Hybrid Transactional Index with Scalable Phantom Avoidance

Yutaro Bessho
NTT
Japan

Hideyuki Kawashima
Keio University
Japan

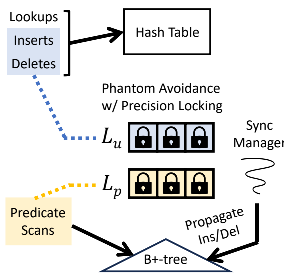

Figure 1: Overview of Griffin.

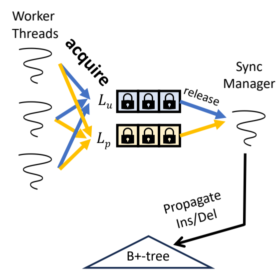

Figure 2: Precision Locking in the original Griffin.

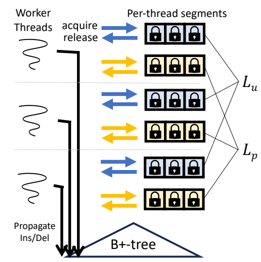

Figure 3: Proposed design (Griffin-PPL).

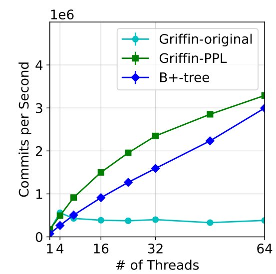

Figure 4: Microbenchmark throughput.

1 Introduction

Griffin [3] (Fig. 1) is a hybrid transactional index structure with a hash table and B+-tree, each of which manages the key set of a table. The hash table processes single-key operations (lookups, inserts and deletes), while the B+-tree processes predicate scans. Inserts and deletes to the hash table are propagated to the B+-tree by an asynchronous thread, *sync manager.* The key benefit of Griffin is that it processes single-key operations in $O(1)$, while a B+-tree does in $O(\log N)$ (N: table size). To achieve serializability, Griffin avoids phantoms with *precision locking* [1]. [3] shows that Griffin outperforms a B+-tree in workloads with few inserts and deletes.

2 Problem

In workloads with more inserts or deletes, precision locking causes tremendous overhead for Griffin.

Overview of precision locking. Inserts, deletes, and predicate scans are guarded with a lock. A lock is acquired before accessing data and released after the transaction terminates. Locks are managed in two lists: one for inserts/deletes (L_u), and one for predicate scans (L_p). A pair of insert/delete and predicate scan that conflict cannot hold their locks at the same time; before acquiring a lock, an insert/delete searches L_p for a conflict, and aborts on detecting one. [1] A predicate scan searches L_u and aborts on any conflict.

Bloating lock lists. In Griffin, high loads of inserts/deletes can cause L_u and L_p to grow indefinitely, because: (1) *Locks in L_u are costlier to release than to acquire.* For correctness reasons, before releasing a lock in L_u, the insert/delete must be propagated to the B+-tree, which involves an $O(\log N)$ access. The cost of acquiring the lock is significantly smaller ($O(1)$). (2) *While multiple threads acquire locks, only one thread releases them.* As shown in Fig. 2, lock acquisition is done by worker threads, i.e., threads that process operations. Lock release is done by sync manager, the single thread that also propagates inserts/deletes. (3) *New locks can be acquired without limit.* Sync manager does not throttle worker threads in any way. (4) *The bloating of L_u can lead L_p to also bloat.* When there are many predicate scans as well, L_p can bloat because sync manager is busy releasing locks in L_u. Since every insert, delete, predicate scan searches the whole L_u or L_p on acquiring its lock, performance significantly degrades when they grow.

3 Proposal

We propose *Griffin-PPL (Parallel Precision Locking)* (Fig. 3). Most notably, it removes sync manager; its jobs of propagating inserts/deletes and releasing locks are instead processed by all worker threads in parallel. For parallelization, L_u and L_p are split into per-worker-thread segments. An insert/delete, or predicate scan acquires its lock in its per-thread segment of L_u or L_p, respectively, and it searches all segments of L_p and L_u, respectively, for conflicting locks. Every time a worker thread acquires a set number of locks in its segment of L_u or L_p, it tries to release as many locks as possible in the segment. Of the aforementioned reasons that L_u and L_p can bloat, Griffin-PPL addresses (2), (3), and (4). (2) Releasing locks is parallelized by worker threads. (3) When worker threads release locks and propagate inserts/deletes, they cannot process operations, i.e., lock acquisition is throttled. (4) Releasing locks in L_p is removed from the critical path, i.e., sync manager. (1) is not addressed because B+-tree updates remain $O(\log N)$, but some batch-update schemes might be exploited (future work). Fig. 4 shows the transaction throughput of a B+-tree, original Griffin, and Griffin-PPL in an insert-heavy (50% inserts, 50% predicate scans) microbenchmark. [2] Griffin-PPL performs the best while the original Griffin collapses due to the bloating lock lists.

References

[1] J. R. Jordan et al. 1981. Precision Locks. In *SIGMOD Conf.* 143–147.

[2] Per-Åke Larson et al. 2011. High-Performance Concurrency Control Mechanisms for Main-Memory Databases. *Proc. VLDB Endow.* (2011).

[3] Sho Nakazono et al. 2024. Griffin: Fast Transactional Database Index with Hash and B+-tree. arXiv:2407.13294 Accepted for presentation at IEEE eScience 2024.

[1] Although the mutual exclusion of precision locking can be achieved by blocking the issuer of the lock request, this work and [3] assume an abort of the issuer.

[2] As with [3], every insert or predicate scan is executed as a transaction, all data fits in memory, and no table or logging is involved. `B+-tree` avoids phantoms through repeating scans at commit time [2].