Bonsai Design

- Index Layer
  - **Unified Indexing**
- Log Layer
  - **Epoch-based Batch Log Persistent**
  - Per-CPU Overflow Pages（小优化）
- Data Layer
  - **Pre-split Optimization**
  - **Write-optimzed Epoch-based Self-invalidation**
  - Parallel Cache-Friendly Lazy-Persist PNode（小优化）

Index Layer (with Shim Layer)

Unified Indexing

使用统一的一个index，将key映射到pnode1或log。

```
/* 3 cachelines */

struct shim_leaf {
    uint32_t bitmap;
    uint8_t fgprt[12];

    struct oplog *logs[12];
    struct pnode *pnode;

    spinlock_t lock;
    seqlock_t seq;

    struct shim_leaf *next;
    pkey_t fence;
}
```

Motivation

传统方法通常采用：

- 在DRAM中保存所有Key和Value指针（如old bonsai）。缺点：空间占用很大，不实用。
- 读时等待，直到日志全部刷完。（如Black-box Concurrent Data Structures for NUMA Architectures, ASPLOS 2017）。缺点：NVM刷日志远远慢于在DRAM中插入，等待时间很长。
- Separate Indexing，即使用一个Data Index和一个Log Index。Log Index索引尚未flush完毕的数据，Data Index索引已经flush的数据（从key映射到PNode）。查询时，先查V-Index，后查NV-Index。（如TIPS）。缺点：存在查两次的问题。

关键：**确保所有PNode的fence key也驻留在Index Layer中**。

基本操作

Upsert(key, log)

如果节点已满，prefetch所有logs后排序，找到分裂点，然后进行分裂。首先通过fgprt，查是否已存在该key。然后通过更新log指针和bitmap进行插入。

Lookup

首先通过fgprt找log层有没有。若没有，则去pnode层查找。

几个小优化

Leaf Prefetch

shim_leaf和pnode之间存在一种ordering：先shim_leaf，后pnode，无法充分利用MLP。因此，在index layer存储packed双指针，一半是shim_leaf的偏移，一半是pnode的偏移。然后在开始lookup前prefetch shim_leaf和pnode的metadata cachelines。

Range Update

首先确保shim_leaf里面的所有log对应的pnode全是一样的。然后做两次分裂，并update pnode。

Checkpoint-based Batch Remove

将已刷回的日志从shim layer删除具有很大开销。

bitmap中，每个log占2 bit。00、01、02。00表示未占用，01，02表示log的checkpoint。插入时，读取当前的checkpoint值（模2）。在checkpoint t

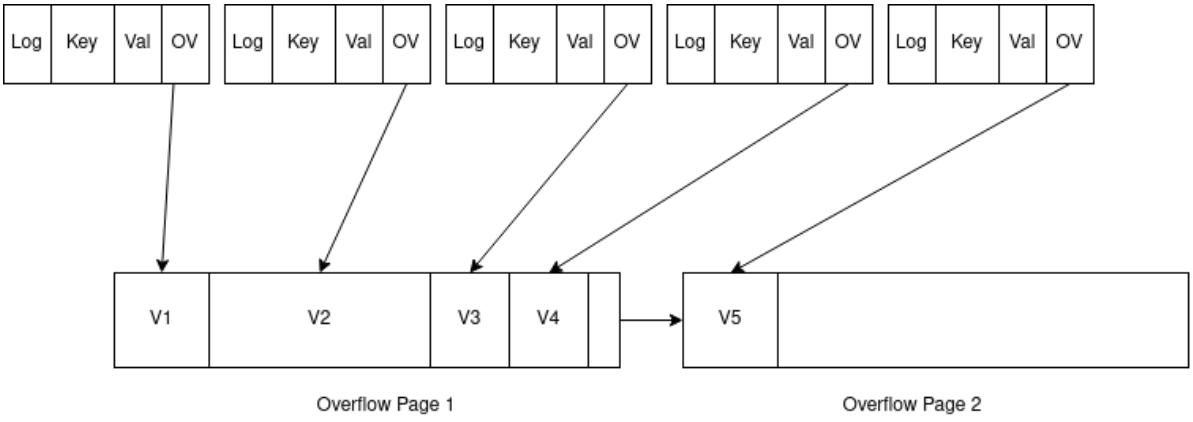
结束后，批量修改所有shim_leaf的bitmap，将t的都变成0即可。

Log Layer

Epoch-based Batch Log Persistent

略

Per-CPU Overflow Pages（小优化）



得益于log本身是顺序的，写溢出的Value无需内存分配，而且是顺序写，开销很低。

Data Layer

Pre-split Optimization

通过Flush前的预分裂（Split and Recolor操作），将数据均匀分布在各个DIMM上，**提升DIMM-level Parallelism**。并**让Flusher负载均衡**。具体算法：

令$D(i) = |\{a_k | a_k \in I \wedge c(I) = i\}|$，$E(I) = |a_k | a_k \in I|$。将原问题转化为一个带约束的单目标优化问题：现在需要一种SR的操作序列，该序列需要保证最终能满足$max_i(D(i)) \leq \frac{\Sigma_{i=1}^{k} D(i)}{k} + \epsilon$，并使SR次数最少。

下面将ϵ约束后的问题进行求解。令$P = \frac{\Sigma_{i=1}^{k} D(i)}{k} + \epsilon$

- 首先，对每种颜色i，得到其对应的所有区间构成的向量$V_i = [I_{i1}, I_{i2}, I_{i3}, I_{i4}, \ldots, I_{i|V_i|}]$，并对$V_i$中的所有区间按里面包含的数字个数（即$E(I)$）从大到小排序。
- 初始化items向量和boxes向量。
- 对每种满足$D(i) > P$的颜色i，将V_i取出前t个：$I_{i1}, I_{i2}, \cdots, I_{it-1}, I_{it}^*$。（最后一个可能不完全等于$I_{it}$），使得$E(I_{i1}) + E(I_{i2}) + \cdots + E(I_{it-1}) + E(I_{it}^*) = P$。将这$t$个区间每一个都作为一个MIN-FIBP问题中的item，大小为在各自范围内的数字的个数（即$E(I)$），全部加入items向量。每个都对应一次SR操作（共t次）。
- 对每种满足$D(i) < P$的颜色i，创建一个大小为$P - D(i)$的box，加入boxes向量。

注意到boxes向量里面的元素个数非常少，因此装满所有box造成的SR操作的远远少于刚才划分时的SR操作次数。所以，按任意顺序将item全部移到boxes即可。

Data migration

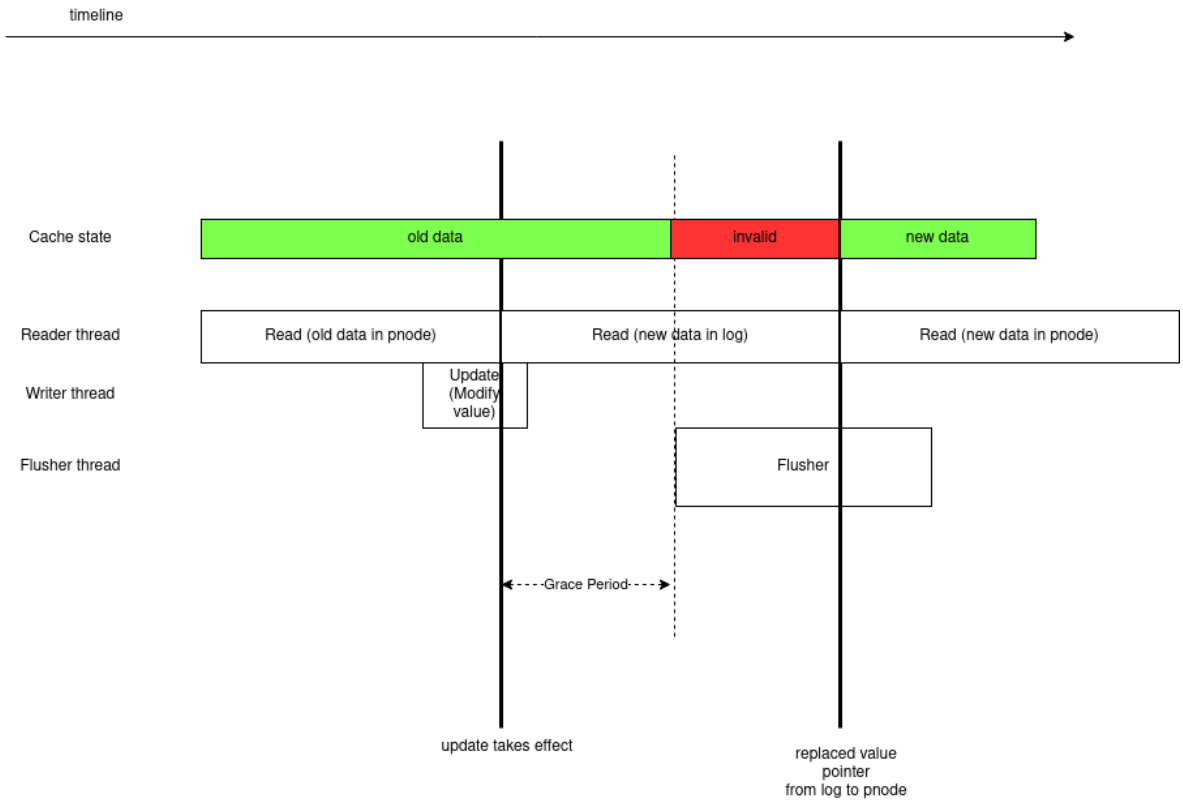
迁移对象：

- PNode Metadata
- Entry

第一次读就直接迁移。

Write-optimzed Timestamp-based Self-invalidation

本地缓存超时时间为t_{GP}。**Flush线程刷t_{GP}以前的日志**。示意图：



主要原因：当update生效时，pnode的value此时将不再会被访问到。当t_{GP}时间过了，所有本地缓存都已经invalid了。因此，这个时候之后再改pnode的value是安全的。改完value之后，把value指针指向pnode，所有NUMA节点访问到的都是最新的数据。

小优化：Write-optimized Epoch-based Self-invalidation

问题背景：

- rdtscp指令本身开销较大，Skylake上21 uops，30 cycles（https://www.agner.org/optimize/instruction_tables.pdf）。
- rdtscp指令对pipeline影响较大，因为他是serializing的。
- 需要在pnode的每个entry，再加一个8B存储timestamp，不cache-friendly。

方法：

- 每隔t_e时间（可以选为1s），全局epoch自增。
- 每个entry里面用2B存经过的epoch数目。由于value是一个指针，有效地址占据48bit，因此取其前2B来存epoch数目，无额外的空间开销。
- 每次检查cache是否合法时，首先读取全局epoch e_g，以及entry里面存储的entry对应的epoch e_e。如果$e_g - e_e \geq 2$，则说明cache不合法。这样，每个entry的存活时间在$[t_e, 2t_e)$内。
- 每次刷log的时间间隔设置为$2t_e$即可。
- **回环问题**

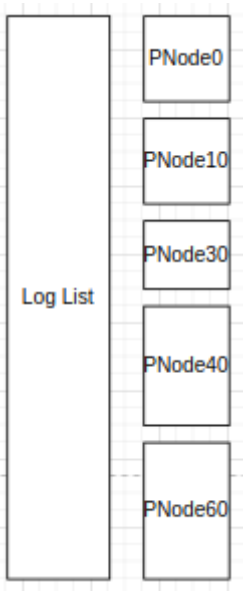
Parallel Cache-Friendly Lazy-Persist PNode（小优化）

Motivation

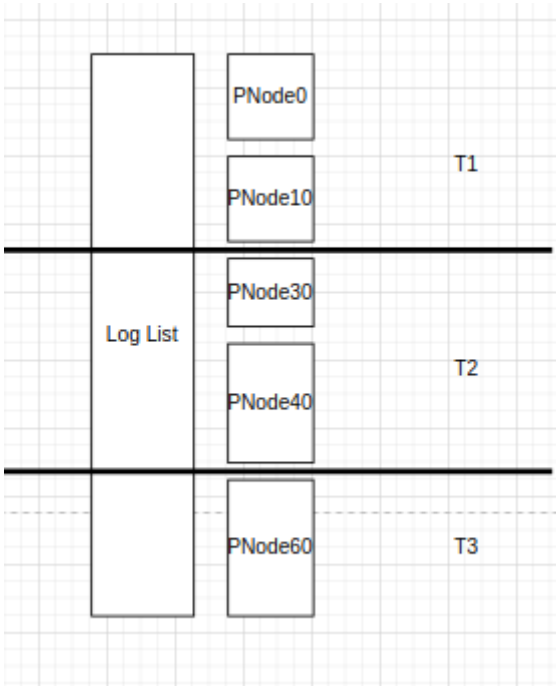
- Cacheline thrashing
- 锁竞争开销

设计

- Log List Partition。（少量查找即可，find_leaf的开销被减少很多。集中起来进行顺序的Search对search layer cache 友好。）



- Job Assignment.
-



- Flush。
  - Parallel：由于PNode只分裂，不合并，而各个线程没有相同的PNode，因此无需任何同步机制。
  - Lazy-Persist：对每个线程，插入多个数据之后，再做一次Persistent操作。具体来说，当处理完一个PNode，进入下一个PNode之前，对前一个PNode按照先Entry，后Bitmap的顺序持久化。如果出现崩溃，相当与少插入了连续的一段数据。而这个可以通过日志来恢复。
  - Cache Friendly：预取三个PNode。