Write-optimized Timestamp-based Self-invalidation Technique

Write-invalidate protocol

主要问题：Message passing次数太多。写者需要通知所有读者Flush掉cacheline，消息传递的开销很高。（Bonsai中，对应的是Remote NUMA write的开销）。

Self-Invalidation

思想：将Writer通知Reader Flush，转变为Reader主动Flush。Reader使用启发式方法投机地刷Cacheline。结合DirCC，会少做很多次Message passing。

Last-Touch

需要记录的信息很多，算法复杂，且仍然需要Message passing（次数减少了）。

Timestamp-based Self-invalidation

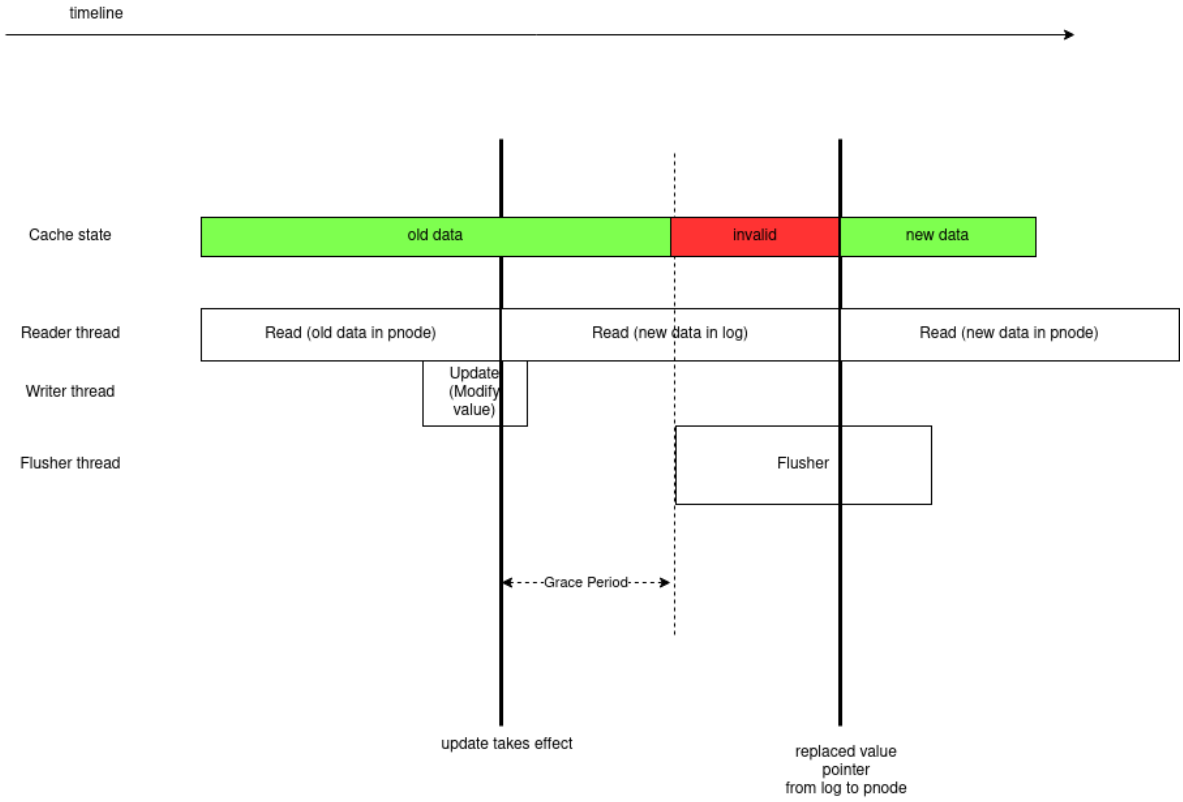
Library Cache Coherence

Reader持具有超时时间的Cacheline，超时则算作不命中。Writer等到所有Reader全部超时，再写入DRAM，确保SC。

对应在Bonsai中，PFlush线程等所有Reader手中的对应块全部超时，然后再写入。缺点：对于一直访问热点数据的情况，写入会非常慢。

Write-optimized 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：

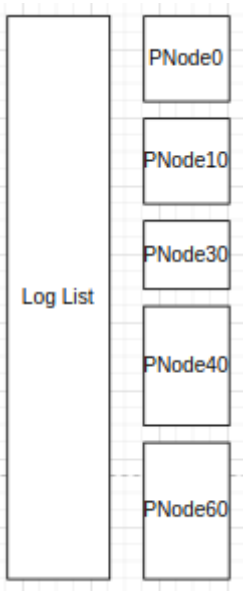


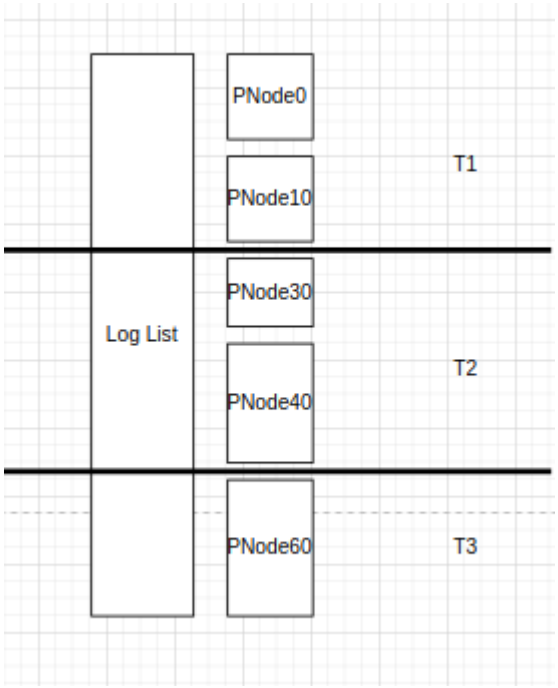
主要原因：当update生效时，pnode的value此时将不再会被访问到。当t_{GP}时间过了，所有本地缓存都已经invalid了。因此，这个时候之后再改pnode的value是安全的。改完value之后，把value指针指向pnode，所有NUMA节点访问到的都是最新的数据。

Parallel Cache-Friendly Lazy-Persist PNode

考虑只有一个NUMA节点的情况。在日志阶段已经对所有操作进行过排序。按下列步骤处理：

- 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。

Per-Node Data Layer

目的：优化后台线程写操作，让写操作都发生在本地。

Data Layer设计为Per-Numa Node。

Volatile view

- 插入：插入操作直接在本地进行。因为shim layer里面会调整，指向新的NUMA节点里的PNode，旧的Entry不再对上面可见，因此没问题。
- 删除：从上面删掉之后，旧的Entry自然不再可见，因此没问题。

Non-volatile view

关键是recovery。如何从PNode重建上面的Index。两个问题：

- 多个NUMA节点里都有一个key，哪个是最新的？——**存Timestamp**
- **删除掉之后**