个元素的Tower的高度都是h的时候，查找、插入、删除的性能最差，为O(n+h)，其中n为跳表的元素数目，h为跳表的高度

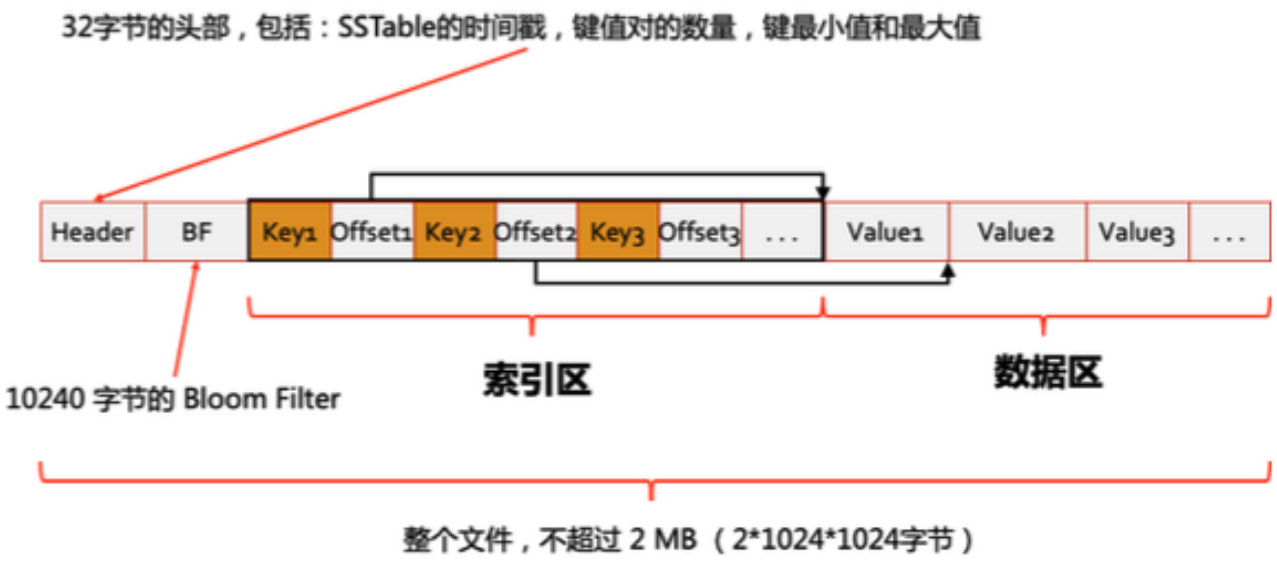
在正常情况下，跳表的高度为O(logn)，增删改查的时间复杂度为O(logn)，占用的空间为O(n)

> 知乎给出的和红黑树的比较：
>
> 1. 跳表和红黑树的插入、删除、查找效率都是O（logN），都可以顺序遍历所有元素（红黑树通过中序遍历）。红黑树更稳定一些，**跳表恶化是存在概率的，虽然概率极低**。
> 2. 跳表实现简单，但是**浪费了很多空间**。红黑树实现麻烦，但是没有使用额外的空间。
> 3. 跳表**区间查找**很方便，redis中的zset实现区间查找就是用的跳表。

> 面试可能会出现的点：
>
> 手写SkipList、和AVL树/红黑树/B树/B+树的比较、时间空间复杂度

SSTable

SSTable由Header、Bloom Filter、索引区、数据区构成



1. Header：元数据，包括时间戳、键值对数量、键的最小值和最大值
2. BloomFilter：相当于一个哈希表。我们使用课程组给出的哈希函数，得出一个128bit的值，将其切分成4个32bit的值使用，分别设置对应位置的bit（要模8*10240）。这样子我们在查询一个key是否在该SSTable中时，便可以通过哈希函数对key进行hash，然后判断Bloom Filter中各个位是否被设

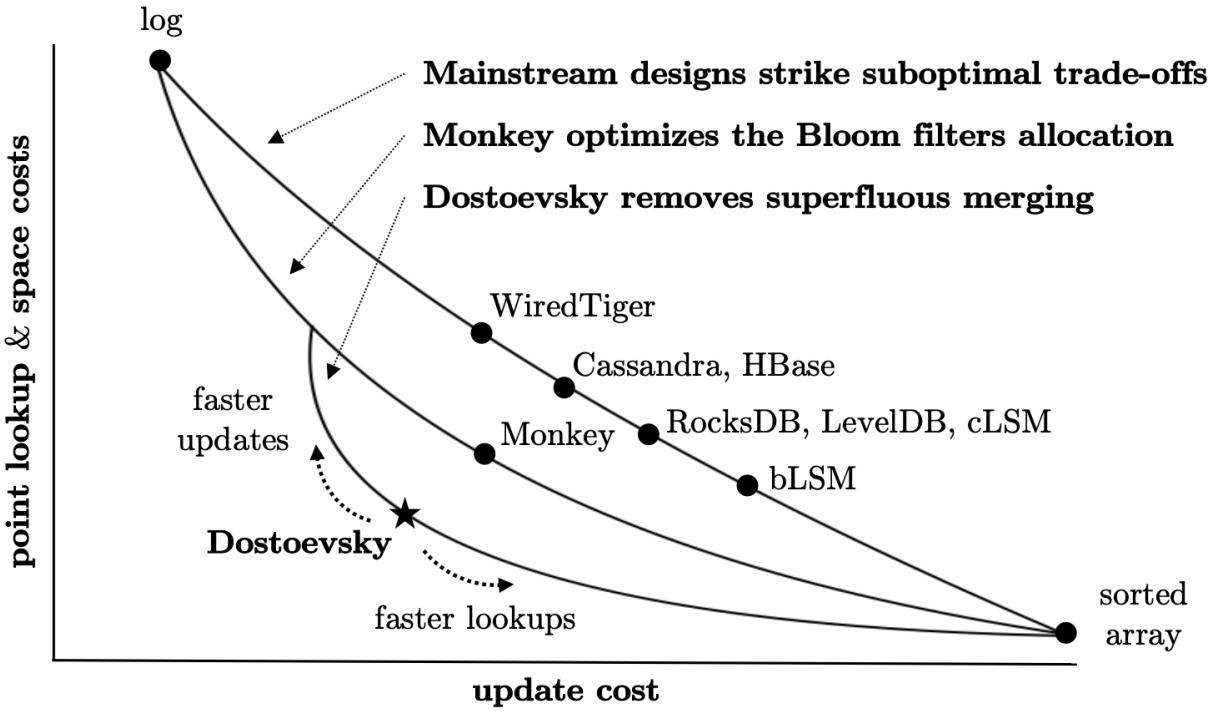
置。如果被设置了，不一定有；但是如果没有被设置，那一定没有。这个结构大大加速了**判断key是否在SSTable中**这一逻辑。

3. 索引区：存放着key和它在SSTable中的offset，即数据开始区域和SSTable文件头的距离，我们在判断key在SSTable中后，可以通过二分查找去查找key（最终确定key是否在SSTable中，有则能得到offset，若无则返回）
4. 数据区：存放着字符串数据

Header、BloomFilter、索引区在内存中有缓存，用于加速查找

Compaction

我们在项目中用的是Level-Compaction，实际上compaction有很多种，比较常见的两种分别是Level Compaction和Size/Tier Compaction。Level Compaction的代表是Facebook的RocksDB，Size/Tier Compaction的代表是Cassandra，他们的性能如图所示：



Level Compaction

当内存中MemTable数据达到阈值时，要将MemTable写入磁盘

1. 首先将MemTable转换成SSTable的形式，然后直接写入到level0中
2. 然后判断Level0中SSTable的数量是否超过上限（Level n中SSTable数量不能超过2^(n+1)），若超过则将Level 0中所有的SSTable和Level 1中的部分SSTable合并（key值有交集的SSTable合并，时

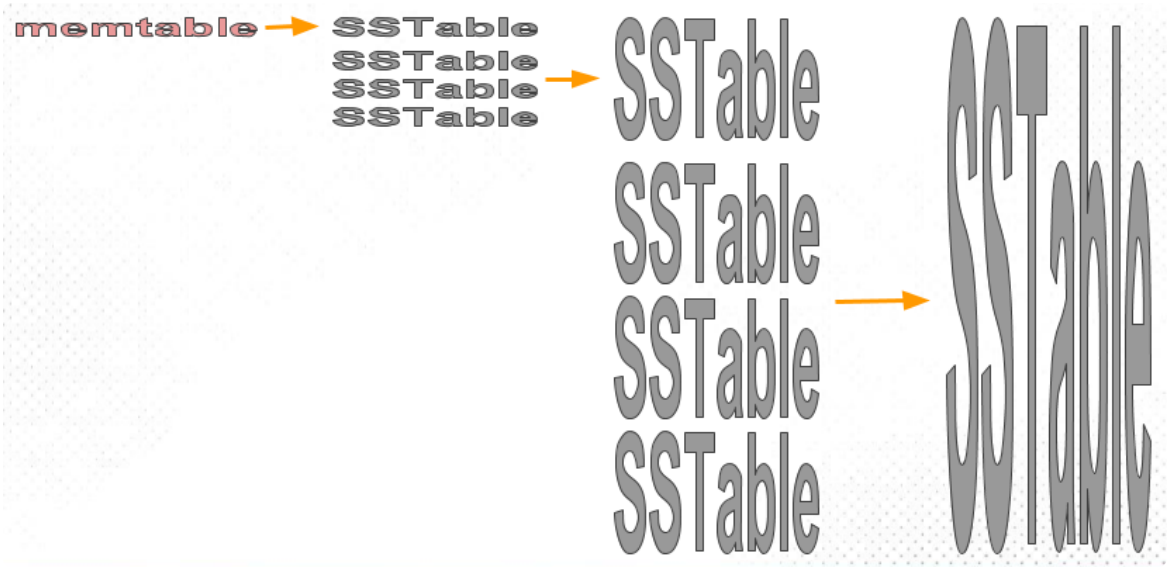
间戳选最大的）

3. 若合并后Level k中的SSTable数量超过了Level k能容纳的上限，则选择时间戳最小的若干个文件（时间戳相等则选择键最小的文件），然后以同样的方式Level k+1合并（k >= 1）
4. 更新SSTable在DRAM中的缓存

> Level compaction目标就是维持每个level都保持住**one data sorted run**
>
> Each run contains data sorted by the index key. A run can be represented on disk as a single file, or alternatively as a collection of files with non-overlapping key ranges.

Size/Tier Compaction

memtables 首先会不停flush 到第一层很小的sstables，等到sstable数量够了（图里4个），compaction成一个sstable写到下一层，下一层sstable写满4个，再compact，如此以往



这种compaction的方法，能够保证每个sstable是sorted，但是不能保证每一层只有一个Run

Comparison

KV数据库三大指标：Read/Write/Space Amplifier（读放大、写放大、空间放大）

1. Write Amplifier：Write amplification意味着同一个记录需要被多次写入Disk，数字越大就意味着Disk write越多
2. Read Amplifier：Read amplification意味着查询一个记录，需要读多少次Disk，数字越大意味Disk Read越多
3. Space Amplifier：Space Amplifier等于所占空间大小/实际数据量大小，空间放大主要与未回收的过期数据量(old version/deleted entries)相关。数字越大意味着Disk overhead越多

Level Compaction上一层的SSTable可能需要全部读出来，然后和下一层某些overlapping的SSTable合并，会导致写的放大远远大于Size/Tier Compaction

Level Compaction相对每次更新compaction更频繁，保证每个level唯一Run，所以Read Amplifier比Tier的compaction有优势

Level Compaction对SSTable所做的维护更多，因此Space Amplifier比Tier的Compaction更小

面经

Keven Huang：

1. LSM-DB的实现
2. 数据库相关的东西
3. 如果希望时间戳是key的话要怎么设计比较好?
4. LSM还有哪些Compaction（合起来变成一个&时间戳Compaction...）

WindowsXp：

RisingWave一二面面经

比较好的参考资料

LSM Tree的Leveling 和 Tiering Compaction

LSM Tree 的compaction两难境地和balanceLSM卓越优秀的性能使其成为了目前流行的众多KV store...

知乎专栏

纯干货！深入探讨 LSM Compaction 机制

简介： compaction在以LSM-Tree为架构的系统中是非常关键的模块，log append的方式带来了高吞...

知乎专栏