### Motivation

- **Approximate Query Processing (AQP)** uses **random samples**
  - to provide fast and approximate answers with error guarantees
  - existing solutions often make trade-off between
    - efficient online updates and
    - low response time

**Our goal:** design an index structure that can support AQP with all the three desired properties.

- Fast AQP query: sampling scales (almost) linear to sample size
- Query over latest updates
- Fast concurrent update

### Why concurrency is hard for aggregate B-trees?

- **Aggregate B-tree (example: uniform weights)**
  - Maintains sub-tree weights \( w_i \), along with page pointer \( c \)
  - \( w_i \) is the sum of weights in the sub-tree
  - Starting from root, randomly descend into sub-trees with probability \( w_i \)
  - It can be shown the leaf tuple sampled has a probability proportional to its weight
  - Weight updates must be applied atomically along a tree path from root to leaf where insertion happens

- **Baseline:** X-latch tree path for each update
  - Every update blocks every other thread
  - Sampling and update throughput drops under heavy update workload
  - DBMS with multi-version CC can further make decrease sampling throughput for old snapshots due to "live version bloom"

**Our solution:** AB-tree

- based on B-link tree in PostgreSQL 13
  - (available on Github: [https://github.com/zzy7896321/abtree_public](https://github.com/zzy7896321/abtree_public))

### Challenge 1: non-blocking weight updates

- **Higher Contention**
  - \( P_5 \)
  - \( 9 \) \( 12 \)

- **Lower Contention**
  - \( P_7 \)
    - \( 3 \) \( 2 \) \( 2 \)
  - \( P_8 \)
    - \( 7 \) \( 8 \) \( 9 \) \( 10 \) \( 11 \) \( 13 \) \( 14 \)

- Internal pages have higher contention for weight updates
- Root page is always contended in any update

**Can we update weights without X-latching the entire tree path?**

- Yes, use CAS with S-latch one page at a time!
  - S-latch guarantees no concurrent SMO while CAS is applied
  - Weight updater does not block others
  - Correctness of sampling? (see challenge 2)

### Challenge 2: weight consistency for sampling

- **Consistent weights needed for sampling purpose**
  - perform rejection sampling as in [Olken'93]

**Definition 1:** An aggregate B-tree \( T \) is said to be consistent for sampling purpose if and only if for any index tuple \( t \in T: w_1 \geq \sum_{i=1}^{n} w_i \).

**Natural idea is to update weights along the path before leaf insertion**

- However, it is incorrect!
  - Concurrent Structural Modification Operation (SMO) **may undo** the change
    - \( T_1 \): insert \( x_4 = 4 \)
    - \( T_2 \): insert \( x_5 = 5 \)

**Steps:**

1. \( T_1 \) increments \( w_{x_4} \)
2. \( T_2 \) increments \( w_{x_5} \)
3. \( T_2 \) splits \( p_2 \) and inserts \( x^* \)
4. \( T_1 \) inserts \( x^* \)

**Solution:** two-pass insertion

- **Pass 1:** regular key insertion
  - assign zero weight to new key
- **Pass 2:** descend in the tree again and modify weights
  - redo weight modification on certain pages in case of concurrent SMO
  - use page and tuple update counters to detect concurrent SMO (see paper for details)

### Challenge 3: sampling efficiency under MVCC

- **Sampling under an old snapshot with MVCC could suffer from "live version bloom"**
  - Many live versions of tuples are
    - not visible to that sampling thread
    - but are physically present in the index
  - \( \rightarrow \) high rejections rates \( \rightarrow \) decreased sampling throughput

**Solution:** build an in-memory multi-version weight store to allow

- Querying upper bound of weights under an old snapshot
  - Tight enough for minimizing rejection due to live version bloom
- No logging/persistence required
  - Only queries by active transactions
  - Old snapshots do not live across crashes
- Details in the paper

### Evaluation: insertion scalability

- **B-tree** is the original B-link tree without aggregates in PostgreSQL.
- Its insertion throughput is an upper bound.

**Conclusion:** AB-tree scales similarly to the original B-link tree while baseline cannot.

### Evaluation: read-write workload

- **B-tree** is the original B-link tree without aggregates in PostgreSQL.
- Its insertion throughput is an upper bound.

**Conclusion:** AB-tree can sustain a reasonably high insertion and sampling throughput when there are heavy updates while baseline can’t.

**Future direction:** we hope to use AB-tree to enable HTAP within AQP systems.