LSM VS B+Tree

B-tree实现比LSM-tree的实现更为成熟，然而由于LSM-tree的特点，LSM-tree目前更有吸引力。根据经验，LSM-tree通常对于写入更快，而B-tree被认为读取更快。

LSM-tree优点：

B-tree索引必须至少写入两次数据：一次写入预写日志，一次写入树的页本身（还可能发生页分裂）。即使该页只有几个字节的修改，也必须承受写整个页的开销。一些存储引擎甚至覆盖相同的页两次，以避免在电源故障的情况下最终出现部分更新的页。

由于反复压缩和SSTtable的合并，日志结构也会重新数据多次，这种影响称为写放大（在数据库内，由于一次数据库写入请求导致的多次磁盘写）。对于SSD，由于只能承受有限次的擦出覆盖，因此尤为关注写放大指标。

对于大量写密集的应用程序，性能瓶颈可能在于数据库写入磁盘的效率。在这种情况下，写放大具有直接的性能成本：存储引擎下写入次磁盘的次数越多，可用磁盘带宽中每秒可用处理的写入越少。

LSM-tree可用比B-tree更高的写入吞吐量，部分是因为它们有时具有较低的写放大，部分原因是因为它们以顺序方式写入紧凑的SSTable文件，而不必重写树中的多个页。

LSM-tree可用支持更好的压缩，因此通常磁盘上的文件比B-tree小很多。由于碎片，B-tree存储引擎使某些磁盘空间无法使用：当页被分裂或当一行的内容不能适合现有页时，页的某些空间无法使用。由于LSM-tree不是面向页的，并且定期重写SSTable以消除碎片化，所以它们具有较低的存储开销，特别是在使用分层压缩时。

LSM-tree的缺点：

日志结构存储的缺点是压缩过程有时会干扰正在进行的读写操作。即使存储引擎尝试增量的执行压缩，并且不影响并发访问，但由于磁盘的并发资源有限，所以当磁盘执行昂贵的压缩操作时，很容易发生读写请求等待的情况。这对吞吐量和平均响应时间的影响通常很小，但是观察较高的百分位，日志结构化存储引擎的查询响应时间有时会相当高，而B-tree的响应延迟则更具确定性。

高写入吞吐量时，压缩的另一个问题就会冒出来：磁盘的有限写入带宽需要在初始写入（记录并刷新内存表到磁盘）和后台运行的压缩线程之间所共享。写入空数据库时，全部的磁盘带宽可用于初始写入，但数据库的数据量越大，压缩所需的磁盘带宽就越多。

如果写入吞吐量很高并且压缩没有仔细配置，那么就会发生压缩无法匹配新数据写入速率的情况。在这种情况下，磁盘上未合并段的数量不断增加，直到磁盘空间不足，由于它们需要检查更多的段文

件，因此读取速度也会降低。通常，即使压缩不能跟上，基于SSTable的存储引擎也不会限制到来的写入速率，因此需要额外的监控存储来发现这种情况。

B-tree的优点则是每个键都恰好唯一对应索引中的某个位置，而日志结构的存储引擎可能在不同的段具有相同的键的多个副本。如果数据库希望提供强大的事务语义，这方面B-tree显得更具有吸引力：在许多关系性数据库中，事务隔离是通过键范围上的锁实现的，并且在B-tree索引中，这些锁可用直接定义在树中。