AB-tree: Index for Concurrent Random Sampling and Update

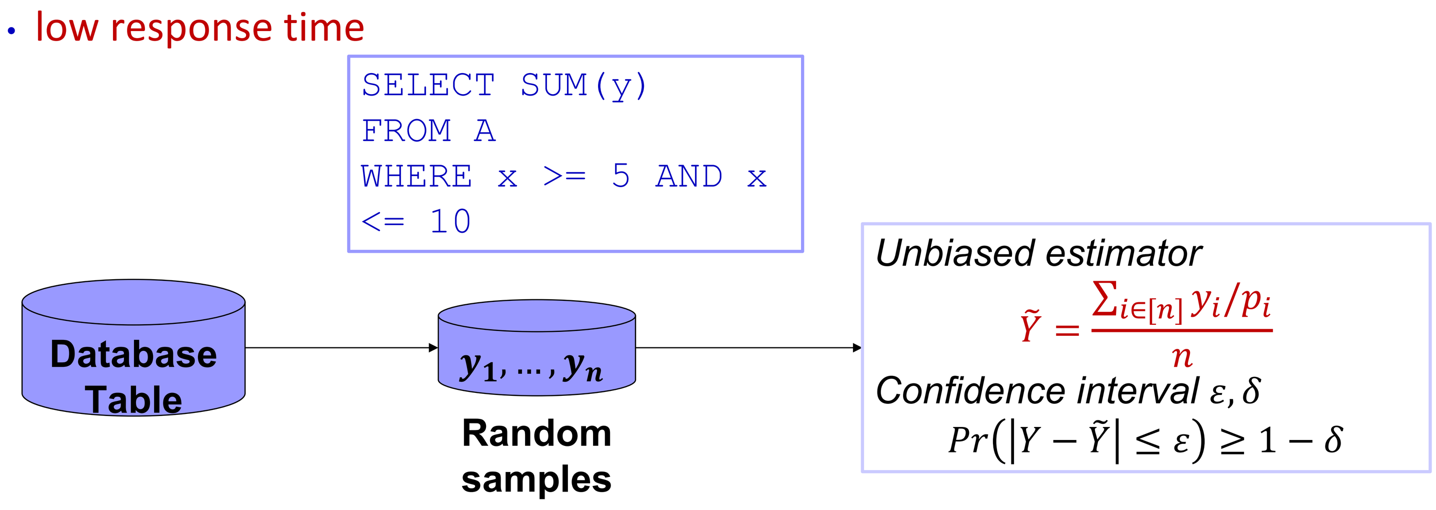
Zhuoyue Zhao
University at Buffalo
zzhao35@buffalo.edu

Dong Xie
Pennsylvania State University
dongx@psu.edu

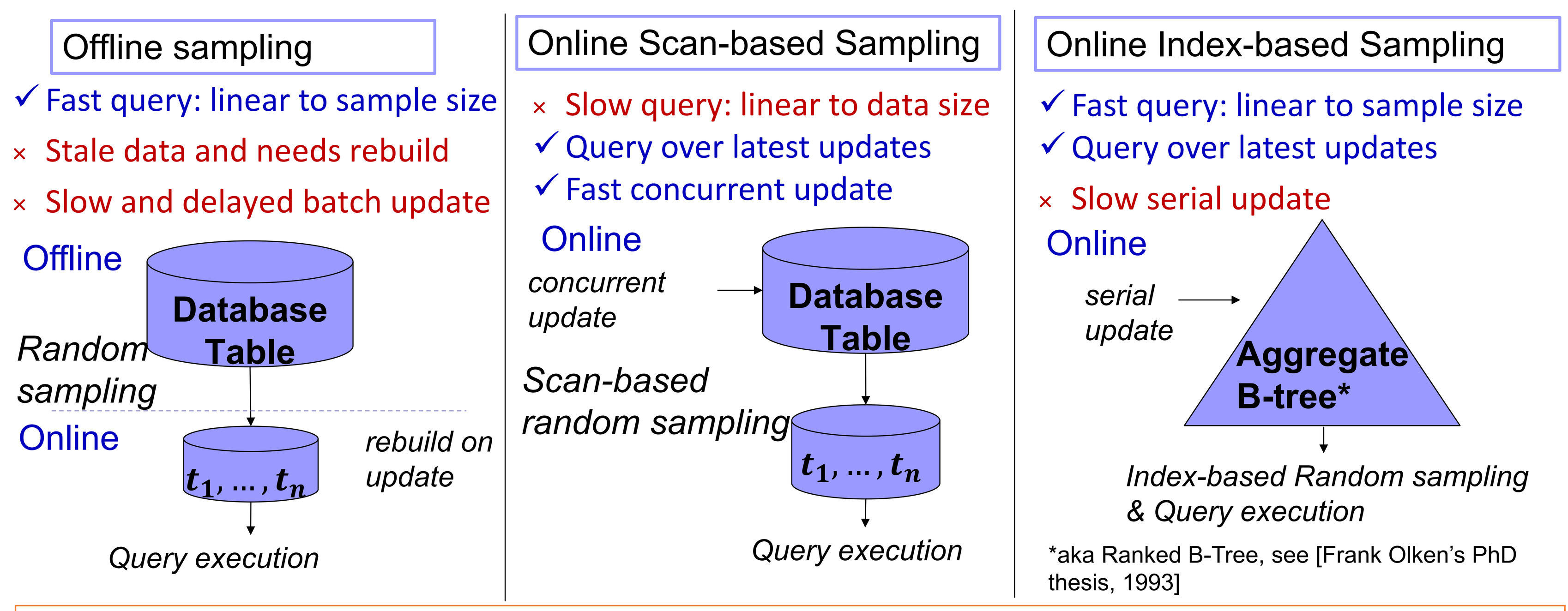
Feifei Li
Alibaba
lifeifei@alibaba-inc.com

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



- How do existing AQP systems perform random sampling?

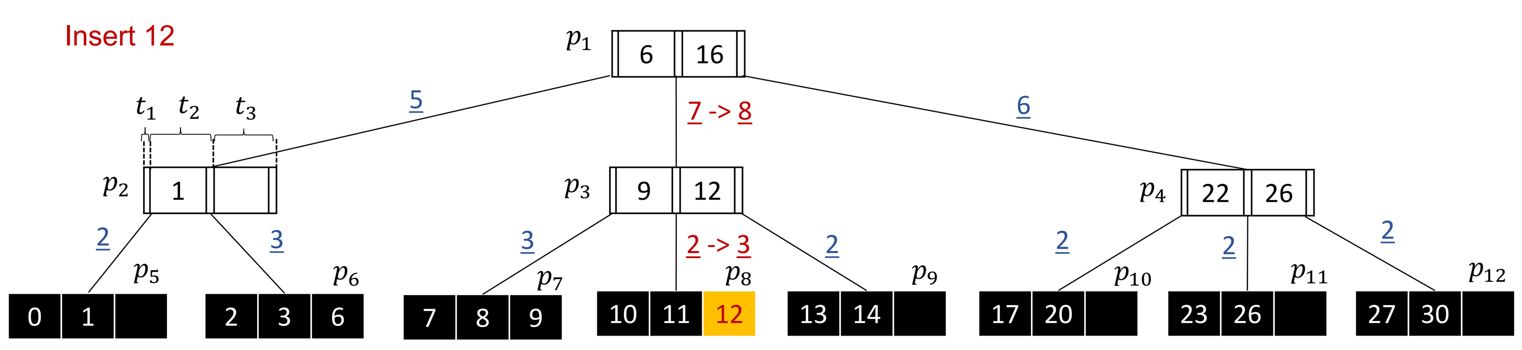


Our goal: design an index structure that can support AQP with all the three desired properties.

✓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?

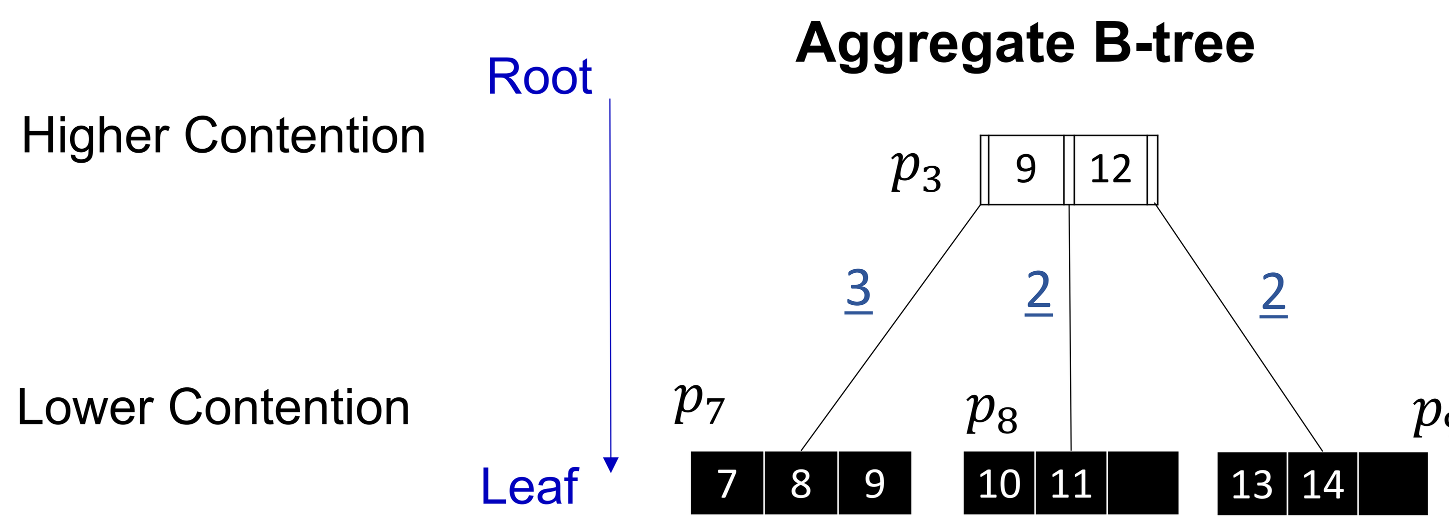
- Aggregate B-tree (example: uniform weights)
  - Maintains sub-tree weights w_c along with page pointer c
    - w_c is the sum of weights in the sub-tree
  - Starting from root, randomly descend into sub-trees with probability $\propto w_c$
    - It can be shown the leaf tuple sampled has a probability 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 bloat"

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

Challenge 1: non-blocking weight updates



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

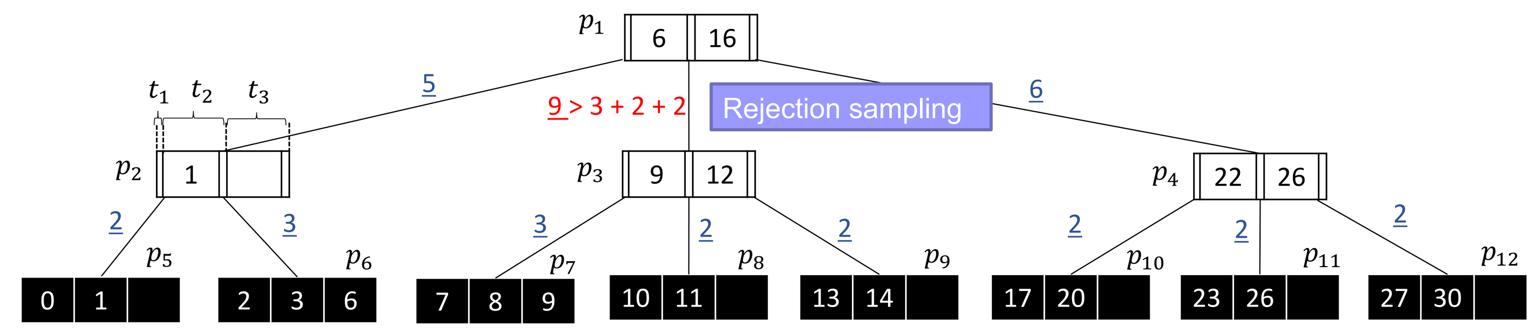
Can we update weights without X-latching the entire tree path?

- 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)

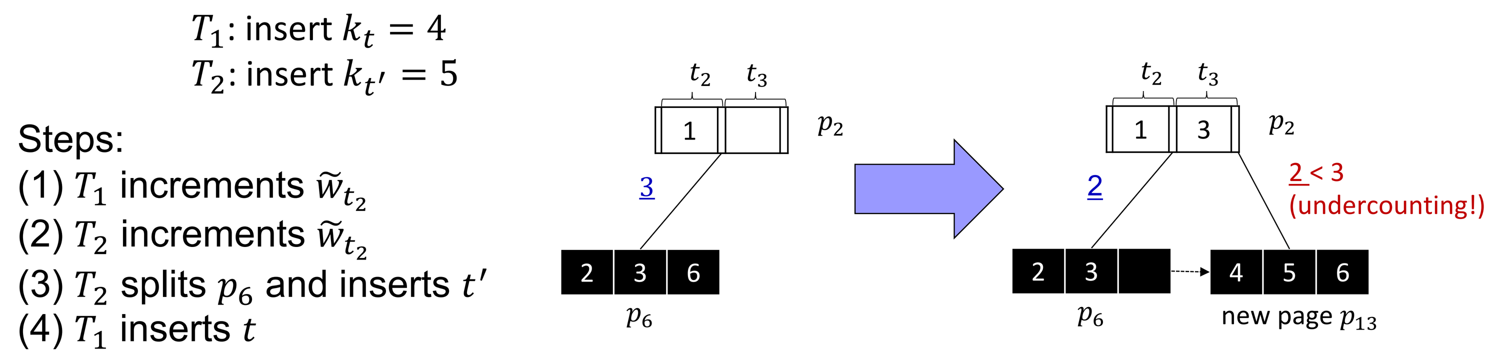
Challenge 2: weight consistency for sampling

- 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 $t \in T$: $\tilde{w}_t \ge \sum_{t' \in c_t} \tilde{w}_{t'}$.



- 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



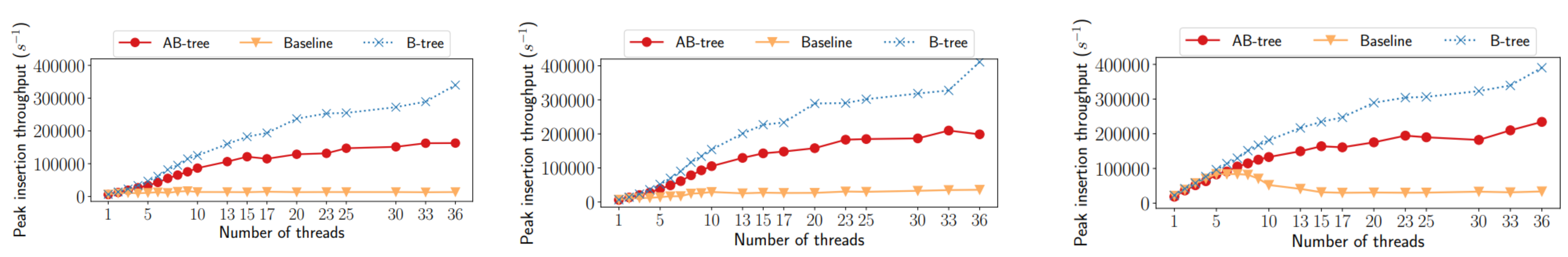
- Solution: two-pass insertion
  - Pass 1: regular key insertion
    - assign zero weight to new key
  - Pass 2: descend in the tree again and modify weights
    - redo weight modification on certain pages in case of concurrent SMO
    - use page and tuple update counters to detect concurrent SMO (see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

Challenge 3: sampling efficiency under MVCC

- Sampling under an old snapshot with MVCC could suffer from "live version bloat"
  - Many live versions of tuples are
    - not visible to that sampling thread
    - but are physically present in the index
    - → high rejections rates → 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 bloat
  - No logging/persistency required
    - Only queries by active transactions
    - Old snapshots do not live across crashes
  - Details in the paper

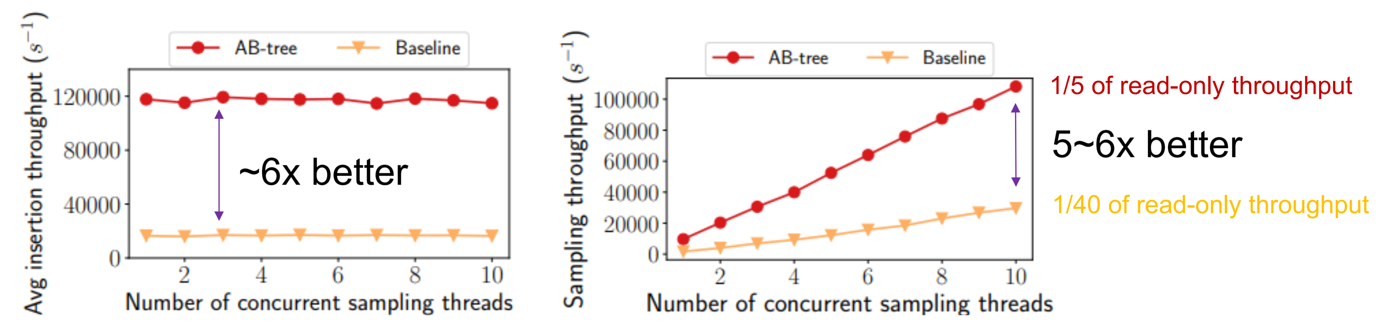
Evaluation: insertion scalability



(a) Small buffer (128MB) (b) Large buffer (32GB) (c) In-memory (32 GB, simulated with same seed)

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



Read-write workload with 10 insertion threads and varying # of sampling threads

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.