StellarDB 底层存储优化

当前阶段，针对 StellarDB 底层存储性能，我们选取 Compaction 作为突破点，进行存储的性能优化。到目前为止，通对 compaction 策略的调整，可以实现对整个存储流程的性能提升，具体表现如下：

- 数据写入 mem-table 的性能提升 20%
- mem-table 刷新到磁盘的性能提升 3 倍
- compaction 性能提升 5 倍

以上结果均在测试环境下得出，在真实环境下的效果需要加以验证。

compaction 影响因素

通过阅读 compaction 相关资料，以及 StellarDB 中 compaction 的实现。我们从提升单次 compaction 性能表现和降低 compaction 次数两个角度对 compaction 进行调整，确定出以下几个可能的影响因素：

- 每一层是否开启压缩算法
- 每一层单次 compaction 选取文件数
- 每一层 compaction 的线程数
- 每一层 compaction 的触发文件数
- 层内并行 Compaction 的可行性
- 每一层文件的大小
- 底层存储的层数
- 定期的 compaction
- Compaction 任务被抛弃

Compression

在每一层是否开启压缩算法对于 compaction 单次的性能以及 compaction 的次数均存在影响。

- 关闭 $L_k(k=0)$的压缩，主要影响 mem-table 刷新到磁盘的性能，关闭后

mem-table 刷新的速度可以提升 5 倍，同时受到 mem-table 刷新速度的影响，mem-table 的写入速度也提升了 20%。

- 关闭 L_k(0<k<n)的压缩，k-1 层 compaction 的总时间大幅度下降，代价是在第 k 层产生的文件数大量增加，导致第 k 层 compaction 次数也急剧增多，compaction 总时间大幅度上升。
- 优化方案：L0 层 compression 必须关掉，Lk(0<k<n)层可以通过使用轻量级压缩算法，从而在提升 k-1 层 compaction 性能表现的同时使得第 k 层 compaction 的性能下降不那么严重。

Compaction pick files

Compaction pick files 是指每一层在进行 compaction 时所选取的文件数量。其作用在于使得单次 compaction 能合并更多的文件。

在当前的场景下，批量导入一大批数据，所有的生成的与 mem-table 等同大小的小文件都会堆积在第 0 层，L_k(k>0)的 compacion 都处于饥饿的状态，没有数据可处理。

通过增大每一层的 compaction pick files， 可以使得数据在层与层之间的流动变快，达到层与层之间近似并行 compaction 的状态；同时也可以减少每一层的 compaction 总次数，有助于降低该层 compaction 的总时间。以上作用理论上可以通过增大该层文件的大小达到同样的效果，但尚未经过测试。

Compaction thread number

Compaction thread number 是指每一层用于执行 compaction 任务的线程数，在当前状态下，经测试验证，调整 compaction thread number 对于 compaction 的总时间没有影响。原因是 L_k(k=0,1)层单次执行单词 compaction 所要合并的总文件大小有限，在 L_k 层第二个 compaction 触发时，第一个 compaction 可能已经结束或者接近尾声，从头到尾需要的线程数只有少许几个，故而增大线程数无用，无法实现真正的层内并行 compaction。

Compaction trigger files

Compaction trigger files 是指每一层触发 compaction 的文件数。其次，我对 compaction 的理解是当每一层的数据量达到该层的上限时，进行 compaction。由于当前我们每一层文件大小可以设置，一旦文件大小设定，这时候每一层能够容纳的数据量就受到 compaction trigger files 的影响。

而 compaction trigger files 和 compaction pick files 以及 compaction thread number, 共同影响着每一次 compaction 的并行度。比如我们在 L_k 层设置单个文件大小为 128m, 10 个文件时触发 compaction, 那么该层能够容纳的数据量为 1280m, 假定每次 compaction pick files 为 5, 那么最多只有 2 个线程同时 compaction, 并行度不足；假定每次 compaction pick files 为 2, 那么最多只有 5 个线程同时 compaction，该层的 compaction 并行度足够，但单个 compaction 所要合并的数据量减少，导致 compaction thread number 中提到的问题。

上面提到的问题理论上可以通过增加每一层的文件大小改善，但未经过测试验证。

层内并行 Compaction 的可行性

当前的 compaction 做法是在每一层选取指定的文件数进行 compaction，并没有考虑该层内所堆积的数据量。

我的一个想法是，当某一层 compaction 被触发时，选中该层内所有需要 compaction 的文件，分割成多个 sub compaction task，然后并行执行。

Level file size

Level file size 是指每一层的文件大小，针对其作用在前面的部分有所猜测，但未经过实验验证。

底层存储的层数

由于存储要承载的数据量可能从几个 G 到几个 T, 在承载的数据量不同的情况

下，固定底层存储的层数是否合理，待验证。

定期的 compaction

我对当前定期 compaction 的方式存疑，一方面是定期 compaction 的触发时间，应当是文件的存活时间，而不是定期系统触发 compaction 任务；另一方面是太短的定期compaction的触发时间太短，并不会出现我们所期望的L_0~L_{n-1}存在文件，所有的数据都被在合并到了最后一层。

Compaction 任务被抛弃

当前的 compaction 实现，当没有可用的 compaction 线程时，触发的 compaction 任务会被抛弃。

待解决的问题

1. 明确以上尚未明确其影响的因素的准却影响
2. 提供可选的轻量级压缩算法
3. 层内进行并行 compaction 的可行性
4. 单独列出 compaction 日志，以获得对当前 compaction 状态的反馈