按大小分层压实的一个问题叫作表饥饿：如果压实后的表仍然足够小（例如，数据记录被墓碑遮蔽，没有进入合并的表），则更高的层可能产生压实饥饿，其墓碑将一直不被回收，从而增加读取的开销。在这种情况下，对于一个层将不得不进行强制压实，即使它没有包含足够数量的表。

还有其他一些常见的压实策略实现，可以针对不同的工作负载对其进行优化。例如，Apache Cassandra 还实现了一个时间窗口压实策略，该策略对于具有生存期（time-to-live）的时间序列型工作负载（换句话说，数据项必须在给定的一段时间之后过期）特别有用。

时间窗口压实策略将写入时间戳考虑在内，允许整个丢弃那些持有已过期时间范围的数据的文件，而不需要对它们的内容进行压实和重写。

7.2 读写放大与空间放大

在实现最优压实策略时，我们必须考虑多个因素。一种方法是回收重复记录占用的空间，减少空间开销，但这会产生由不断重写表导致的更高的写放大。替代方案是避免连续重写数据，而这又增加了读放大（在读取期间协调关联到相同键的数据记录的开销）和空间放大（因为冗余记录会被保存更长时间）。

数据库界的一大争议是 B 树和 LSM 树之中到底哪个的写放大较低。对于二者，理解写放大的来源是极其重要的。在 B 树中，写放大来自回写操作以及后续对同一节点的更新。而在 LSM 树中，写放大是由在压实过程中将数据从一个文件迁移到另一个文件引起的。直接比较两者可能会导致不正确的结论。

总结起来，当以不可变的方式在磁盘上存储数据时，我们面临三个问题：

读放大
: 由为了检索数据而需要读取多个表所引起。

写放大
: 由压实过程中不断进行的重写所引起。

空间放大
: 由存储关联到同一键的多个记录所引起。

在本章的剩余部分，我们将逐一对这些问题进行讨论。