Tips: Making Volatile Index Structures Persistent with DRAM-NVMM Tiering

Thursday, November 25, 2021 10:01 AM

https://www.usenix.org/system/files/atc21-krishnan.pdf

背景介绍：

作者提出了一个叫TIPS的框架，用于把volatile index持久化到PMEM上。不需要太了解volatile index的实现，就可以把它转换成persistent index，同时获得很不错的性能。作者做了一些测试，例如，相比针对PMEM优化的B+Tree index (BzTree和FastFair)，hash index (CCEH和Level Hash)以及Trie index (WOART)，有3~10倍的性能提升。

实现一个高性能持久化的index，并且把它做成熟和稳定不是一件容易的事情。但volatile indexes的实现经过几十年的积累，已经很成熟，并且应用于各种kv存储和数据库类的应用。因此作者提出一种系统化的方式，把volatile indexes转换成persistent indexes，主要优点：

- 转换比较简单，只需要修改少量代码
- 使用volatile indexes的应用，可以很方便迁移到对应的persistent indexes上

实现细节：

对于如何改装一个volatile index，论文中给了一个例子，看上去还是比较简单的。主要是把内存分配改用tips的接口，然后记录下undolog。

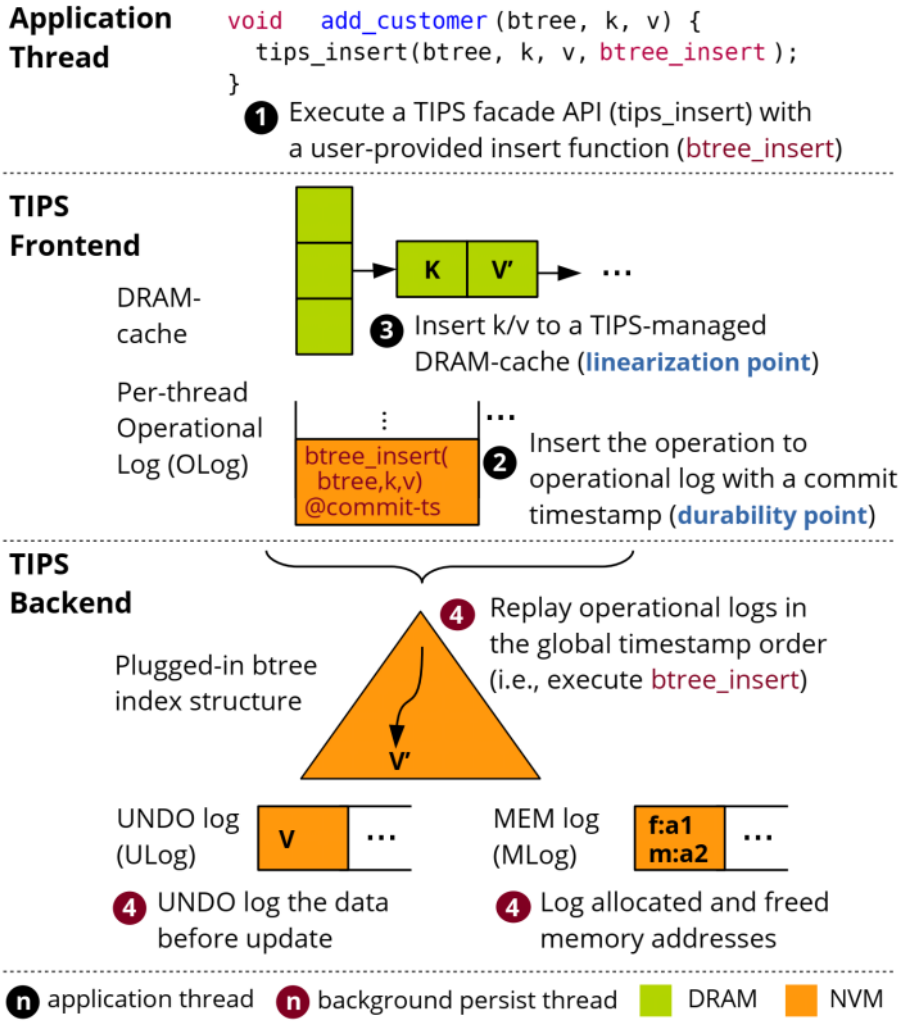
```
1  void hash_insert(hash_t *hash, key_t key, val_t value) {
2      node_t **pprev_next, *node, *new_node;
3      int bucket_idx;
4      pthread_rwlock_wrlock(&hash->lock);
5      // Find a node in a collision list
6      // Case 1: update an existing key
7      if (node->key == key) {
8          // Before modifying the value, backup the old value
9  +       tips_ulog_add(&node->value, sizeof(node->value));
10         node->value = value; // then update the value
11         goto unlock_out;
12     }
13     // Case 2: add a new key
14     // Allocate a new node using tips_alloc
15 +   new_node = tips_alloc(sizeof(*new_node));
16     new_node->key   = key; new_node->value = value;
17     new_node->next  = node;
18     // Backup the prev node before modifying it
19 +   tips_ulog_add(pprev_next, sizeof(*pprev_next));
20     *pprev_next = new_node; // then update then the node
21 unlock_out:
22     pthread_rwlock_unlock(&hash->lock);
23 }
```

作者也给出了，对于常见volatile index修改，代码的改动量

Indexes	Concurrency Control	LoC
Hash table (HT)	Readers-writer lock	5/211
Lock-Free HT (LFHT) [51]	Non-blocking reads and writes	5/199
Binary Search Tree (BST)	Readers-writer lock	5/203
Lock-Free BST (LFBST) [12]	Non-blocking reads and writes	5/194
B+tree	Readers-Writer lock	8/711
Adaptive Radix Tree (ART) [40]	Non-blocking reads/blocking writes	9/1.5K
Cache-line Hash Table (CLHT) [16]	Non-blocking reads/blocking writes	8/2.8K
Redis [8]	Blocking reads and writes	18/10K

Table 2: Lines of code (LoC) to convert volatile indexes using TIPS.

之后作者很大篇幅讲了之前各种方案的问题，大家有兴趣可以看下原文。我们直接来看TIPS的实现



这是TIPS的架构图，同时以插入一对kv为例子，来理解每一个模块的设计。

(1) TIPS提供的API是把volatile的API做一层封装，有点类似装饰器模式。

(2) 持久化操作记录到PMEM上的OLog中。为了优化性能，OLog是线程独立的。

(3) 把KV记录插入到DRAM cache中，这样一个写操作就可以完成返回了。这里DRAM中是一个concurrent hash map。这种写DRAM+persistent log然后返回，让后台线程再去搬数据的方式很常见，比如rocksdb也是这样。

(4) 后台线程replay Olog, 把KV插入到改装过的persistent index中。其中用到了ULog和Mlog

a. ULog (Undo Log) 用来保证数据的完整性。这里TIPS做了一些优化来减少ULog的记录以提升性能:

i. 判断需要做undo数据的ts (记作 alloc-ts)是否大于OLog最近一次回收的ts（记作reclaim-ts）。如果alloc-ts > reclaim-ts，那就不需要记录undo log，因为即使挂掉重启，也可以通过OLog把数据恢复回来。

ii. 检查是否已经对这个数据做过undo，如果有的化，也不需要再做一次。

iii. undo log只需要在定期做reclamation的时候做一次持久化就可以。如果中间挂了，还是可以根据上次reclamation的数据，然后重放OLog来得到最新内容的。

b. MLog (MEM Log)记录分配和释放的PMEM地址，主要是为了

i. 防止异常重启后，PMEM的内存泄漏，TIPS可以利用MLog记录的信息，把没被使用的空间释放掉。

ii. 通过延迟free PMEM（直到OLog里相应的操作记录被使用），防止重启后double free的问题。

iii. 使用了pmemobj提供的持久内存分配器。

看到这里，我第一个疑问是，TIPS怎么做scan？因为DRAM中是一个非排序的hash。文章4.1.6给出了它的方案
它首先会去persistent index，比如B+tree中去做scan，然后扫一遍OLog，把属于scan range内的KV找出来，合并到persistent index scan的结果中去。为了减少扫一遍OLog的开销，TIPS做了一些优化，记scan操作开始的时间为scan-ts， 记OLog中record的时间为commit-ts。如果commit-ts > scan-ts，则忽略这个record，因为它是scan开始后才插入的KV。
论文作者观察到扫OLog的开销不大，因为：

1. 跳过了时间戳大于scan开始时间的记录
2. 后台线程可以及时把OLog的数据重放到persistent index中，所以OLog的数据量不大
3. PMEM的读性能非常好

另外TIPS还有两个比较有意思的优化点：

1. 自适应的后台线程数。因为后台线程要写PMEM，所以很可能会比前台操作慢。所以通过监控前端和后端的吞吐，来动态调整后台工作线程的数目。
2. 采用ORDO来，来解决rdtsc在多个CPU core之间可能有skew的问题。

实验结果:

文章从多个角度，做了性能测试。跟现有方案比较，看上去还是相当不错的。

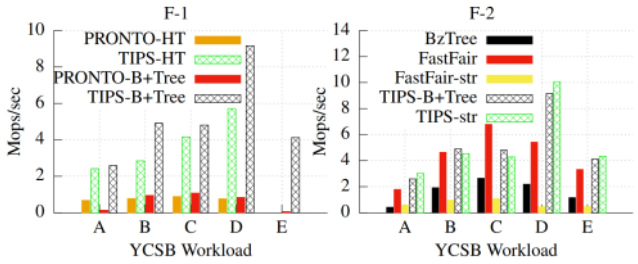


Figure 4: Performance comparison of TIPS against PRONTO for Hash Table (HT) and B+Tree (F-1) and TIPS-B+Tree against the NVMM-optimized B+Tree indexes– FastFair and BzTree (F-2).

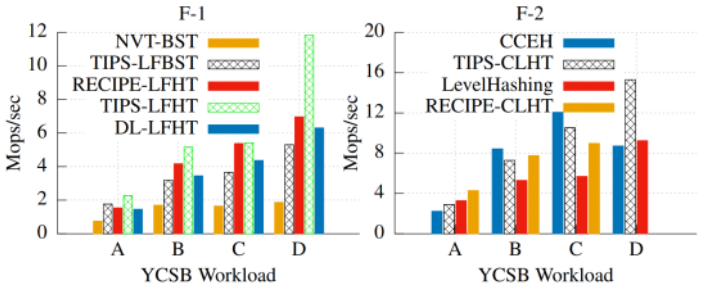


Figure 5: Performance comparison of TIPS with NVTraverse (F-1), RECIPE (F-1, F-2) and TIPS-CLHT with NVMM-optimized hash indexes– CCEH and LevelHashing (F-2).

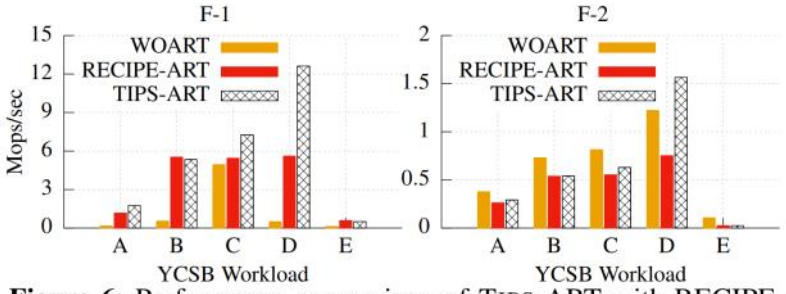


Figure 6: Performance comparison of TIPS-ART with RECIPE-ART and WOART for 32 threads (F-1) and 1 thread (F-2).

也做了scalability和热点数据的测试

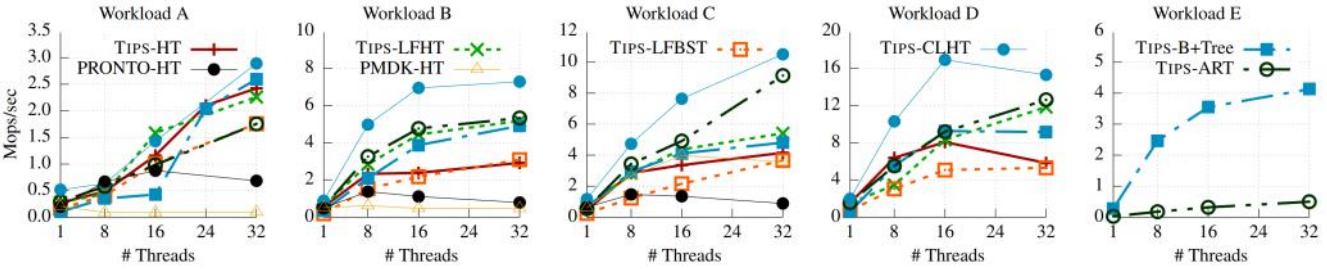


Figure 7: Scalability of TIPS-HT (RW lock), TIPS-B+Tree (RW lock), TIPS-LFHT, TIPS-LFBST, TIPS-CLHT and TIPS-ART.

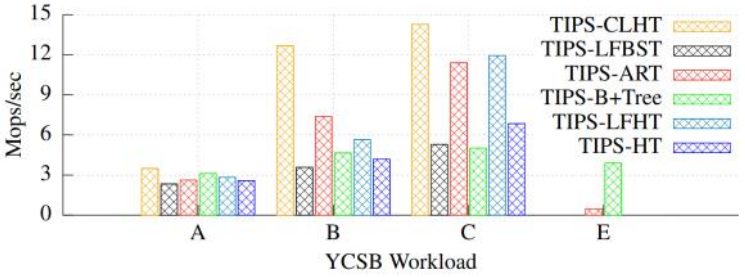


Figure 8: Performance of TIPS indexes for Zipfian workloads.

总结：

这个paper提供了一个很好的思路来实现持久化index，有些具体的优化技术，比如Olog/Ulog/Mlog, ORDO，动态worker数等都是可以借鉴的。唯一的遗憾是没有开源:)