Paper Showcase

Introduction to DBMS
CS, NTHU
Spring, 2019

Main Fields

- Query Optimization (by YC Lin)
- Storage Engine (by YS Chang)
- Transaction Management (by YS Lin)

Query Optimization

Organized by Yi-Chun Chen

Paper Lists

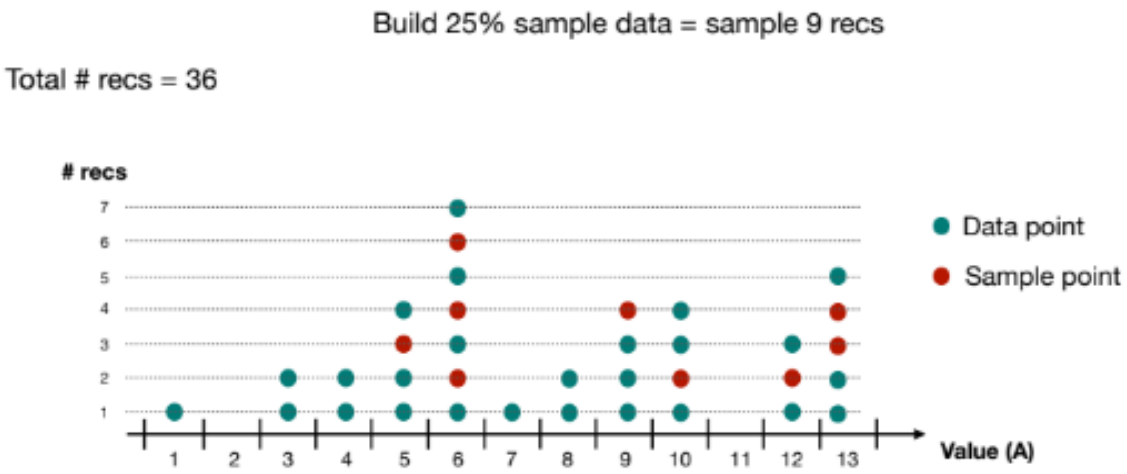
- [TODS'07] Optimized stratified sampling for approximate query processing
- [IEEE Data Eng. Bull.'99] Approximate Query Answering using Histograms
- [SIGMOD'90] Randomized Algorithms For Optimizing Large Join Queries
- [SIGMOD'18] AQP++: Connecting Approximate Query Processing With Aggregate Precomputation for Interactive Analytics
- [arXiv'18] Learning to Optimize Join Queries With Deep Reinforcement Learning
- [arXiv'19] Approximate Query Processing using Deep Generative Models

Optimized stratified sampling for approximate query processing

- Paper Link: http://ranger.uta.edu/~gdas/websitepages/preprints-papers/a9-chaudhuri.pdf
- Difficulty: **Easy**
- Problem:
  - Instead of spending lots of time to get the exact answer, OLAP application usually prefer to spend less time and get an approximate answer.
- Main Idea:
  - There have been several previous efforts to address this problem, ranging from using small random samples of the data, to other forms of data reduction techniques. In this article, we exclusively focus on the approach of using precomputed samples of data to answer queries.

Optimized stratified sampling for approximate query processing

- Challenges:
  - You need to implement the sampling method and notice how to sample data points(besides uniform sample), in order to get better performance of answering queries.



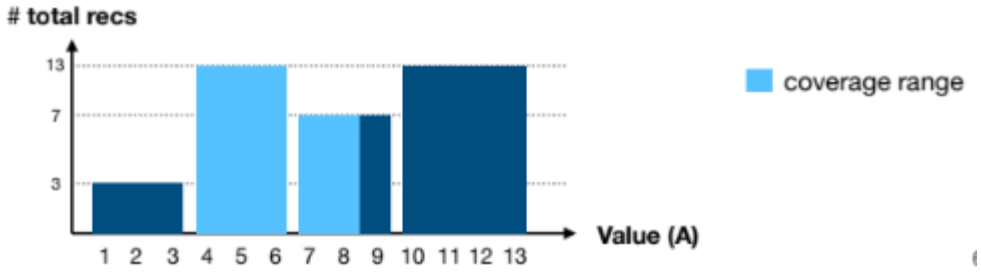
Approximate Query Answering using Histograms

- Paper Link:
  http://www.madgik.di.uoa.gr/sites/default/files/ieee_deb_v22.4.pp5-14.pdf
- Difficulty: **Easy**
- Problem:
  - Instead of spending lots of time to get the exact answer, OLAP application usually prefer to spend less time and get an approximate answer.
- Main Idea:
  - Use pre compute histogram or cube(in multi-dimension) to compute the approximate answer of the queries.

Approximate Query Answering using Histograms

- Challenges:
  - You need to implement the algorithm of histogram and notice that how to decided the width of the histogram to get the better performance.

    - For example: q = *SUM (A) [4:8]*
      - $q = 0 + (5*13) * 1 + (8*7) * \frac{2}{3} + 0$
        $= 102.3$



Randomized Algorithms For Optimizing Large Join Queries

- Paper Link: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.96.9069&rep=rep1&type=pdf
- Difficulty: **Medium**
- Problem:
  - Implement a algorithm to optimize project-select-join query to get a plan with lower access cost
- Main Idea:
  - Used a method called Two Phase Optimization (2PO) which combine Iterative Improvement (II) and Simulated Annealing (SA) to decide the join order of plans.

Randomized Algorithms For Optimizing Large Join Queries

- Challenges:
  - You may have to implement Iterative Improvement (II) and Simulated Annealing (SA) before Two Phase Optimization (2PO) .

Join method choice	$A \bowtie_{method_i} B \rightarrow A \bowtie_{method_j} B$
Join commutativity	$A \bowtie B \rightarrow B \bowtie A$
Join associativity	$(A \bowtie B) \bowtie C \leftrightarrow A \bowtie (B \bowtie C)$
Left join exchange	$(A \bowtie B) \bowtie C \rightarrow (A \bowtie C) \bowtie B$
Right join exchange	$A \bowtie (B \bowtie C) \rightarrow B \bowtie (A \bowtie C)$

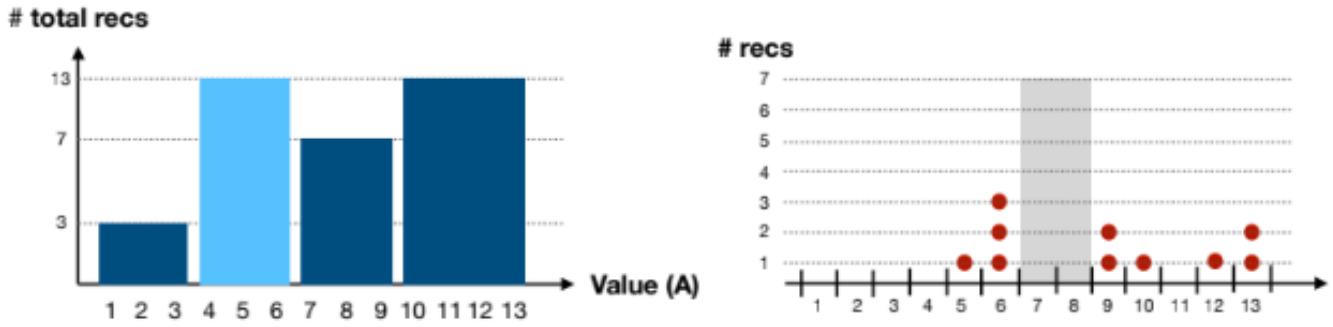
AQP++: Connecting Approximate Query Processing With Aggregate Precomputation for Interactive Analytics

- Paper Link: http://www.sfu.ca/~jinglinp/sigmod2018-aqp++.pdf
- Difficulty: **Medium**
- Problem:
  - Instead of spending lots of time to get the exact answer, OLAP application usually prefer to spend less time and get an approximate answer.
- Main Idea:
  - In the pasts, the database community has proposed two separate ideas, sampling-based approximate query processing (AQP) and aggregate precomputation(AggPre) .In this paper, they connect this two method.

AQP++: Connecting Approximate Query Processing With Aggregate Precomputation for Interactive Analytics

- Challenges:
  - You may have to know the sampling base and histogram base AQP methods.
  - You may have to implement a method that hybrid both of sampling and histogram .

$$
\begin{aligned}
q &= SUM\ (A)\ [4{:}8] \\
&= SUM\ (A)\ [4{:}6] + SUM\ (A)\ [7{:}8] \\
&= 70 + 0 \\
&= 70
\end{aligned}
$$

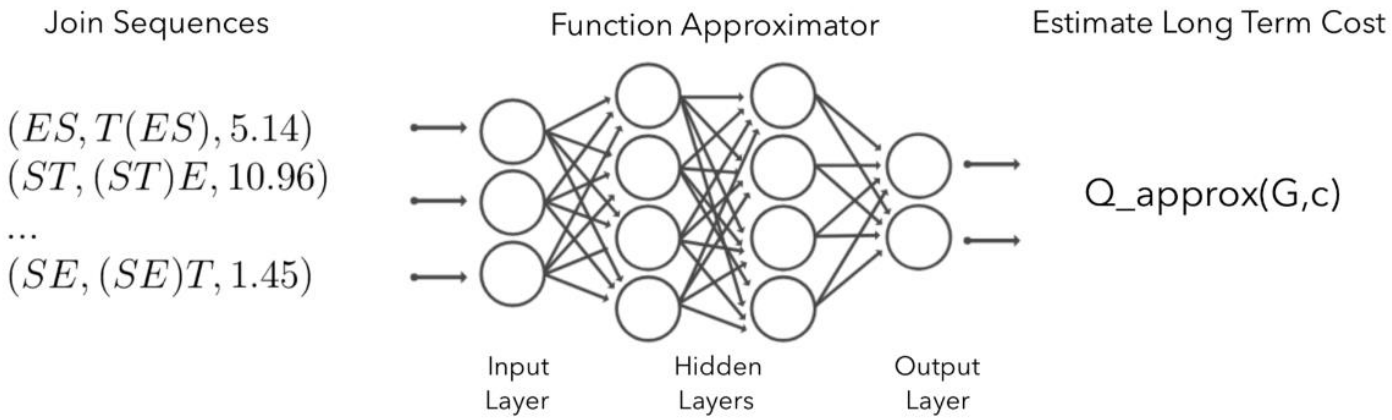


Learning to Optimize Join Queries With Deep Reinforcement Learning

- Paper Link: https://arxiv.org/abs/1808.03196
- Difficulty: **Hard**
- Problem:
  - Implement a reinforcement learning algorithm to optimize join ordering problem with lower access cost.
- Main Idea:
  - This paper formulate the join ordering problem as a Markov Decision Process (MDP), and this paper build an optimizer that uses a Deep Q-Network (DQN) to efficiently order joins.

Learning to Optimize Join Queries With Deep Reinforcement Learning

- Challenges:
  - Implementing Q-learning .
  - States, G: the remaining relations to be joined.
  - Actions, c: a valid join out of the remaining relations.
  - Next states, G': naturally, this is the old "remaining relations" set with two relations removed and their resultant join added.
  - Reward, J: estimated cost of the new join.

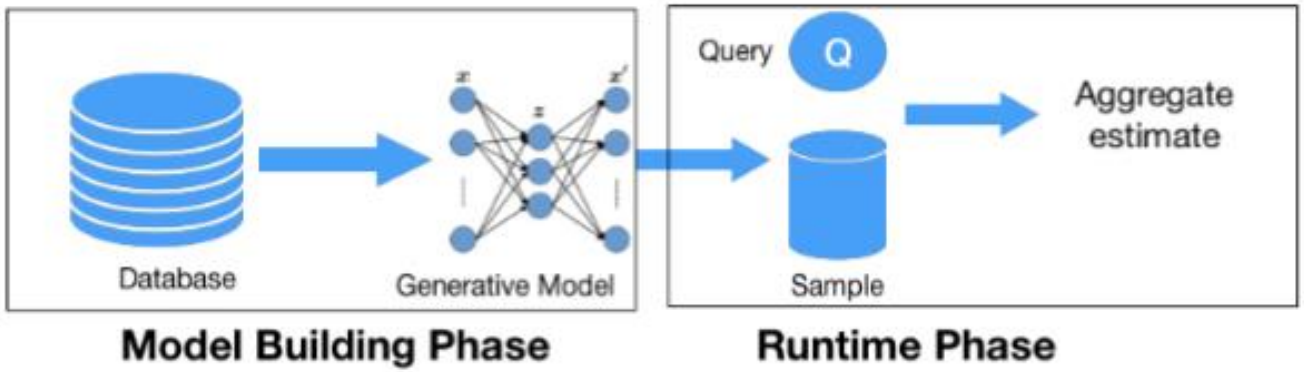


Approximate Query Processing using Deep Generative Models

- Paper Link: https://arxiv.org/abs/1903.10000?fbclid=IwAR2op6n5kiSlhpZ7nLEdCn2mHu86Bv0R-Q3_VUmcnMm2BvJwSlVwccMATSo
- Difficulty: **Hard**
- Problem:
  - To implement a sampling method use variational autoencoders (VAE) to generate samples for Approximate Query Processing (AQP)
- Main Idea:
  - AQP is often achieved by running the query on a pre-computed or on-demand derived sample and generating estimates for the entire dataset based on the result. In this work, we utilizing deep learning (DL) to generate samples from the learned model.

Approximate Query Processing using Deep Generative Models

- Challenges:
  - You may have the knowledge about VAE .
  - You may have to implement a VAE and use the sample that generated by VAE to get the approximate query answer .



Storage Engine

Organized by Yun-Sheng Chang

Paper Lists

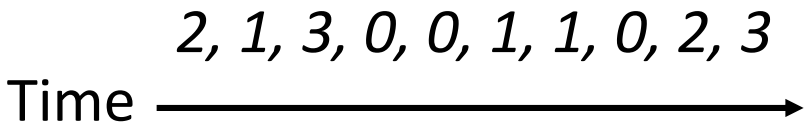
- [SIGMOD'93] The LRU-K page replacement algorithm for database disk buffering
- [FAST'04] CAR: Clock with Adaptive Replacement
- [VLDB'19] Performance-Optimal Filtering: Bloom Overtakes Cuckoo at High Throughput
- [SIGMOD'18] The Case for Learned Index Structures
- [VLDB'05] C-Store: A Column-oriented DBMS
- [VLDB'17] An Empirical Evaluation of In-Memory Multi-Version Concurrency Control

The LRU-K page replacement algorithm for database disk buffering

- Paper Link: http://www.cs.cmu.edu/~christos/courses/721-resources/p297-o_neil.pdf
- Difficulty: **Easy**
- Problem:
  - LRU is unable to differentiate between pages that have relatively frequent reference and pages that have very infrequent reference.
- Main Idea:
  - Evict the page whose K-th most recent access is furthest in the past.
- Components you might have to modify:
  - Buffer management

LRU-1 (a.k.a. LRU) vs. LRU-2

- Assume we have four buffers, and we have the following access history



2, 1, 3, 0, 0, 1, 1, 0, 2, 3

Evict based on LRU-1: 1

2, 1, 3, 0, 0, 1, 1, 0, 2, 3

Evict based on LRU-2: 2

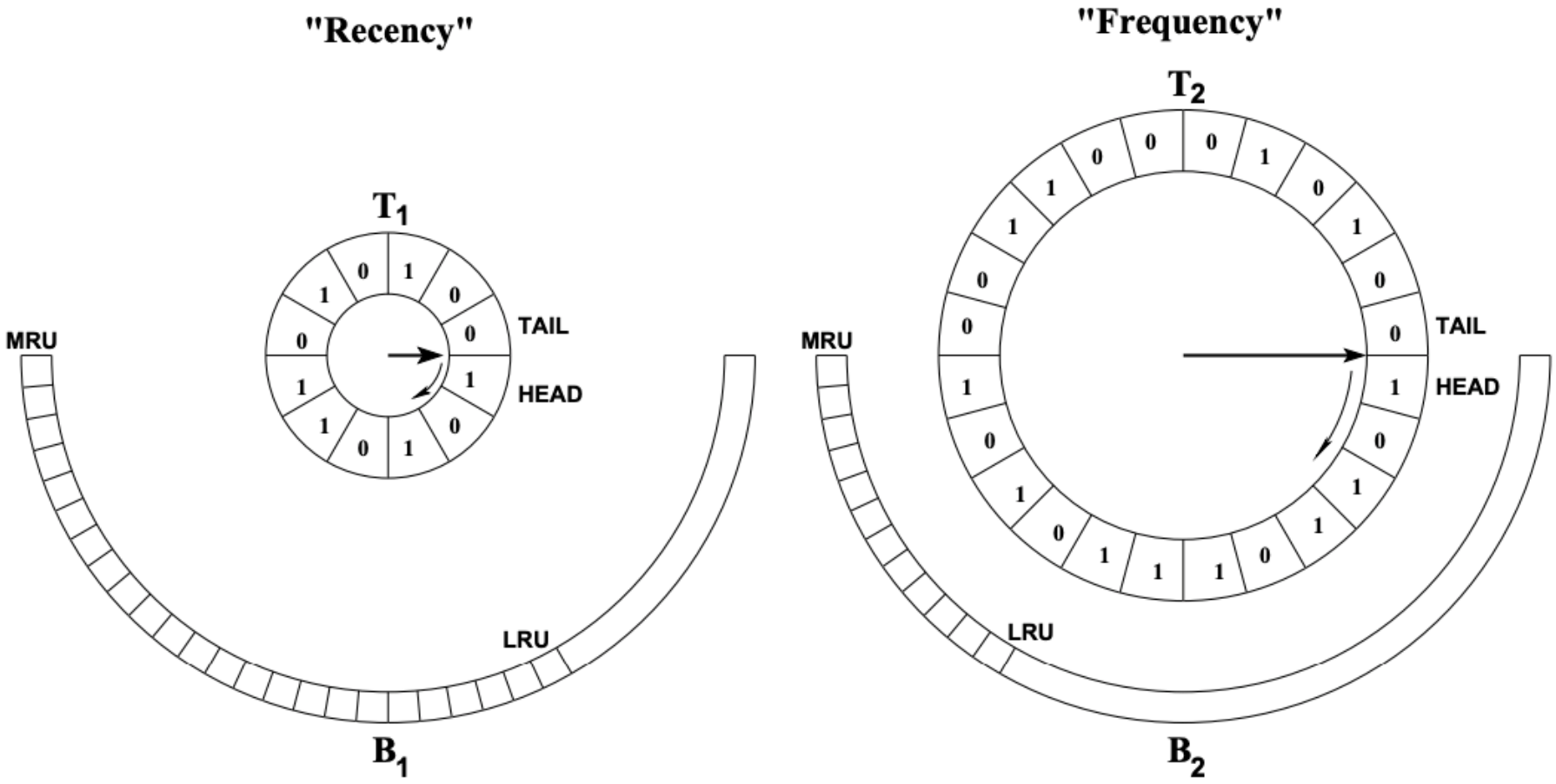
CAR: Clock with Adaptive Replacement

- Paper Link: http://usenix.org/publications/library/proceedings/fast04/tech/full_papers/bansal/bansal.pdf
- Difficulty: **Easy**
- Problem:
  - LRU is unable to differentiate between pages that have relatively frequent reference and pages that have very infrequent reference.
- Main Idea:
  - Maintain a history of recently evicted pages and uses this to change preference to recent or frequent access.
- Components you might have to modify:
  - Buffer management

CAR = ARC + CLOCK

- CLOCK removes the contention nature of LRU
- ARC (Adaptive Replacement Cache) maintains two LRU lists, one for capturing “recency”, and the other for capturing “frequency”
- The *adaptive* part comes from dynamically adjusting the size of the two LRU lists
- CAR (ARC + CLOCK) removes the contention nature of ARC

CAR

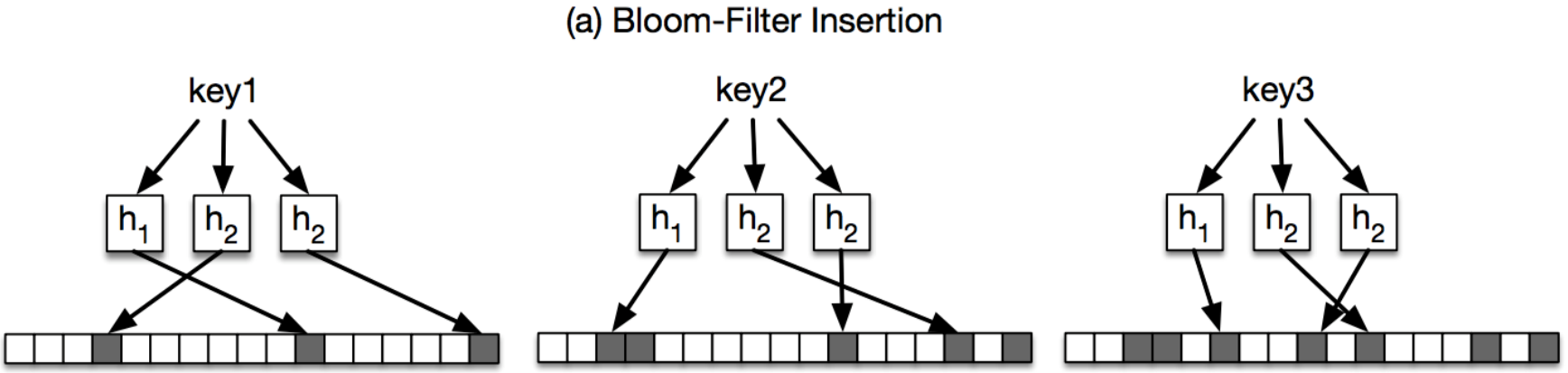


Performance-Optimal Filtering: Bloom Overtakes Cuckoo at High Throughput

- Paper Link: http://www.vldb.org/pvldb/vol12/p502-lang.pdf
- Difficulty: **Medium**
- Main Idea:
  - Comparing the performance of two common *existence indexes*, the *Bloom filter* and the *Cuckoo filter.*
- Existence index:
  - Eliminate *most* of the accesses to non-existing records
  - Correctness: Allow false positive, but forbid false negative
  - Precision: Low false-positive rate
- Components you might have to modify:
  - Index structure

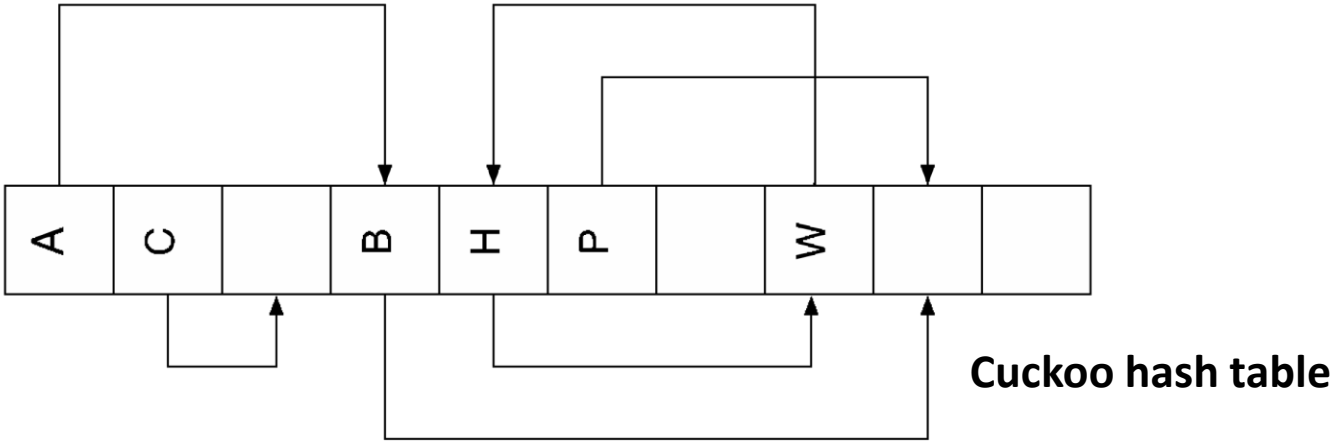
Bloom Filter

- A bit array of size m and k hash functions
- Insertion: a key is fed to the k hash-functions and the bits of the returned positions are set to 1
- Query: If any of the bits at those k positions is 0, the key does not exist



Cuckoo Filter

- Insertion: Store *signature* in a *Cuckoo hash table*
  - A signature approximates a key using fewer bits
  - A Cuckoo hash table resolves collisions by using two hash functions instead of only one
- Query: If the two buckets does not contain the signature of a key, then the key does not exist

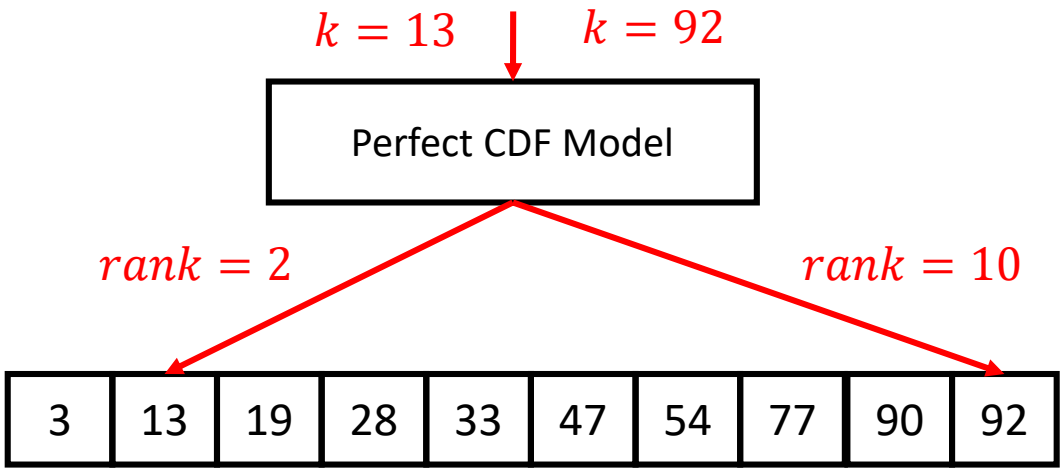


The Case for Learned Index Structures

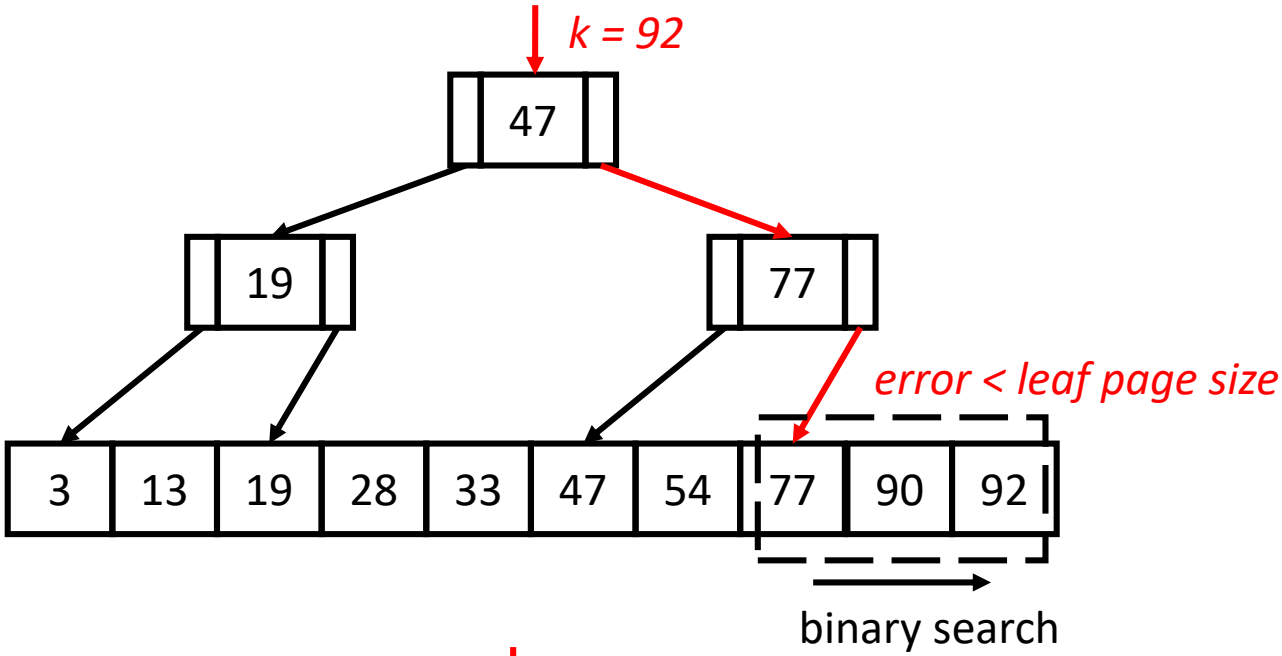
- Paper Link: https://www.cl.cam.ac.uk/~ey204/teaching/ACS/R244_2018_2019/papers/Kraska_SIGMOD_2018.pdf
- Difficulty: **Medium**
- Problem:
  - Traditional indexes (e.g., B-tree and hash table) are general purpose data structures; they assume nothing about the data distribution and do not take advantage of more common patterns prevalent in real world data.
- Main Idea:
  - Machine learning (ML) opens up the opportunity to learn a model that reflects the patterns in the data and thus to enable the automatic synthesis of specialized index structures.
- Components you might have to modify:
  - Index structure

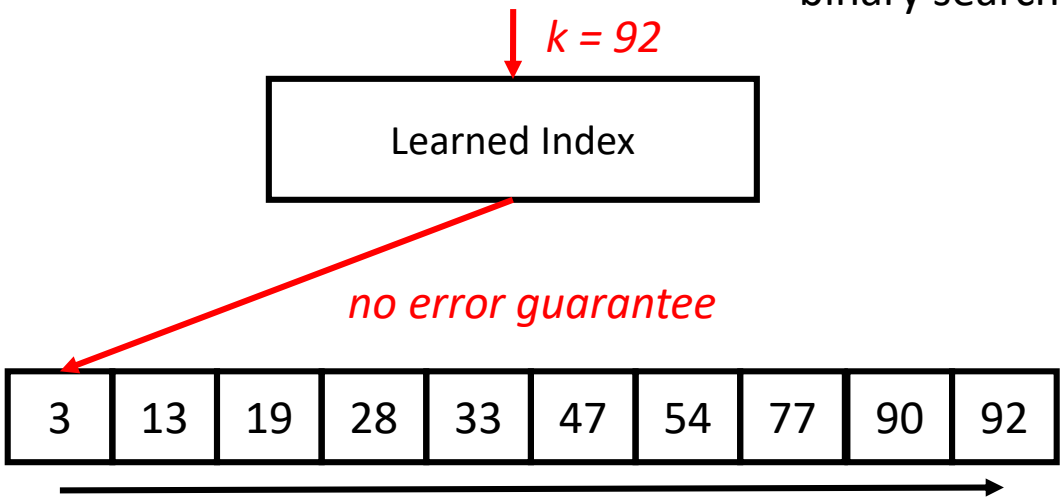
Estimating CDF

- Cumulative distribution function
- Consider a model that perfectly learn the CDF
  - given a key *k*, our model return the *exact rank* of the key among all the keys



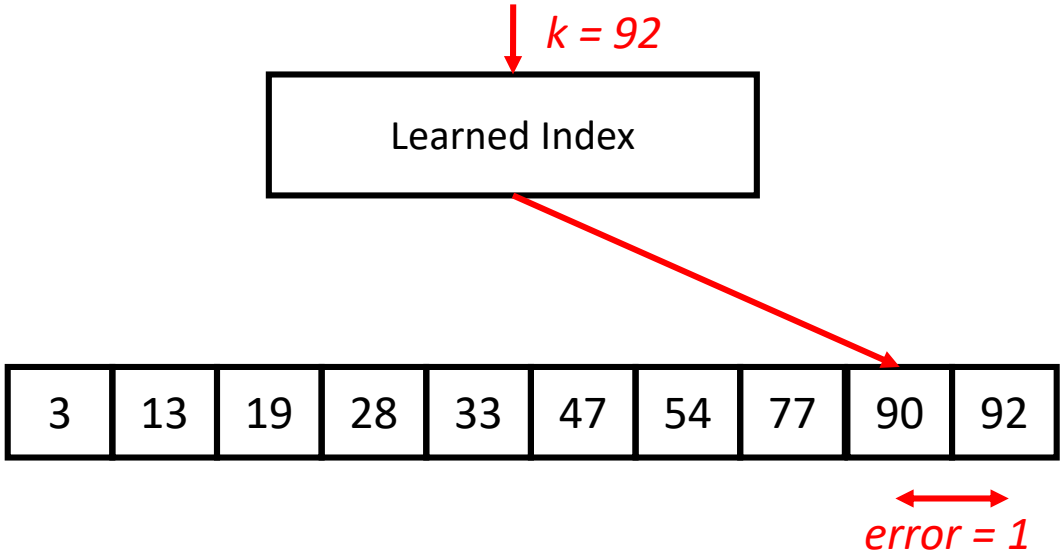
Error Guarantee





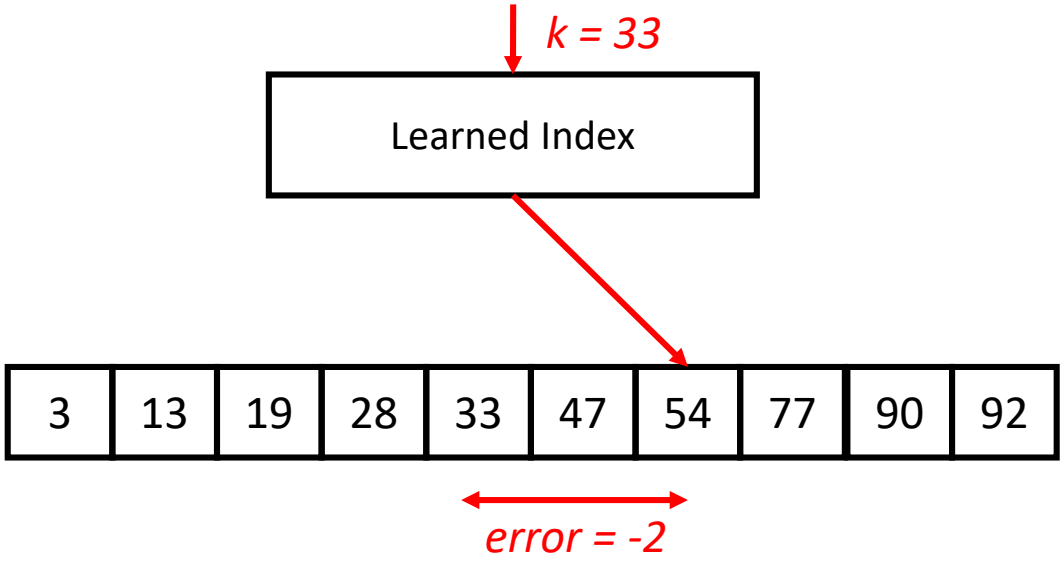
Min Max Errors

- Execute the model for every key and remember the worst over- and under-prediction of a position after the learned index is fixed



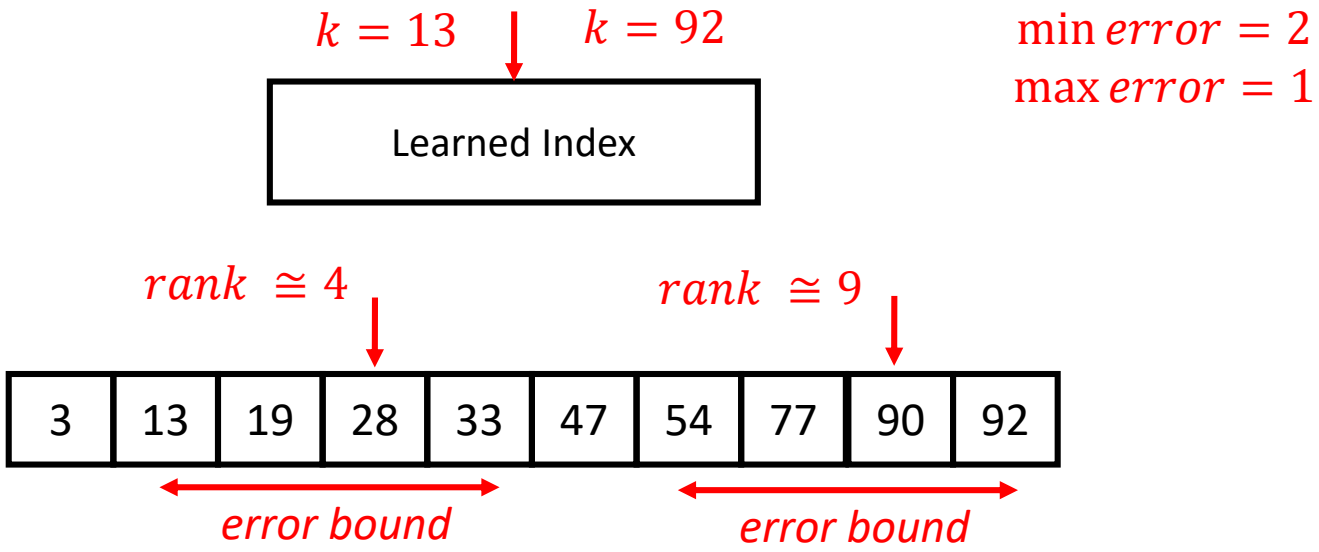
Min Max Errors

- Execute the model for every key and remember the worst over- and under-prediction of a position after the learned index is fixed



Learned Index Structure

- Input
  - a key *k*
- Output
  - the *estimated rank* of *k* among all the keys



C-store: a column-oriented DBMS

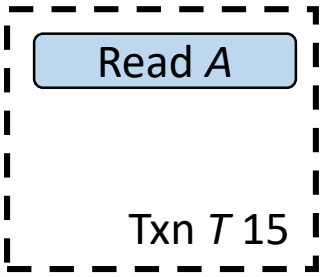
- Paper Link: http://db.csail.mit.edu/projects/cstore/vldb.pdf
- Difficulty: **Hard**
- Problem:
  - Traditional row-oriented scheme is good for write-heavy workload, but may not be suitable for read-heavy workload.
- Main Idea:
  - Propose *column store*, in which the values for each single column (or attribute) are stored contiguously.
- Components you might have to modify:
  - Tuple formats
  - Index structure
  - Recovery

An Empirical Evaluation of In-Memory Multi-Version Concurrency Control

- Paper Link: http://www.vldb.org/pvldb/vol10/p781-Wu.pdf
- Difficulty: **Hard**
- Main Idea:
  - This paper evaluates multiple variants of multi-version concurrency control (MVCC).
  - Time stamp ordering should be the simplest to implement.
- Components you might have to modify:
  - Tuple formats
  - Concurrency control
  - Recovery

Timestamp Ordering (MVTO)

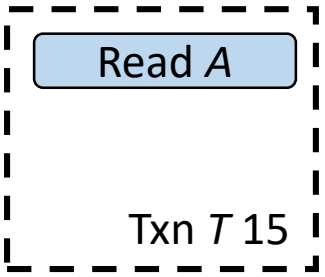
- T_{id} represents the serialization order of txns
- *read-ts*: The last transaction that reads the tuple version
- Read
  - Find a version satisfying: *begin-ts* <= T_{id} < *end-ts*
  - Wait until the version is not write-locked by another txn
  - Modify *read-ts* if it is less than T_{id}



	txn-id	begin-ts	end-ts	read-ts
Ax	0	4	10	0
Ax+1	0	10	20	14

Timestamp Ordering (MVTO)

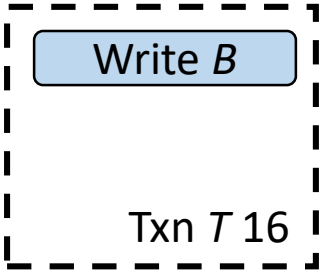
- T_{id} represents the serialization order of txns
- *read-ts*: The last transaction that reads the tuple version
- Read
  - Find a version satisfying: *begin-ts* <= T_{id} < *end-ts*
  - Wait until the version is not write-locked by another txn
  - Modify *read-ts* if it is less than T_{id}



	txn-id	begin-ts	end-ts	read-ts
Ax	0	4	10	0
Ax+1	0	10	20	15

Timestamp Ordering (MVTO)

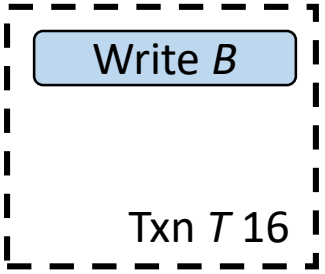
- Write
  - Find the latest version of the tuple
  - Check if the following two conditions are satisfied; if not, the DBMS aborts *T*
    - T_{id} > *read-ts*
    - the version is not write-locked by another txn
  - Acquire the write lock of the version
  - Create a new version (and lock the version)



	txn-id	begin-ts	end-ts	read-ts
Bx	0	4	INF	12

Timestamp Ordering (MVTO)

- Write
  - Find the latest version of the tuple
  - Check if the following two conditions are satisfied; if not, the DBMS aborts *T*
    - T_{id} > *read-ts*
    - the version is not write-locked by another txn
  - Acquire the write lock of the version
  - Create a new version (and lock the version)



	txn-id	begin-ts	end-ts	read-ts
Bx	16	4	INF	12

Timestamp Ordering (MVTO)

- Write
  - Find the latest version of the tuple
  - Check if the following two conditions are satisfied; if not, the DBMS aborts *T*
    - T_{id} > *read-ts*
    - the version is not write-locked by another txn
  - Acquire the write lock of the version
  - Create a new version (and lock the version)

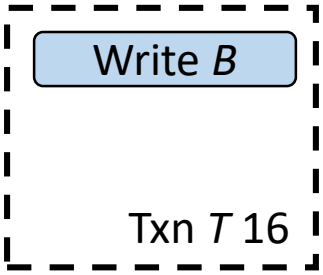
Write *B*

Txn *T* 16

	txn-id	begin-ts	end-ts	read-ts
Bx	16	4	INF	12
Bx+1	16	-	-	0

Timestamp Ordering (MVTO)

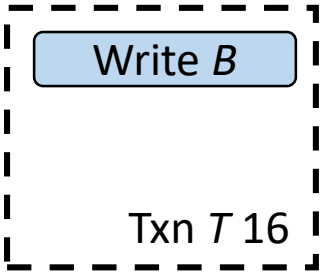
- Commit
  - Modify the lifetime of the old and new versions
    - Old version: *end-ts* -> T_{id}
    - New version: (*begin-ts*, *end-ts*) -> (T_{id}, INF)
  - Release write locks



	txn-id	begin-ts	end-ts	read-ts
Bx	16	4	INF	12
Bx+1	16	-	-	0

Timestamp Ordering (MVTO)

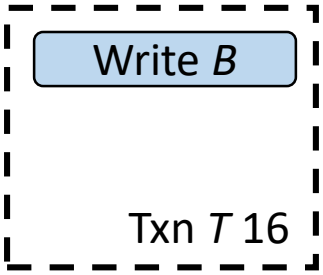
- Commit
  - Modify the lifetime of the old and new versions
    - Old version: *end-ts* -> T_{id}
    - New version: (*begin-ts*, *end-ts*) -> (T_{id}, INF)
  - Release write locks



	txn-id	begin-ts	end-ts	read-ts
Bx	16	4	16	12
Bx+1	16	16	INF	0

Timestamp Ordering (MVTO)

- Commit
  - Modify the lifetime of the old and new versions
    - Old version: *end-ts* -> T_{id}
    - New version: (*begin-ts*, *end-ts*) -> (T_{id}, INF)
  - Release write locks



	txn-id	begin-ts	end-ts	read-ts
Bx	0	4	16	12
Bx+1	0	16	INF	0

Transaction Management

Organized by Yu-Shan Lin

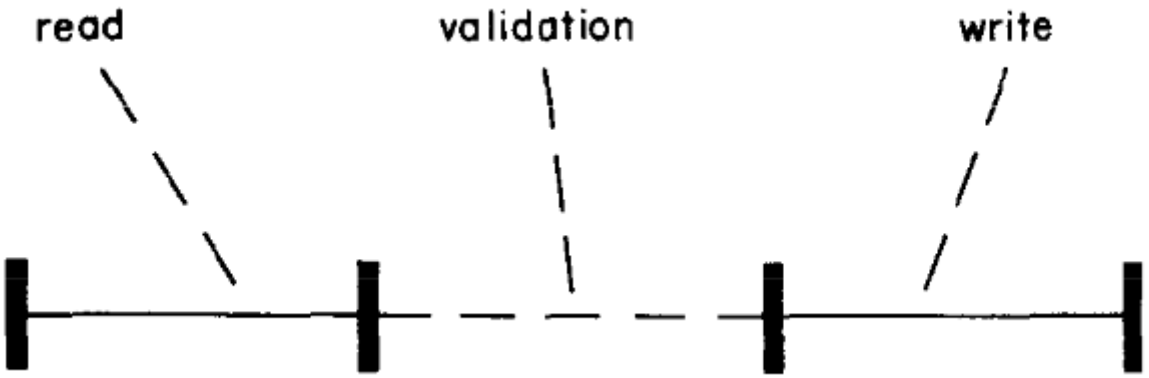
Paper Lists

- [TODS'81] On Optimistic Methods for Concurrency Control
- [VLDB’18] Contention-Aware Lock Scheduling for Transactional Databases
- [VLDB’18] Improving Optimistic Concurrency Control Through Transaction Batching and Operation Reordering
- [SIGMOD’16] Low-Overhead Asynchronous Checkpointing in Main-Memory Database Systems
- [SIGMOD’15] Fast Serializable Multi-Version Concurrency Control for Main-Memory Database Systems
- [ATC’16] Toward Coordination-free and Reconfigurable Mixed Concurrency Control

On Optimistic Methods for Concurrency Control

- Paper Link: https://www.eecs.harvard.edu/~htk/publication/1981-tods-kung-robinson.pdf
- Difficulty: **Easy**
- Problem:
  - Lock-based protocols always lock a object even if no concurrent transaction accesses the same object. This creates unnecessary cost in low-contention workloads.
- Main Idea:
  - They purposes an “optimistic” concurrency control protocol that does not lock any objects and only checks for conflict when the transaction is about to commit.

On Optimistic Methods for Concurrency Control

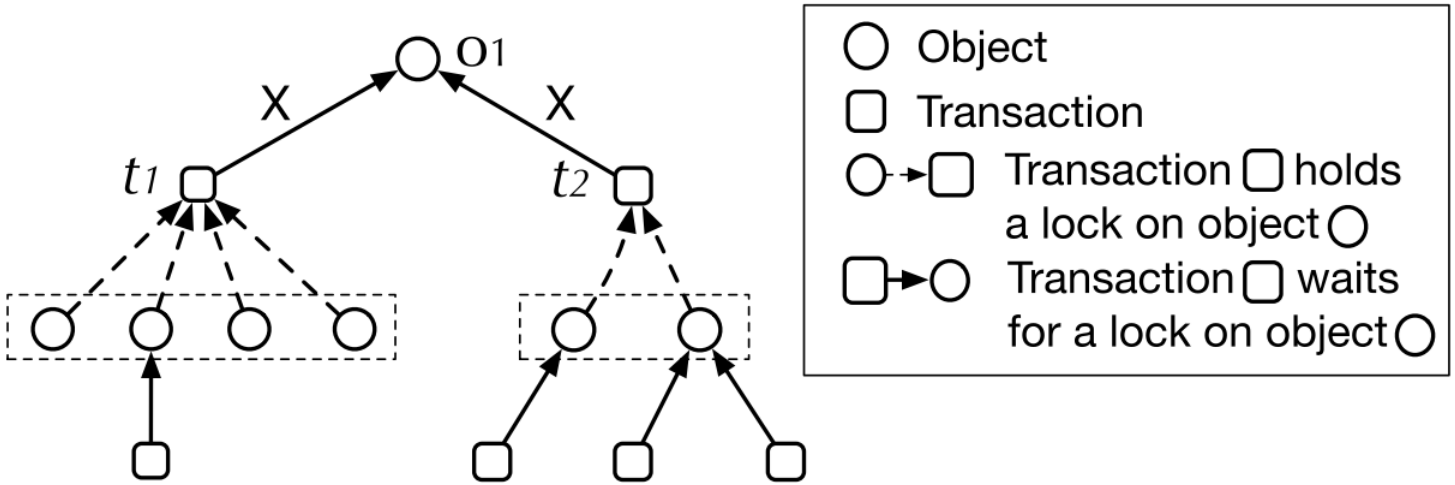


- Challenges:
  - It may be not easy to show an improvement of performance on VanillaDB.

Contention-Aware Lock Scheduling for Transactional Databases

- Paper Link: http://www.vldb.org/pvldb/vol11/p648-tian.pdf
- Difficulty: **Easy**
- Problem:
  - When there are multiple transactions requesting an object, which one should be granted first? This paper tries to find an algorithm to grant locks such that the average latency is minimized.
- Main Idea:
  - Granting the lock to the transaction likely blocking the most transactions.

Contention-Aware Lock Scheduling for Transactional Databases

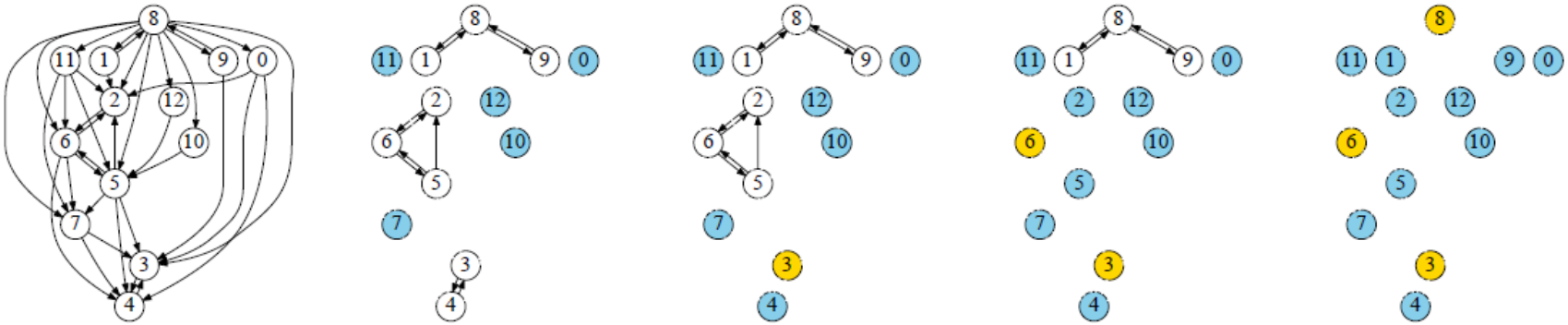


- Challenges:
  - We expect to see that the teams choosing this paper implement all the algorithms in the paper.

Improving Optimistic Concurrency Control Through Transaction Batching and Operation Reordering

- Paper Link: http://www.vldb.org/pvldb/vol12/p169-ding.pdf
- Difficulty: **Medium**
- Problem:
  - An Optimistic Concurrency Controls (OCC) is fast, but it aborts transactions more frequently than pessimistic methods. This paper tries to deduce unnecessary aborts.
- Main Idea:
  - It batches lock requests so that it can analyze dependencies between them and efficiently avoid unnecessary aborts.

Improving Optimistic Concurrency Control Through Transaction Batching and Operation Reordering



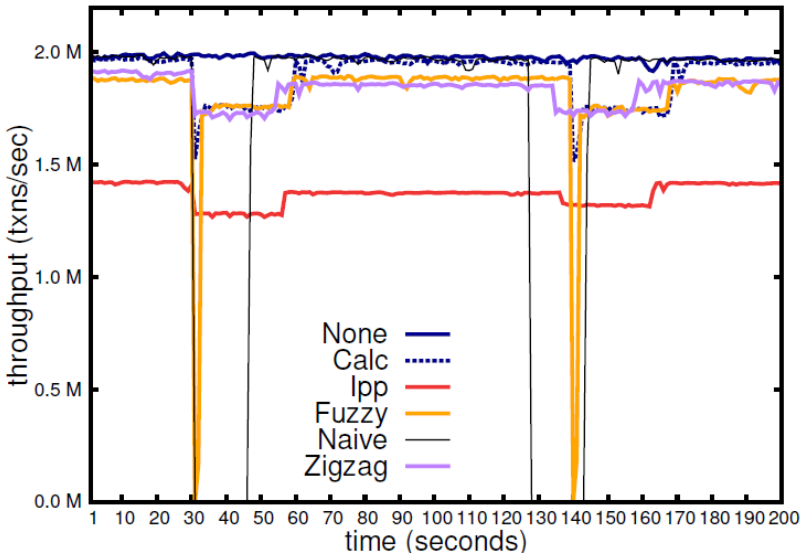
- Challenges:
  - You must implement an OCC first.

Low-Overhead Asynchronous Checkpointing in Main-Memory Database Systems

- Paper Link: http://cs-www.cs.yale.edu/homes/dna/papers/fast-checkpoint-sigmod16.pdf
- Difficulty: **Medium**
- Problem:
  - Checkpointing always makes a great impact to system performance.
- Main Idea:
  - This paper designs a asynchronous checkpointing algorithm that minimizes the impact.

Low-Overhead Asynchronous Checkpointing in Main-Memory Database Systems

- Challenges:
  - The algorithm is designed for main-memory DBMSs. You may have to change it slightly so that it can fit in VanillaDB.

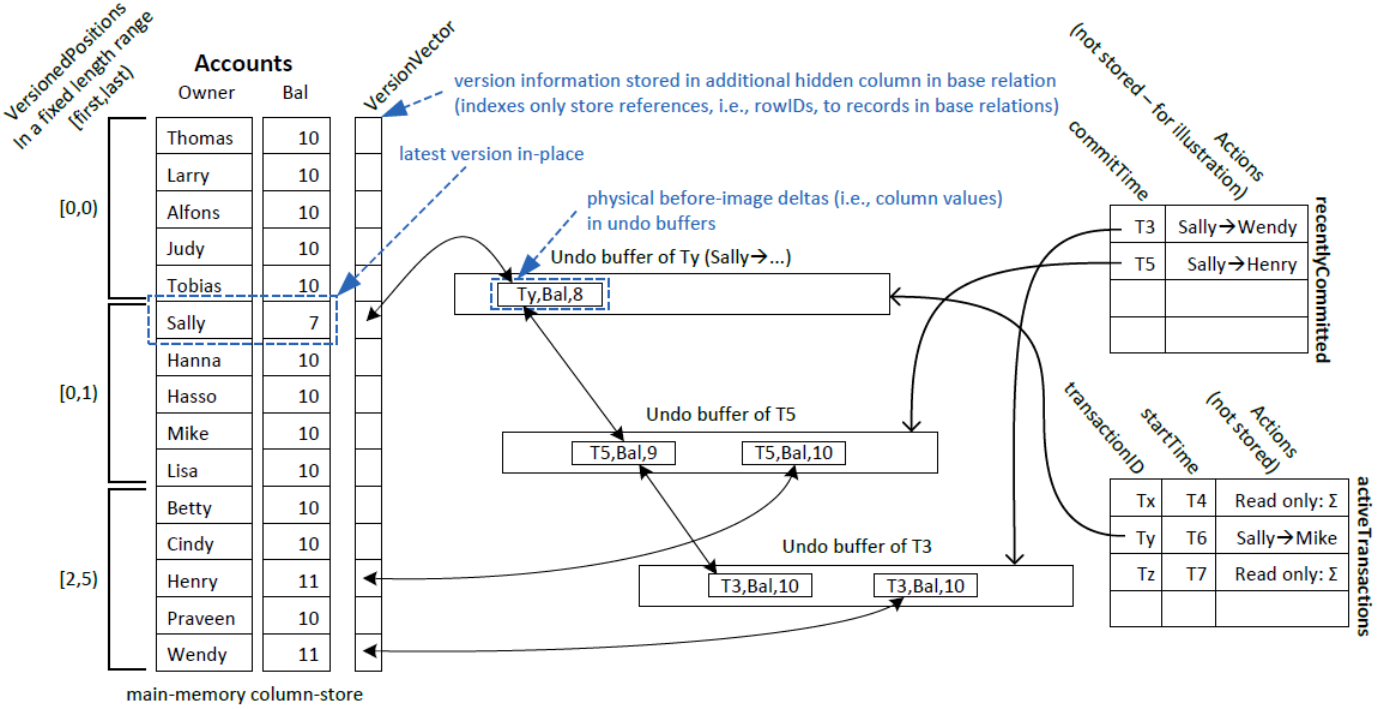


Fast Serializable Multi-Version Concurrency Control for Main-Memory Database Systems

- Paper Link: https://db.in.tum.de/~muehlbau/papers/mvcc.pdf
- Difficulty: **Hard**
- Problem:
  - To implement a fast validation for SERIALIZABLE isolation in a multi-versioning DBMS.
- Main Idea:
  - It maintains an undo buffer for each record and saves the old versions in the buffers. They then use **predicates** to search for conflicts, which is much faster than other algorithm.

Fast Serializable Multi-Version Concurrency Control for Main-Memory Database Systems

- Challenges:
  - Implementing a MVCC along with a OCC is not a easy work.
  - You may have to implement a traditional validation for comparison.



Toward Coordination-free and Reconfigurable Mixed Concurrency Control

- Paper Link: https://www.usenix.org/conference/atc18/presentation/tang
- Difficulty: **Hard**
- Problem:
  - Each concurrency control scheme has its own suitable scenario, but there is no a scheme fitting all workloads.
- Main Idea:
  - This paper proposes a protocol to fix a few major CC schemes together.

Toward Coordination-free and Reconfigurable Mixed Concurrency Control

- Challenges:
  - You must implement at least 2 more concurrency control protocols.
  - You may have to redesign the benchmarks to demonstrate the effectiveness of this work.