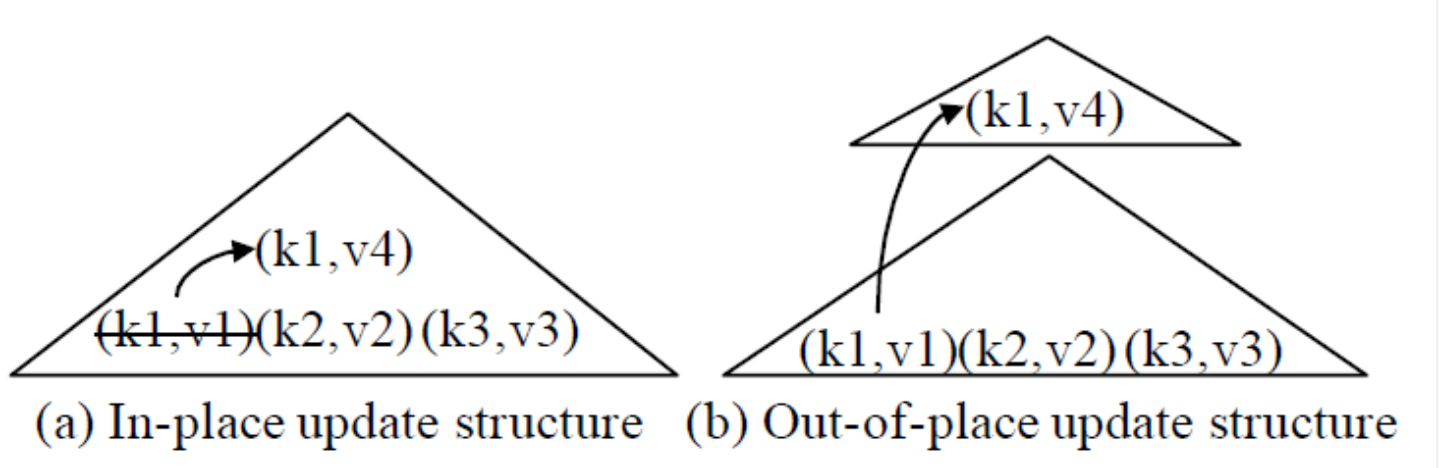
LSM Compaction Design

2021/12/10

1. LSM——what, why
2. LSM 的 compaction
3. compaction 策略对性能的影响
4. 工作负载对compaction策略的影响
5. 总结

LSM

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）

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。

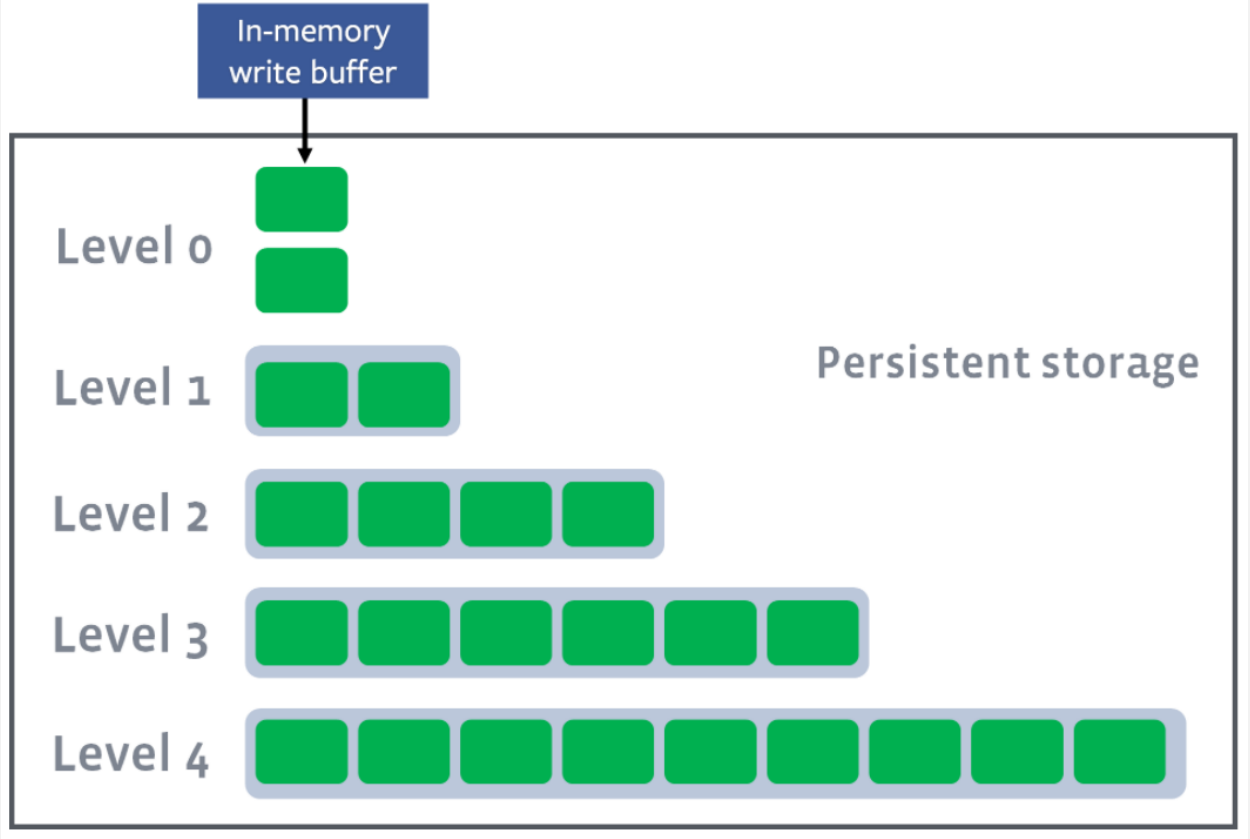
LSM

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、内存中缓存写入的数据（排序），满后刷到磁盘，称为**sorted runs**。（包含一个或多个文件）

2、磁盘上积累的runs，会被排序合并以构建更少但更长的sorted runs。这个过程被称为**Compaction**。

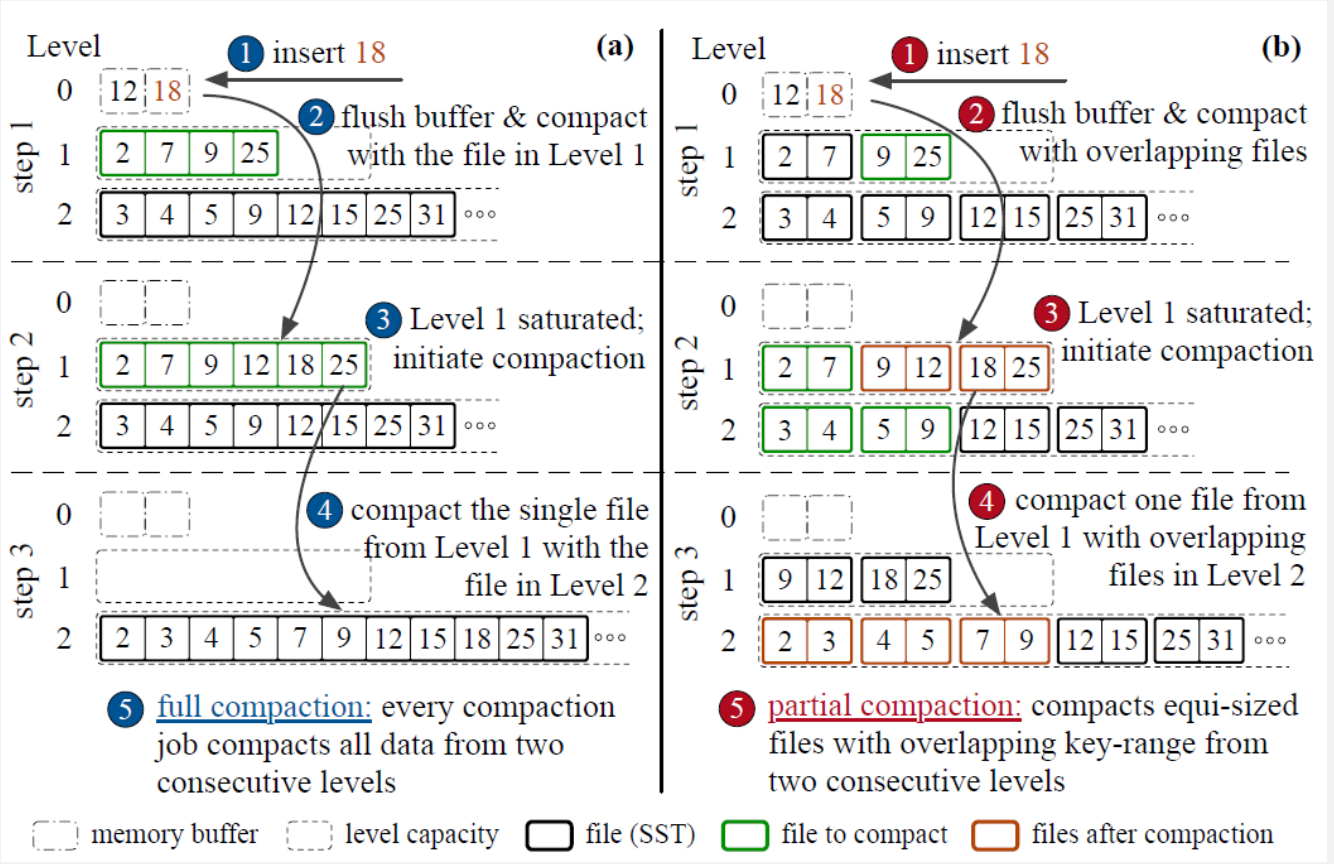
3、 **Compaction**通常是到一个新的level，磁盘上，每个level （>1）具有比上级大t倍的容量。



LSM

compaction需要数据在内存和磁盘之间来回移动，为了分摊IO代价，避免延迟峰值，**通常以文件的粒度而不是sorted run执行compaction**。

Partial compaction：
如果 Level_i 的数据增长超过阈值，将触发compaction，Level_i 中的一个文件（或文件子集）与选择的来自Level_i+1的文件（**具有重叠key范围**）进行压缩，



LSM

LSM删除

通过插入一种特殊类型的键值条目来实现，称为tombstone，即逻辑上使目标条目无效。

compaction期间才会真正删除有效条目。

tombstone条目到达最后一个level时才能清理，这时才是持久化的删除。

LSM查询

由于 LSM 树以异地方式实现更新和删除，因此树中可能存在多个具有相同键的条目，只有最近的版本才有效。

点查找：从内存缓冲区开始，从最小level到最大level，遍历LSM树。找到匹配的键后立即终止。为了提升效率，通常会使用如布隆过滤器和栅栏指针。

范围扫描： 范围扫描需要对符合范围查询条件的runs进行排序合并，跨越树的所有级别。runs在内存中进行排序合并，并返回每个符合条件的条目的最新版本，同时丢弃所有旧的、逻辑上无效的版本。

LSM


APACHE HBASE
WT
levelDB
SCYLLA
cassandra
tarantool
LSM-tree
RocksDB
Bigtable
DynamoDB
accumulo
SQLite
riak
QuasarDB
influxdb

1. LSM——what, why
2. LSM 的 compaction
3. compaction 策略对性能的影响
4. 工作负载对compaction策略的影响
5. 总结

Compaction

Compaction 主要由四部分组成：
Trigger(触发器)、Data Layout(数据布局)、Granularity(粒度)、Data Movement Policy(数据移动策略)

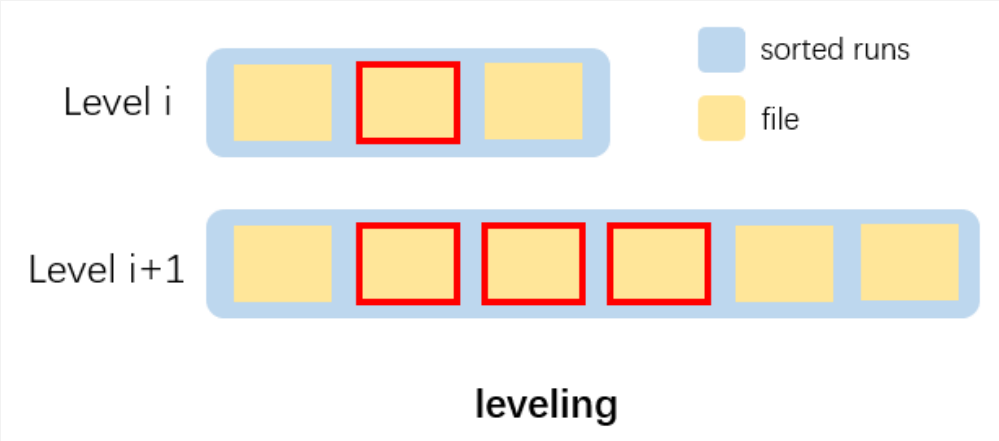
触发器
什么时候可以开始进行compaction。
常见的触发器基于LSM树中一个level的饱和度。即level i中的数据字节量与level i的理论容量的比例。一旦饱和度超过预定义的阈值，level i中的一个或多个文件会被标记为待压缩。

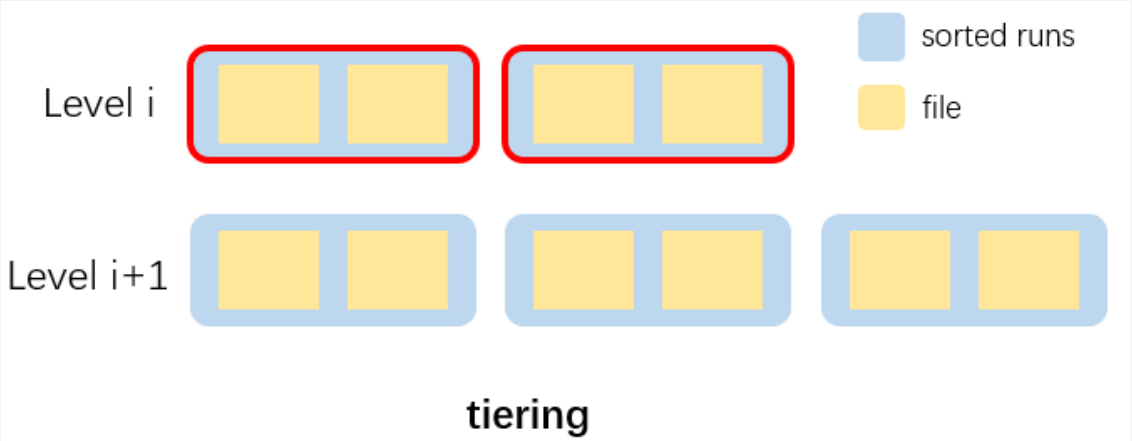
一些常见的**compaction**触发器：
1）level 饱和：level 中的数据大小达到阈值
2）sorted runs：一个 level 的 sorted runs 数量达到阈值
3）文件陈旧：文件在一个level待的时间过长
4）空间放大：整体空间放大超过阈值
5）删除条目_TTL：有删除条目的文件有额外的期望生存周期（更短）

Compaction

Compaction 主要由四部分组成:
Trigger(触发器)、Data Layout(数据布局)、Granularity(粒度)、Data Movement Policy(数据移动策略)

数据布局
控制每个level的sorted runs数量来确定磁盘上的数据组织结构。
通常分为两类: **leveling和tiering**。





一般来说，tiering提供更好的写放大，更差的读放大。

Compaction

Tiering和leveling可结合。

例如Dostoevsky中，**最后一层是leveling，其它层是tiering**。（SIGMOD 2018）
The Log-Structured Merge-Bush & the Wacky Continuum中，**每个level都可以决定选择leveling还是tiering**。（SIGMOD 2019）

一些常见的数据布局:
1）leveling：每 level 一个 sorted runs
2）tiering：每 level 多个 sorted runs
3）1-leveling: tiering for level 1, leveling otherwise
4）L-leveling: leveling for last level; tiering otherwise
5）Hybrid: 每层都可以独自选择是leveling还是tiering

Compaction

Compaction 主要由四部分组成:
Trigger(触发器)、Data Layout(数据布局)、Granularity(粒度)、Data Movement Policy(数据移动策略)

粒度
一次 compaction 作业期间移动的数据量。
例如直观的一种方法：将level i的所有数据排序合并到下一层，即 full compaction。

一些常见的粒度:
1）level: 两个连续level的所有数据
2）sorted runs: 一个level的所有sorted run
3）several sorted files: sorted runs 中的几个 file
4）sorted file: sorted runs 中的一个 file

Compaction

Compaction 主要由四部分组成:
Trigger(触发器)、Data Layout(数据布局)、Granularity(粒度)、Data Movement Policy(数据移动策略)

数据移动策略
当使用非Full compaction时，需要选择哪些文件（sorted runs）。
例如直观的一种方法：随机或使用循环策略（Level DB）。

一些常见的数据移动策略:
1）轮询：轮询方式选择文件，即循环策略
2）least overlapping parent：与下层重叠key范围最小的文件
3）least overlapping grandparent：类上
4）coldest: 最近最少被访问的文件
5）oldest: 该层中最老的文件，其实和轮询差不多
6）tombstone数量：文件删除标记数量达到阈值
7）tombstone时间：文件的删除周期达到阈值

Compaction

Compaction 主要由四部分组成:
触发器、数据布局、粒度、数据移动策略

其中触发器、粒度和数据移动策略是多值原语，而数据布局是单值。

一个compaction策略就由这四个部分的单个值或多个值组成。

Database	Data layout	Compaction Trigger					Compaction Granularity				Data Movement Policy							
		Level saturation	#Sorted runs	File staleness	Space amp.	Tombstone-TTL	Level	Sorted run	File (single)	File (multiple)	Round-robin	Least overlap (+1)	Least overlap (+2)	Coldest file	Oldest file	Tombstone density	Expired TS-TTL	N/A (entire level)
RocksDB [30], Monkey [22]	Leveling / 1-Leveling	✓		✓					✓	✓		✓		✓	✓	✓		
	Tiering		✓		✓	✓		✓										✓
LevelDB [32], Monkey (J.) [21]	Leveling	✓							✓		✓	✓	✓					
SlimDB [47]	Tiering	✓							✓	✓								✓
Dostoevsky [23]	*L*-leveling	$✓^L$	$✓^T$				$✓^L$	$✓^T$				$✓^L$						$✓^T$
LSM-Bush [24]	Hybrid leveling	$✓^L$	$✓^T$				$✓^L$	$✓^T$				$✓^L$						$✓^T$
Lethe [51]	Leveling	✓				✓			✓	✓		✓					✓	
Silk [11], Silk+ [12]	Leveling	✓							✓	✓	✓							
HyperLevelDB [35]	Leveling	✓							✓		✓	✓	✓					
PebblesDB [46]	Hybrid leveling	✓							✓	✓								✓
Cassandra [8]	Tiering		✓	✓		✓		✓										✓
	Leveling	✓				✓			✓	✓		✓				✓	✓	
WiredTiger [62]	Leveling	✓					✓											✓
X-Engine [34], Leaper [63]	Hybrid leveling	✓							✓	✓		✓				✓		
HBase [7]	Tiering		✓					✓										✓
AsterixDB [3]	Leveling	✓					✓											✓
	Tiering		✓					✓										✓
Tarantool [57]	*L*-leveling	$✓^L$	$✓^T$				$✓^L$	$✓^T$										✓
ScyllaDB [55]	Tiering		✓	✓		✓		✓										✓
	Leveling	✓				✓			✓	✓		✓				✓	✓	
bLSM [56], cLSM [31]	Leveling	✓							✓		✓							
Accumulo [6]	Tiering	✓	✓			✓		✓										✓
LSbM-tree [58, 59]	Leveling	✓					✓											✓
SifrDB [44]	Tiering	✓								✓								✓

Compaction

拿RocksDB的Universal-Compaction来详解一下。

它的本质只有一个level，所以先讲一个level的情况下是怎么compaction的，然后再扩展到多个level。

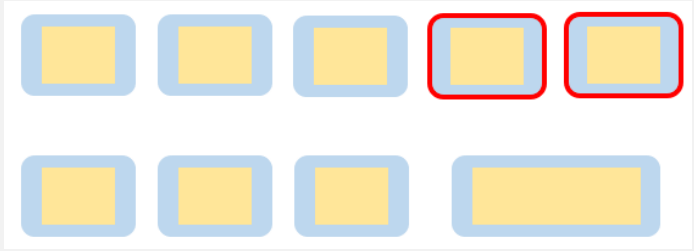
从零开始，内存中缓冲区不断的往磁盘刷，形成一个个file，也就是一个个sorted runs。
什么时候触发Compaction？ 直观的看，就是**sorted runs的数量**

compaction前提条件：sorted runs 的数量超过用户设定的阈值

Compaction

然后还有四个条件触发具体的Compaction，优先级由高至低如下：

1）**sorted runs的生存周期** —— sorted runs 太久没动过
2）**空间放大** —— size(R1 + … + Rn-1) / size(Rn)
3）**sorted runs 大小比例** —— size(R1 + … + Ri-1) / size(Ri)
4）**任意选择**



```
1
1 1   =>  2
1 2   =>  3
1 3   =>  4
1 4
1 1 4   =>  6
1 6
1 1 6   =>  8
```

比值0.25，sorted runs数量触发1

```
1 1 1 1 1  =>  5
1 5  (no compaction triggered)
1 1 5  (no compaction triggered)
1 1 1 5  (no compaction triggered)
1 1 1 1 5  => 4 5
1 4 5  (no compaction triggered)
1 1 4 5  (no compaction triggered)
1 1 1 4 5 => 3 4 5
1 3 4 5  (no compaction triggered)
1 1 3 4 5 => 2 3 4 5
```

比值1，sorted runs数量触发5

Compaction

RocksDB的Universal-Compaction —— 多level

较大的sorted runs尽可能的被放入更高的level，然后被分成多个不超过阈值的文件。

一个可能的布局：6个level，5个sorted runs。

```
Level 0: File0_0, File0_1, File0_2
Level 1: (empty)
Level 2: (empty)
Level 3: (empty)
Level 4: File4_0, File4_1, File4_2, File4_3
Level 5: File5_0, File5_1, File5_2, File5_3, File5_4, File5_5, File5_6, File5_7
```

level 0和原来一样，维护多个包含单一file的sorted run，然后level 1~level n 就是n个sorted runs，只是可能会分成多个文件

Compaction

```
Level 0: File0_0, File0_1, File0_2
Level 1: (empty)
Level 2: (empty)
Level 3: (empty)
Level 4: File4_0, File4_1, File4_2, File4_3
Level 5: File5_0, File5_1, File5_2, File5_3, File5_4, File5_5, File5_6, File5_7
```

条件1触发，选择File0_1, File0_2, level 4，即sorted runs 2/3/4

```
Level 0: File0_0
Level 1: (empty)
Level 2: (empty)
Level 3: (empty)
Level 4: File4_0', File4_1', File4_2', File4_3'
Level 5: File5_0, File5_1, File5_2, File5_3, File5_4, File5_5, File5_6, File5_7
```

1）sorted runs的生存周期 —— sorted runs 太久没动过
2）空间放大 —— size(R1 + … + Rn-1) / size(Rn)
3）sorted runs 大小比例 —— size(R1 + … + Ri-1) / size(Ri)
4）任意选择

Compaction

```
Level 0: File0_0, File0_1, File0_2
Level 1: (empty)
Level 2: (empty)
Level 3: (empty)
Level 4: File4_0, File4_1, File4_2, File4_3
Level 5: File5_0, File5_1, File5_2, File5_3, File5_4, File5_5, File5_6, File5_7
```

条件2触发，即所有sorted runs 参与compaction

```
Level 0: (empty)
Level 1: (empty)
Level 2: (empty)
Level 3: (empty)
Level 4: (empty)
Level 5: File5_0', File5_1', File5_2', File5_3', File5_4', File5_5', File5_6', File5_7'
```

1）**sorted runs的生存周期** —— sorted runs 太久没动过
2）**空间放大** —— size(R1 + … + Rn-1) / size(Rn)
3）**sorted runs 大小比例** —— size(R1 + … + Ri-1) / size(Ri)
4）**任意选择**

Compaction

```
Level 0: File0_0, File0_1, File0_2
Level 1: (empty)
Level 2: (empty)
Level 3: (empty)
Level 4: File4_0, File4_1, File4_2, File4_3
Level 5: File5_0, File5_1, File5_2, File5_3, File5_4, File5_5, File5_6, File5_7
```

条件3或4触发，选择了File0_0, File0_1, File0_2, 即sorted run 1/2/3

```
Level 0: (empty)
Level 1: (empty)
Level 2: (empty)
Level 3: File3_0, File3_1
Level 4: File4_0, File4_1, File4_2, File4_3
Level 5: File5_0, File5_1, File5_2, File5_3, File5_4, File5_5, File5_6, File5_7
```

1）**sorted runs的生存周期** —— sorted runs 太久没动过
2）**空间放大** —— size(R1 + … + Rn-1) / size(Rn)
3）**sorted runs 大小比例** —— size(R1 + … + Ri-1) / size(Ri)
4）**任意选择**

1. LSM——what, why
2. LSM 的 compaction
3. compaction 策略对性能的影响
4. 工作负载对compaction策略的影响
5. 总结

性能影响

Primitives	Full [3, 58, 62]	LO+1 [22, 30, 51]	Cold [30]	Old [30]	TSD [30, 34]	RR [31, 32, 35, 56]	LO+2 [32, 35]	TSA [51]	Tier [8, 33, 47]	1-Lvl [30, 39, 48]
Trigger	level saturation	level sat.	level sat.	level sat.	1. TS-density 2. level sat.	level sat.	level sat.	1. TS age 2. level sat.	1. #sorted runs 2. space amp.	1. #sorted runs^T 2. level sat.^{L}
Data layout	leveling	leveling	leveling	leveling	leveling	leveling	leveling	leveling	tiering	hybrid
Granularity	levels	files	files	files	files	files	files	files	sorted runs	1. sorted runs^T 2. files^L
Data movement policy	N/A	least overlap. parent	coldest file	oldest file	1. most tombstones 2. least overlap. parent	round-robin	least overlap. grandparent	1. expired TS-TTL 2. least overlap. parent	N/A	1. N/A^T 2. least overlap. parent^L

Table 2: Compaction strategies evaluated in this work. [^{L}: levels with leveling; ^{T}: levels with tiering.]

1) Full: level状态触发，leveling布局， 两个整level压缩。

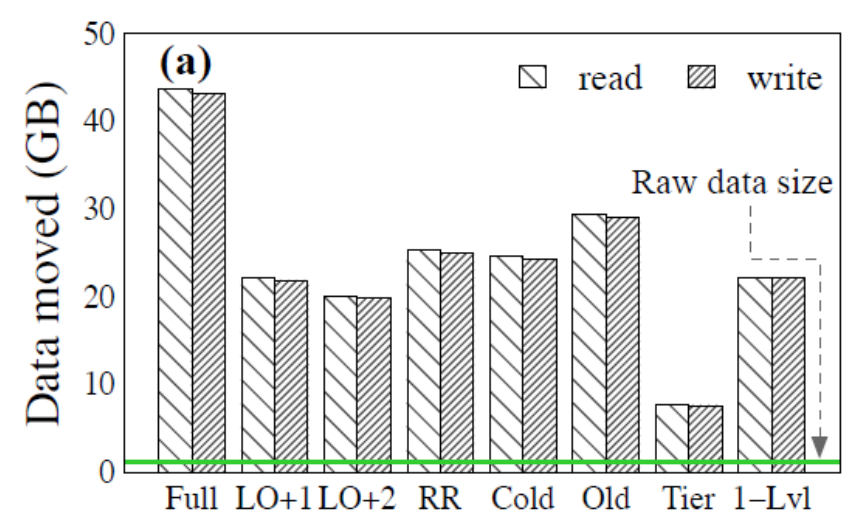
2) L(O+1)：level状态触发，leveling 布局，选择与下层最少文件重叠的文件（sorted runs）。
3) L(O+2): 类上，选择与i+2层最小重叠的文件
4) Cold: 类上，选择最少访问的文件
5) Old: 类上，选择最老的文件
6) RR: 类上，循环选择

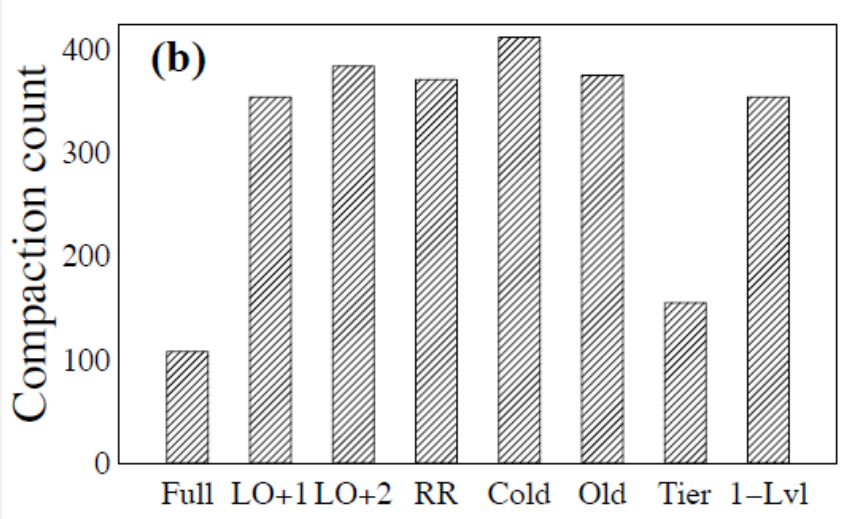
7) TSD: 删除标记数量和level状态触发，leveling布局，选择2) 或删除标记多的。
8) TSA: 删除标记生存周期和level状态触发，leveling布局，选择2) 或生存时间到了的。

9) Tier: sorted run的数量和空间放大触发，tiering布局，选择sorted runs。RocksDB的universal
10) 1-Lvl: sorted runs数量和level状态触发，根据2) 选择文件。RocksDB的leveled

性能影响

1.数据写：此实验中将生成的10M个（数据量估计在1G左右，即平均100B）键值条目插入到一个空数据库中，衡量原始写入性能和compaction性能。



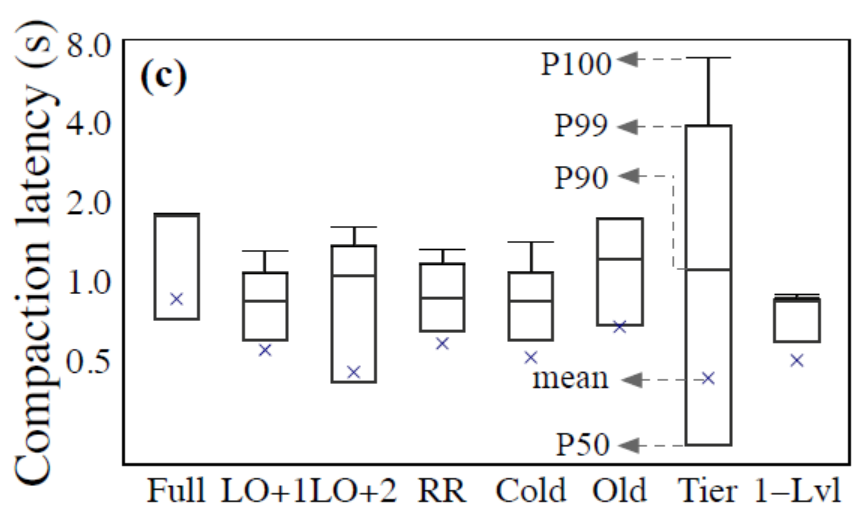


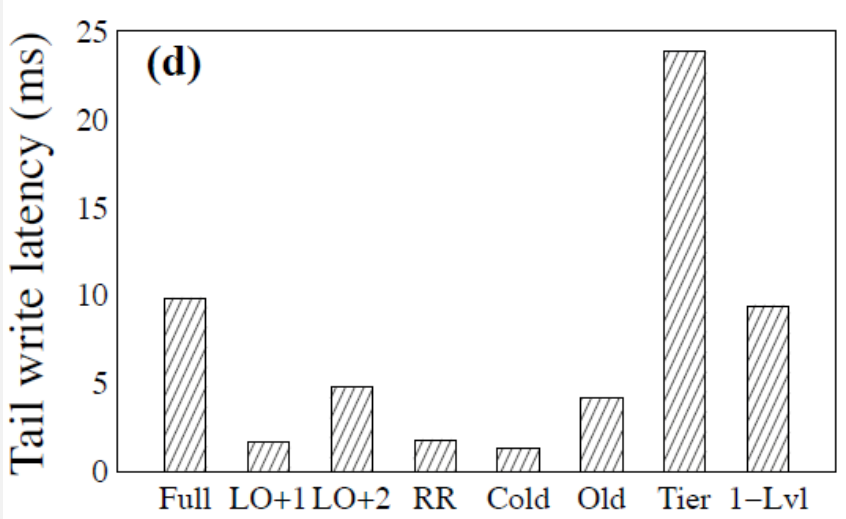
O1: compaction将导致大量数据移动。

O2: partial compaction 减少压缩的数据移动量，增加压缩次数

性能影响

1.数据写：此实验中将生成的10M个（数据量估计在1G左右，即平均100B）键值条目插入到一个空数据库中，衡量原始写入性能和compaction性能。



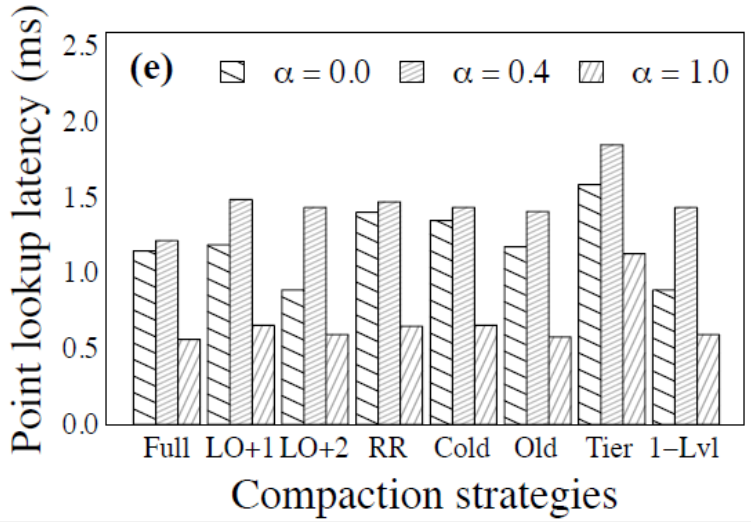


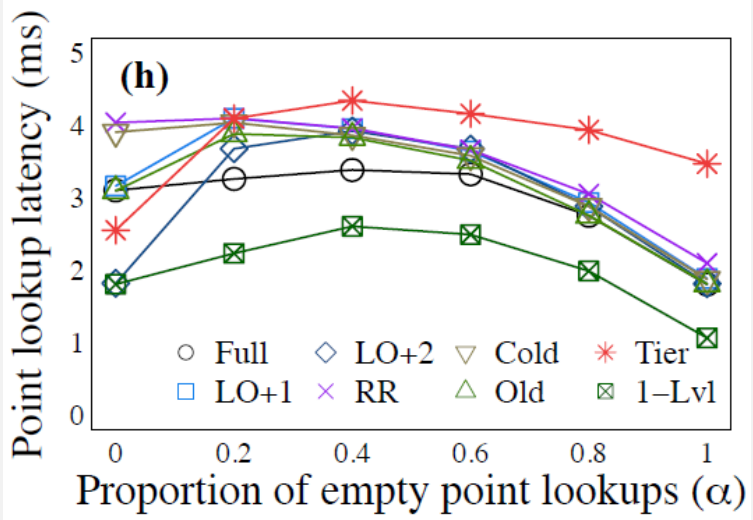
O3: Full compaction平均compaction延迟最高，Tier最低但不稳，1-Lvl最稳

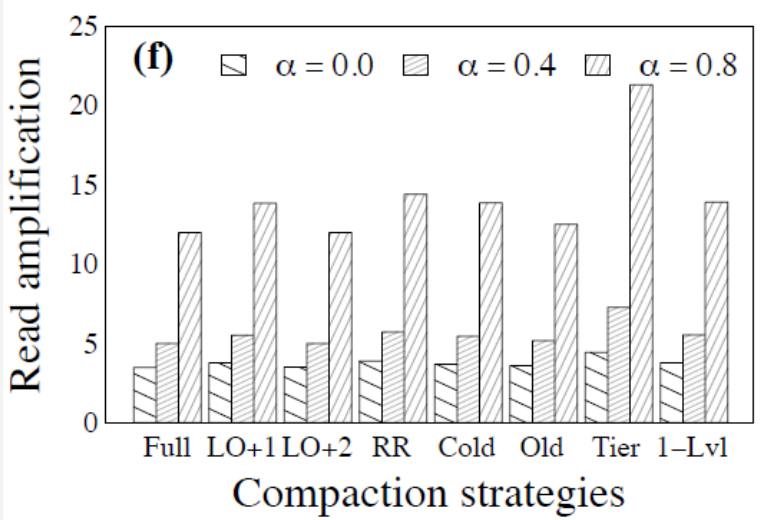
O4: Tier可能会造成长时间的写停顿，tier的尾部写延迟可到25ms，而leveling部分压缩如cold最低可到1.3ms。

性能影响

2.数据读（点查）：根据1中写进去的10M条数据，执行1M个点查找（均匀分布），并用@的取值表示空查的比例（0表示全非空，1表示全空）。







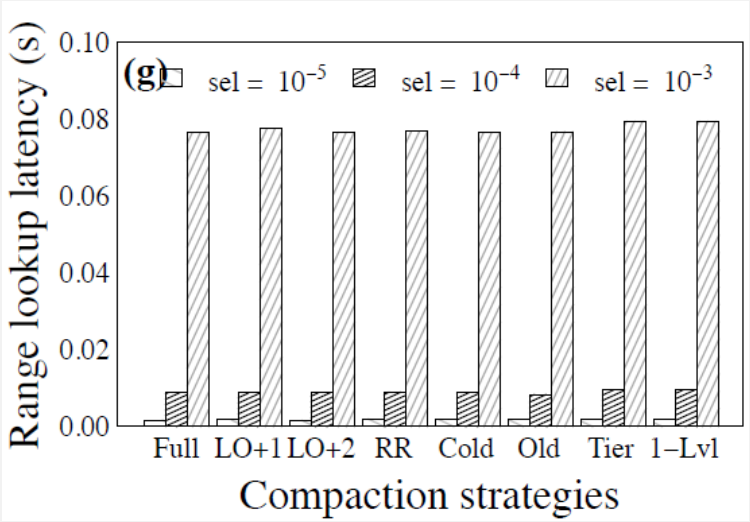
O5: Tier点查延迟最高，Full 最低。(其实整体都没有太大的区别，因为布隆过滤器和缓存。

O6: 空查和非空查比例持平时，平均查找延迟最高。（缓存命中问题）

O7: Tier的读放大最严重。（特别是空查较多的情况下）

性能影响

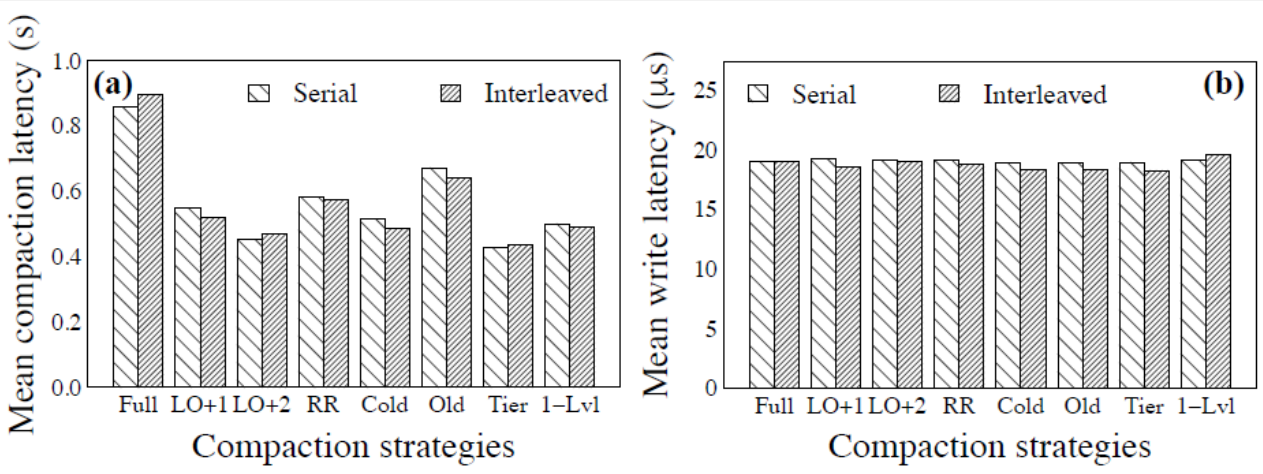
3.数据读（范围）：根据1中写进去的10M条数据，执行1000个范围查询，同时使用不同的选择率。选择率就是符合条件的结果数所占的比例。



O8: compaction 策略对范围查询的影响微乎其微。

性能影响

4.混合负载：把10M个写和1M个读交织在一起。空查比例因实验而异。



O9:平均compaction延迟，写延迟，写放大和单写差不多。但尾部写延迟会大很多（2-15倍）

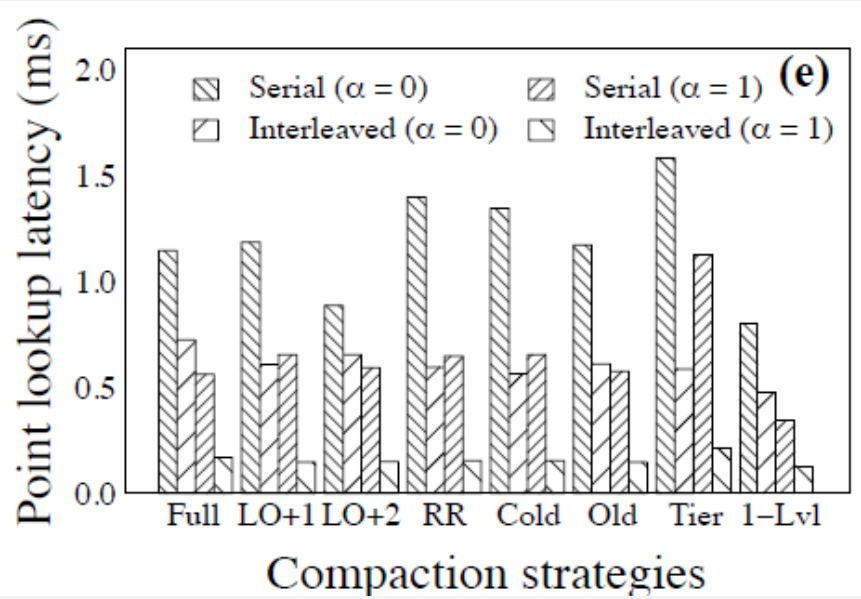
两个原因——

1) 点查和写操作会竞争IO。

2) 在内存缓冲区的查找会延迟刷盘。进一步延迟写操作。

性能影响

4.混合负载：把10M个写和1M个读交织在一起。空查比例因实验而异。

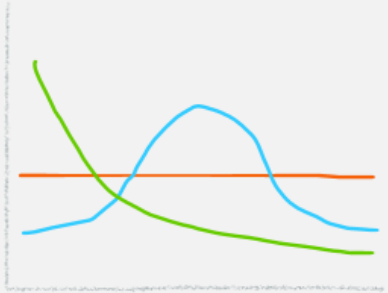


O10: 混合负载有利于缓存命中。

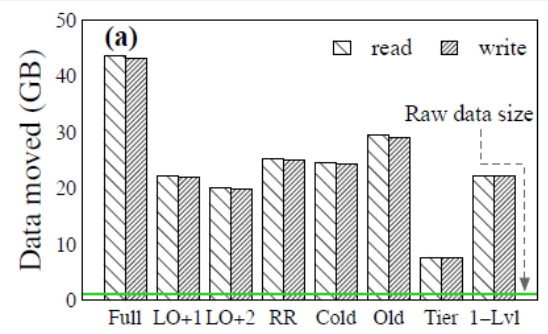
还是两个原因——
1）混合负载，查询就在内存的几率大一点。
2）compaction 期间文件元数据更新缓存也会有利与查询（过滤器，索引）。

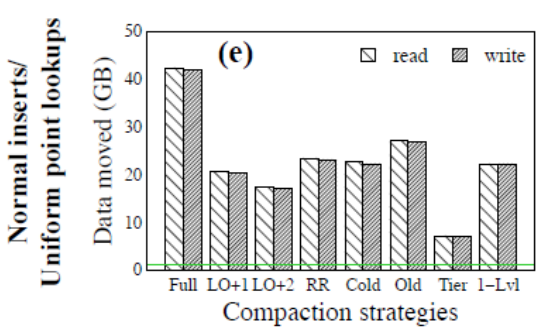
1. LSM——what, why
2. LSM 的 compaction
3. compaction 策略对性能的影响
4. 工作负载对compaction策略的影响
5. 总结

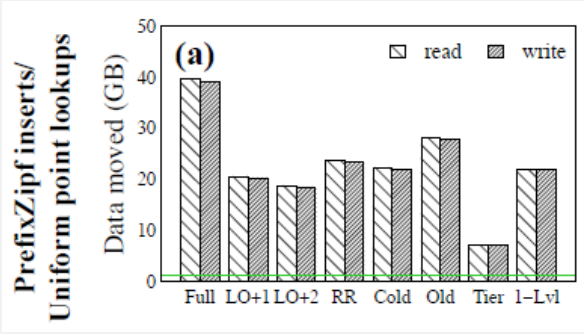
工作负载影响

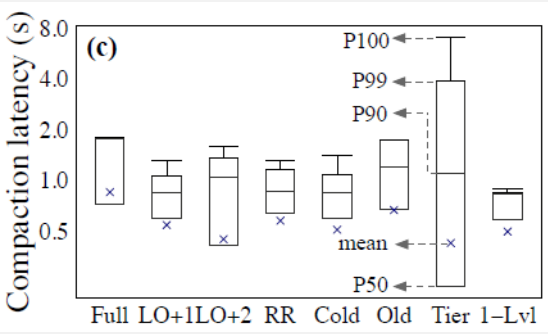


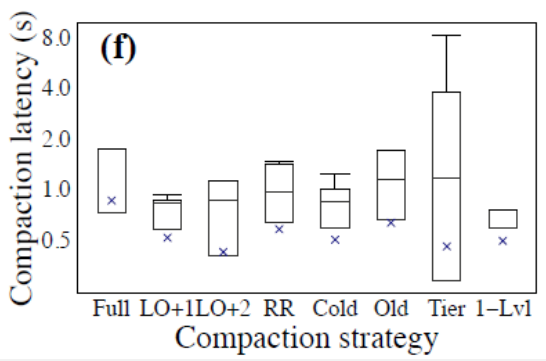
5.不同的写入分布：即写入key的分布情况。
三种——均匀分布，正态分布，Zipfian分布（幂律分布）。

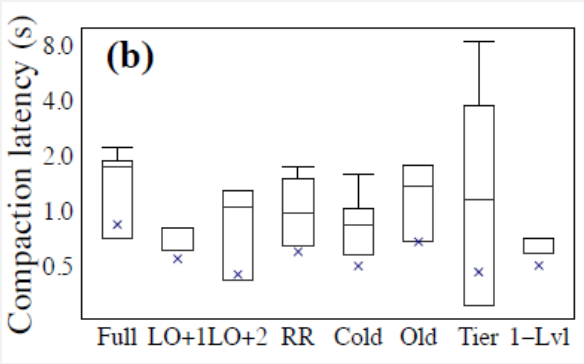






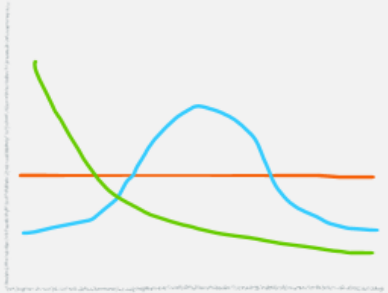




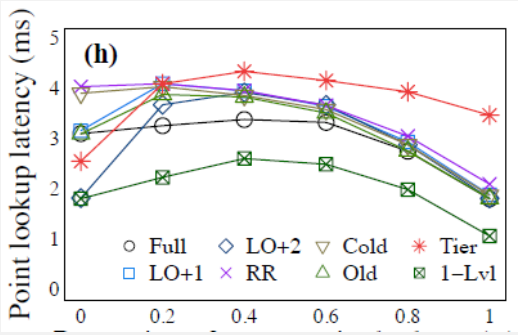


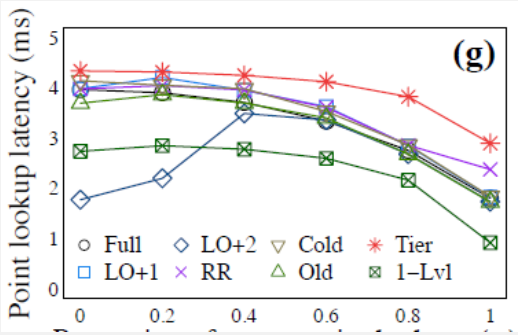
O11: 对于仅写入的工作负载，数据分布不会影响compaction策略的选择。
只要数据分布不随时间变化，那么每一层都遵循相同的分布，不同level之间的重叠保持不变。

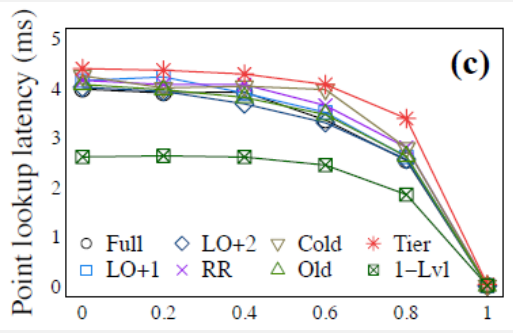
工作负载影响

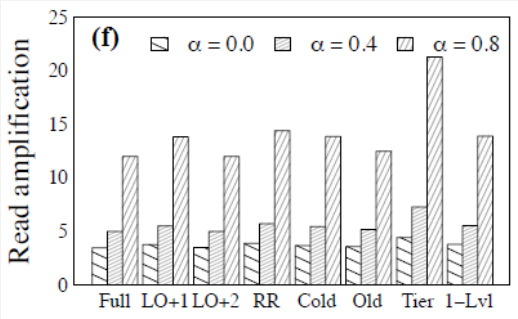


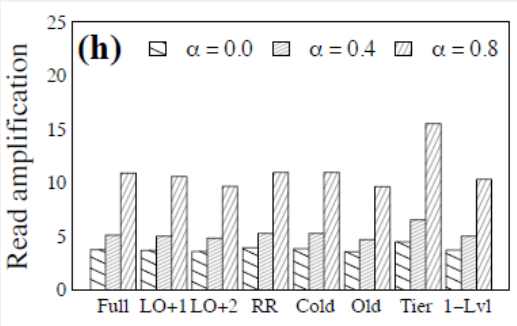
5.不同的写入分布：即写入key的分布情况。
三种——均匀分布，正态分布，Zipfian分布（幂律分布）。

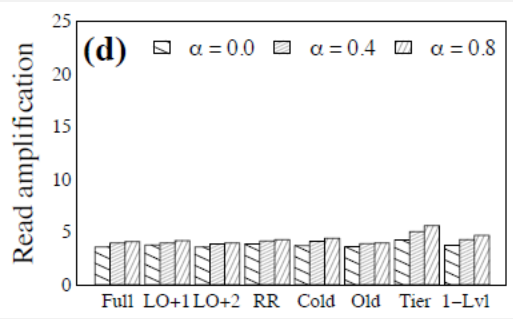






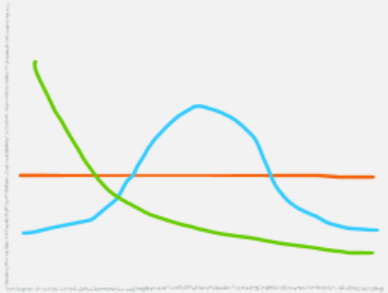




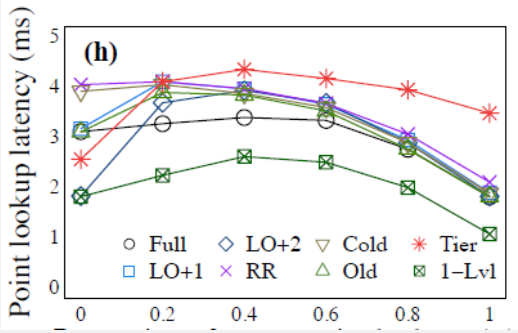


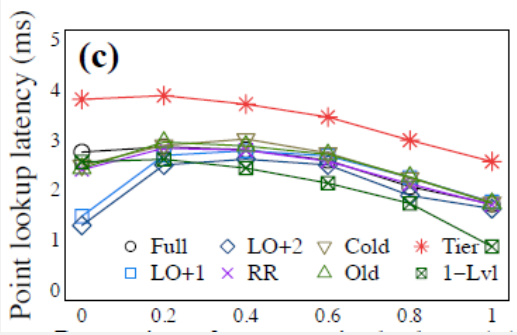
O12: 空查比例比较高的情况，点查性能略有提升；另外读放大会好一点。特别是幂律分布。
毕竟查询是均匀分布的查，对于有偏序分布的数据， 不存在的点查更容易跳过，越偏越厉害。

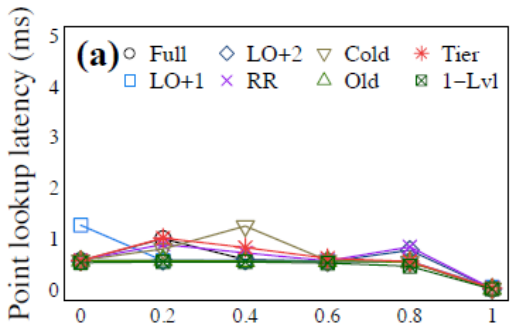
工作负载影响

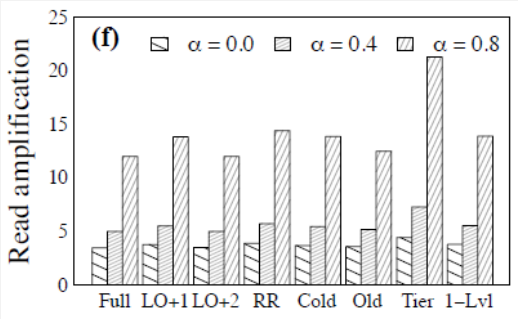


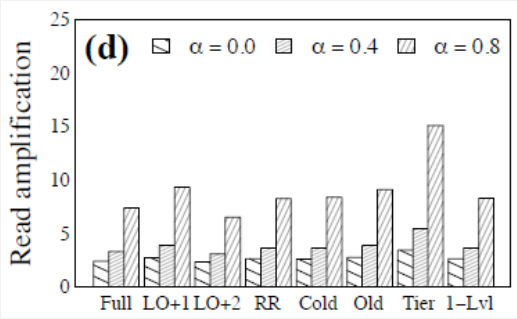
6.不同的查询分布：保持原始写入均匀分布，换成查询是三种分布。

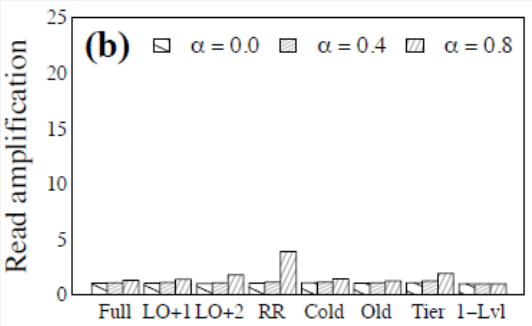








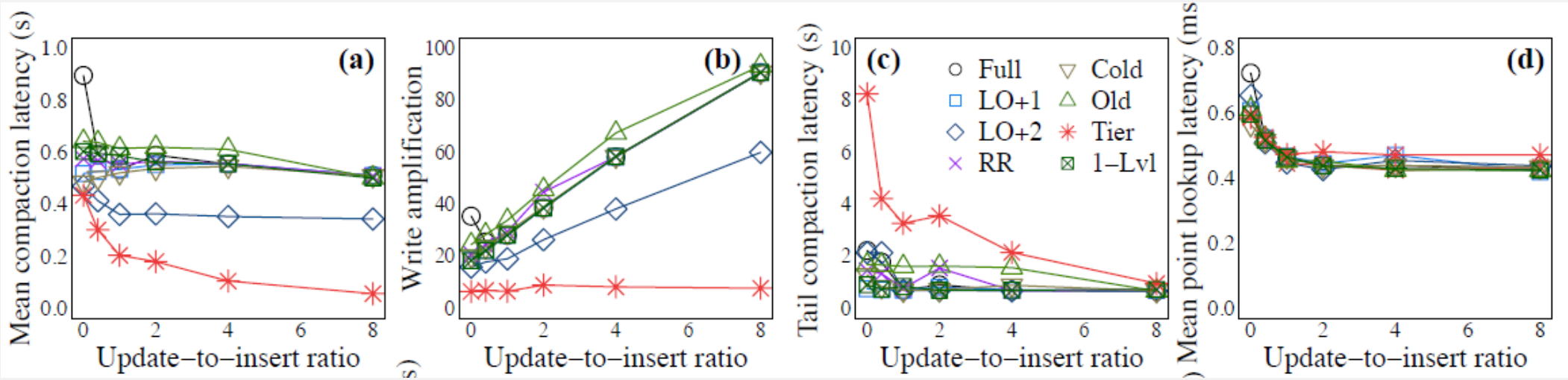




O13: 偏序查询下，查询性能和放大会好很多。
因为大部分查询比较集中，缓存的命中率会高得多。

工作负载影响

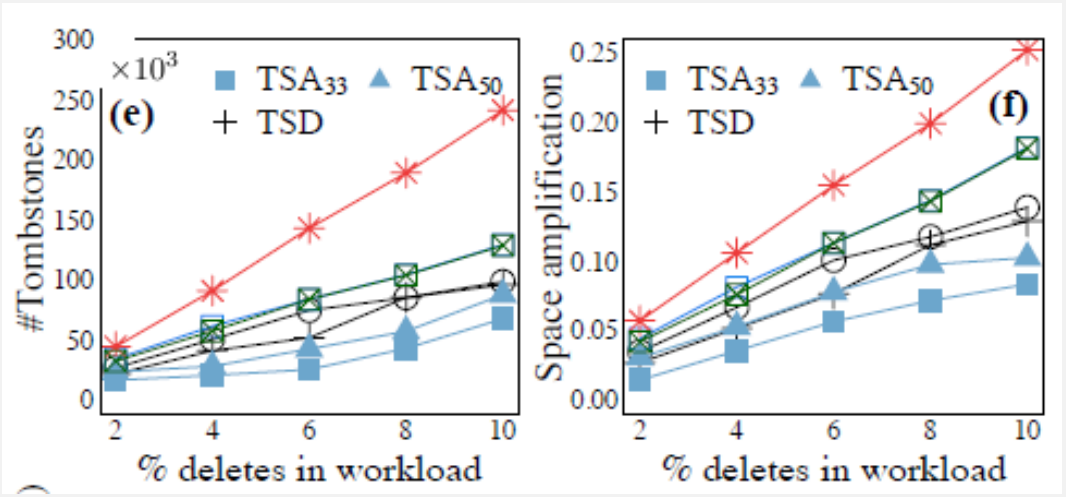
6.不同的更新比例：同时写入和查询交替。例如更新比例是0，那么意味着每次写入都是唯一的key。如果是10，意味着每个key平均有10次更新。

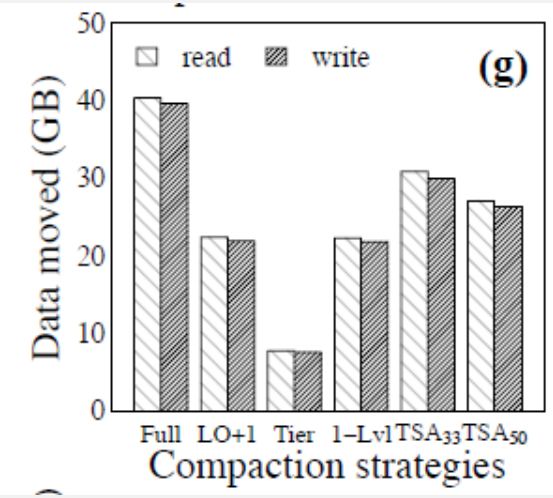


O14:对于更新密集的工作负载，Tier主宰了性能。包括compaction性能和写，读性能也不太差。

工作负载影响

7.不同的删除比例：从已有key发出的删除，与写入交替。

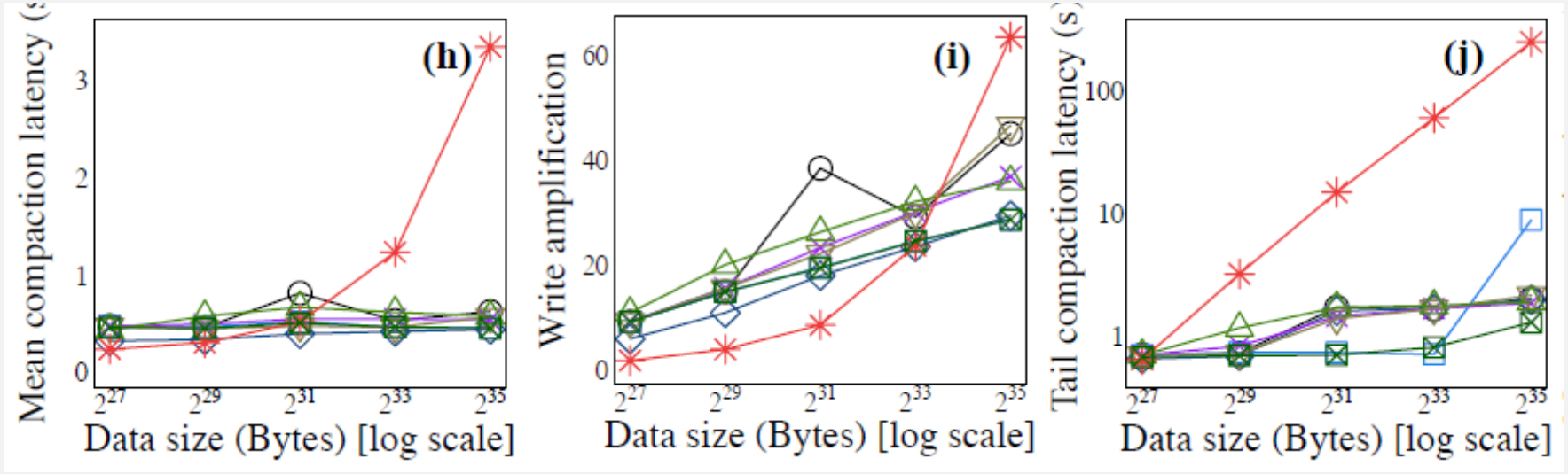




O15: TSA和TSD专为删除而设计，所以实验运行结束时含删除标记的肯定最少，相应的空间放大最低。
相应地，优化删除会导致更多的compaction，所以写放大会更严重。
所以这个适用场景是需要及时删除，或者删除较多需要减少删除引起的空间放大。

工作负载影响

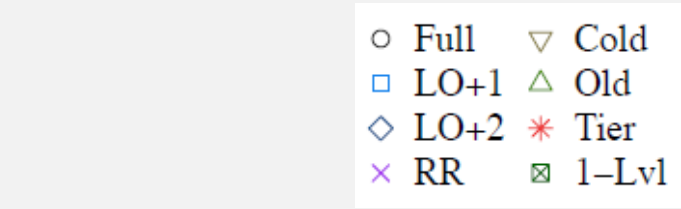
8.增加写入量：现在写更多的数据，主要是看扩展性。

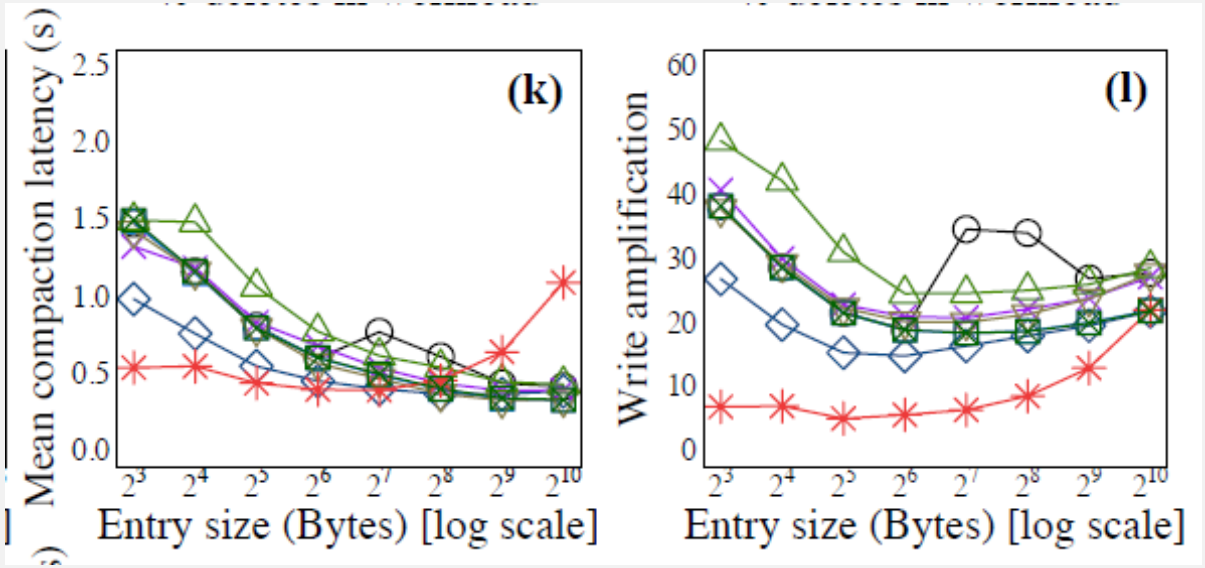


O16: 随着数据大小的增大，Tier的尾部compaction延迟持续增大，这使得Tier不适用于对延迟敏感的应用程序。另外写放大优势也不复存在。
另外，当数据大小达到2GB时，Full跳了一下应该是触发了级联compaction ，将所有数据写入一个新的level，导致写放大和compaction延迟峰值。

工作负载影响

9.不同的条目大小。即kv对的value大小





O17: 条目较小时，压缩代价更高。
这是因为较小的条目会增加每个磁盘页条目的数量，导致1）压缩期间比较更多的键，即消耗更多CPU，2）更大的过滤器。
条目较大又开始上升是因为单纯数据总量多了，compaction次数会多很多。Tiering上升的更快也是因为tiering压缩往往涉及更多的sorted run

1. LSM——what, why
2. LSM 的 compaction
3. compaction 策略对性能的影响
4. 工作负载对compaction策略的影响
5. 总结

总结

关于LSM compaction
的比较完整的指南

**LSM 的
Compaction 设计**

- **LSM what , why** — Out-of-place的结构；更快的写
- **LSM 的compaction** — 定义——四个原语：触发器，粒度，数据布局，数据移动策略
  RocksDB的universal-compaction 详解
- **不同compaction策略对性能的影响** — 十种具有代表性的compaction 策略；
  性能指标包括compaction延迟，读写延迟，读写放大，空间放大
- **工作负载对compaction的影响** — 不同的写入分布；查询分布；更新比例；删除比例
  总数据量大小；单个条目大小

没有完美的压缩策略（RocksDB的两种策略可以涵盖大部分场景）。
正确的压缩策略可以显著提高存储引擎的读写性能。

谢 谢