简介

X-Engine 主要解决电商事务中的三个问题:

- The tsunami problem: 事务峰值高 -> 提高系统整体吞吐;
- The flood discharge problem: 内存和磁盘大量热数据交换 -> 提高不同 Level 间数据交换效率;
- The fast-moving current problem: 数据热度变化快 -> 快速读取冷数据;

整体架构图如下:

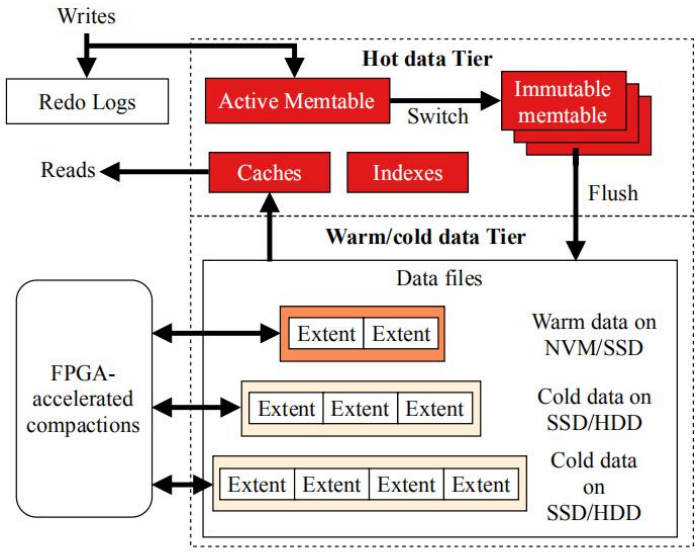


Figure 2: The architecture of X-Engine.

基本思路还是基于 LSM, 内存中的数据被定义为 Hot data;
定期刷新内存的数据到 Level0, 被定义为 Warm data;
下面各层是 Cold data;
其中 LSM 的 compaction 阶段利用了一系列的优化进行了加速;

每个表会被分为多个子表, 每个子表都会单独维护一套上述结构;

写会先做 log, 然后修改 active memtable, 当其大小到达一定大小后, 会转变会 immutable memtable, 然后会被定期刷到 Level0;
读会先走 cache, 没命中会从 index 查询需要 key 所在的 extent 位置, 然后进行读取;
FPGA 用来独立执行 compaction 阶段, 好处会在后面说;

设计

下面分别从 读, 写, flush & compaction 三个阶段说明 X-Engine 做了哪些优化;

读

先介绍自定义的 extent 格式, 如下:

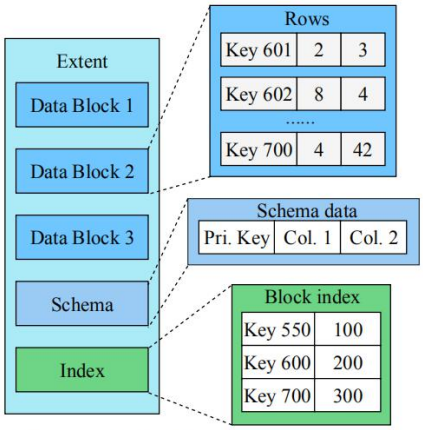


Figure 4: The layout of an extent.

先不对 extent 做过多解读，后续几个优化都会利用到它的格式；

为了加快查询，提高访问冷数据的速度，引入了 cache；
整个 cache 结构如下图：

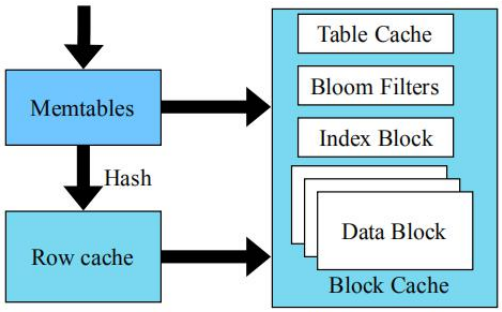
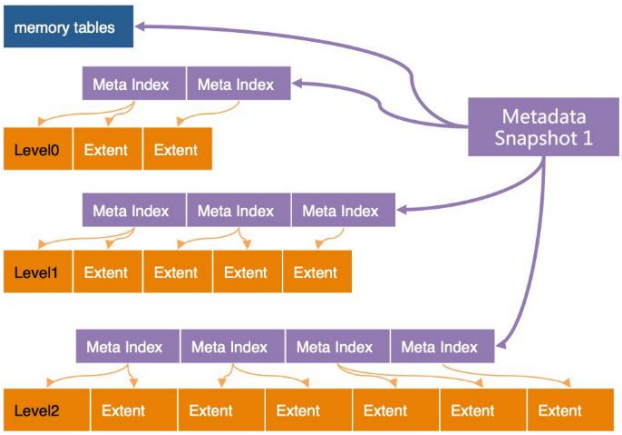


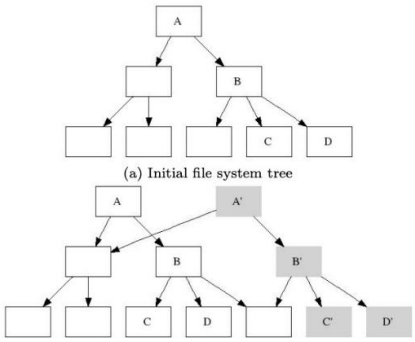
Figure 5: Structure of the caches.

Row cache 用来覆盖点查，LRU 实现，缓存 Level0 及更深的数据；
Block cache 则是 data block 粒度的 cache，用来覆盖范围查询；
Block cache 内用了一些常见的加速优化，如内部索引，bloom 过滤器；

对于 Cache 未命中的数据，则需要去对应的 extent 中查找；
为了加速这个过程，X-Engine 维护了一颗 extent 的树，用来快速定位需要的 extent 位置：



因为 compaction 操作可能会移动 extent，所以会修改这棵树；
为了在修改这棵树的时候，不会阻塞查询操作，使用了多版本的方法；



如上图，在修改树节点的时候用 copy-on-write 的方式；
这样在 compaction 未生效前，查询还能在老的树上进行；
然后再通过 GC 的方式来回收不需要的树节点；

compaction 的时候会修改 extent 以及其 data block，造成已经缓存的 data block 失效；
这个过程可能导致 cache 命中率降低；
为了解决这个问题，会在 compaction 的时候检查对应的 data block 是否被 cache；
如果在 cache 中，则会用 compaction 后的结果对 cache 中的 data block 进行替换，而不是直接将其标记为失效；

写

Memtable 由 skiplist 实现，目前的 skiplist 会有热点 key 的问题；
TODO

一些传统的存储引擎比如 InnoDB 是 one-thread-one-transaction 的方式处理事务；
该方式会让大量线程资源浪费在 commit 时等待磁盘 I/O；
X-Engine 把 commit 阶段从写事务中剥离出来，丢入一个 commit 队列异步执行；
这样线程处理完事务的其他阶段，把 commit 丢出去异步执行后，又能去处理新的事务了；

在异步处理 commit 时，还把其进行了拆分，分成了 4 个阶段，如下图；

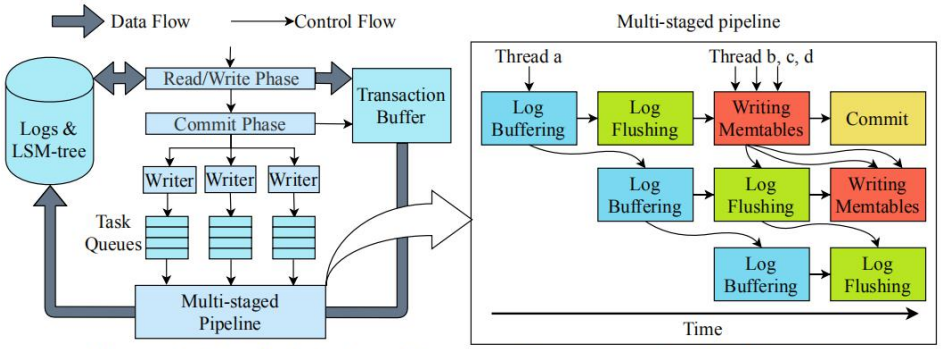


Figure 3: Detailed design of transaction processing in X-Engine.

不同阶段有不同的瓶颈，如磁盘 I/O，内存访问等；
在执行不同阶段时，采用流水线的方式，尽量让不同阶段同时执行；
这样能让系统不会被卡在某一个瓶颈上，提高整体内存和磁盘利用率；

Flush & Compaction

Flush 其实就是把 immutable memtable 转换为 extents，然后放入 level0;
Flush 之前 level0 的所有 extents 是有序的，放入新的 extents 后，有序性可能被打破;
所以 flush 之后会在 level0 内部进行一次 compaction，被称为 intra-level0 compaction;

在 compaction 过程中，会利用 extent 的格式，来做 data block 的复用，如下:

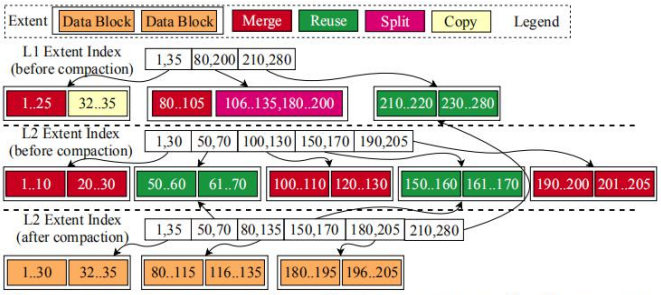


Figure 8: An example of data reuse during compactions.

不需要和其他 extent 做合并的 data block，就留在原地，或者直接 copy 到目标地点;

另外把 compaction 的过程分为 3 步，1) 读入，2) 合并，3) 回写;
可见 1 和 3 都是重 I/O 的操作，2 是重计算;
因此这个过程整体是以异步的方式在进行 I/O，防止线程资源耗在等待;

另外 compaction 会消耗计算资源，于是把 compaction 工作交给单独的硬件 FPGA 执行;
CPU 用来分配工作，然后交给 FPGA 执行;
FPGA 和 CPU 协同工作的具体细节论文中没有展开;

Compaction 被分为了这么几种类型:

- Intral-level0 compaction: 上面已经介绍;
- Minor compaction: 两个相邻 level(最底层除外);
- Major compaction: 最底层和它上层;
- Self-major compaction: 最底层内;

当某个层的数据总量或者 extents 的数量到达某个阈值时，会触发;
被触发的 compaction 会作为任务丢到任务队列内;
每个任务有优先级，取决于配置;

当 compaction 被执行时，会选取该 level 内最冷的一批 extents;
最冷目前由该 extents 在窗口时间内被访问的次数决定;
然后进行合并，并被放置入下一层;
这样能够使得冷数据被放在更底层，而相对热一点的数据在上层;

TODO: 尝试精确到 record 的粒度，转换为”该条 record 未来会不会热”的二分类问题，利用机器学习来判断;

总结

下面是优化的总结

Table 1: Summary of optimizations in X-Engine.

Optimization	Description	Problem
Asynchronous writes in transactions	Decoupling the commits of writes from the processing transactions.	Tsunami
Multi-staged pipeline	Decomposing the commits of writes to multiple pipeline stages.	
Fast flush in $Level_0$	Refining the $Level_0$ in the LSM-tree to accelerate flush.	Flood discharge
Data reuse	Reusing extents with non-overlapped key ranges in compactions.	
FPGA-accelerated compactions	Offloading compactions to FPGAs.	
Optimizing extents	Packaging data blocks, their corresponding filters, and indexes in extents.	Fast-moving current
Multi-version metadata index	Indexing all extents and memtables with versions for fast lookups.	
Caches	Buffering hot records using multiple kinds of caches.	
Incremental cache replacement	Replacing cached data incrementally during compactions.	

下面是用不用 FPGA 的吞吐，从下到上分别为：极小值，1/4 值，中位数，3/4 值，极大值；

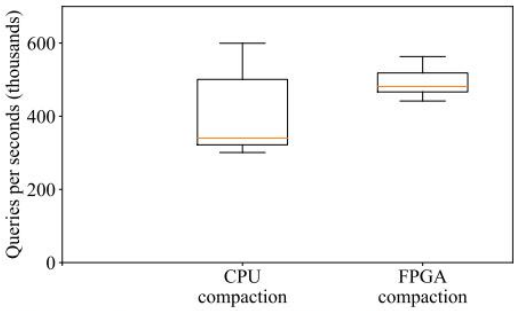
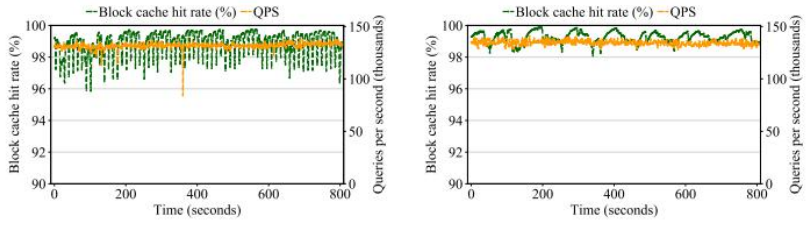


Figure 15: Throughput of MySQL (X-Engine) with and without FPGA offloading for compactions.

下面是 incremental cache 对延迟和 cache 命中率的影响

Figure 16: Throughput of compactions with different percentages of distinct records.



(a) Without incremental cache replacement. **(b) Wth incremental cache replacement.**

Figure 17: The block cache hit rates of X-Engine while processing the e-commerce workload.