Temporal Indexing

Snapshot Index

Transaction Time Environment

- Assume that when an event occurs in the real world it is inserted in the DB
- A timestamp is associated with each operation
- Transaction timestamps are *monotonically increasing*
- Previous transactions cannot be changed → we cannot change the past
Example

- Time evolving set of objects: employees of a company
- Time is discrete and described by a succession of non-negative integers: 1, 2, 3, ...
- Each time instant changes may happen, i.e., addition, deletion or modification
- We assume only insertion & deletion: modifications can be represented by a deletion and an insertion

Records

- Each object is associated with:
  1. An oid (key, time invariant, eid)
  2. Attributes that can change (salary)
  3. A lifespan interval \([t.\text{start}, t.\text{end})\)
- An object is alive from the time it inserted in the set, until it was deleted
- At insertion time deletion is unknown
- Deletions are logical: we change the now variable to the current time,
  \([t1, \text{now}) \rightarrow [t1, t2)\)
Evolving set

- The state $S(t)$ of the evolving set at $t$ is defined as: “the collection of alive objects at $t$”
- The number of changes $n$ represents a good metric for the minimal space needed to store the evolution

Evolving sets

- A new change updates the current state $S(t)$ to create a new state

```
\begin{array}{c}
t_1 \\
a \\
\end{array} \quad \begin{array}{c}
t_2 \\
a,h \\
\end{array} \quad \begin{array}{c}
t_i \\
S(t_i) \\
\end{array} \quad \begin{array}{c}
time \\
a,f,g \\
\end{array}
```
Transaction-time Queries

- Pure-timeslice
- Range-timeslice
- Pure-exact match

Snapshot Index

- Snapshot Index, is a method that answers efficiently pure-timeslice queries
- Based on a main memory method that solves the problem in $O(a + \log_2 n)$, $O(n)$ space
- External memory: $O(a/B + \log_B n)$
MM solution

- Copy approach: $O(a + \log n)$ but $O(n^2)$ space
- Log approach: $O(n)$ space but $O(n)$ query time
- We should combine the fast query time with the small space (and update)

Assumptions

- Assumptions (for clarity)
  - At each time instant there exist exactly one change
  - Each object is named by its creation time
Access Forest

- A double linked list L. Each new object is appended at the right end of L
- A deleted object is removed from L and becomes the next child of its left sibling in L
- Each object stores a pointer to its parent object. Also a parent object points to its first and last children

AF example
Additional structures

- A hashing scheme that keeps pointers to the positions of the alive elements in L
- An array $A$ that stores the time changes. For each time change instant, it keeps the pointer to the current last object in $L$

Properties of AF

- In each tree of the AF the start times are sorted in preorder fashion
- The lifetime of an object includes the lifetimes of its descendants
- The intervals of two consecutive children under the same parent may have the following orderings:
  
  $s_i < e_i < s_{i+1} < e_{i+1}$
  
  or
  
  $s_i < s_{i+1} < e_i < e_{i+1}$
Searching

- Find all objects alive at $t_q$
- Use A to find the starting object in the access forest $L$ (O(logn))
- Traverse the access forest and report all alive objects at $t_q$ O(a) using the properties

Disk based Solution

- Keep changes in pages as it was a Log
- Use hashing scheme to find objects by name (update O(1))
- Acceptor: the current page that receives objects
Definitions

- A page is useful for the following time instants:
  - I-useful: while this page was the acceptor block
  - II-useful: for all time instants for which it contains at least uB “alive” records
- u is the usefulness parameter

Meta-evolution

- From the actual evolution of objects, now we have the evolution of pages! *meta-evolution*
- The “lifetime” of a page is its usefulness

List $L$:

- $(SP, [0, now])$ → $(D, [29, now])$ → $(H, [70, now])$
- $(A, [1, 52])$ → $(C, [46, 80])$
- $(E, [15, 51])$ → $(F, [60, 64])$ → $(B, [64, 70])$
Searching

- Find all alive objects at $t_q \rightarrow$ Find all useful pages at $t_q$
- The search can be done in $O(a/B + \log_8 n)$

Copying procedure

- To maintain the answer in few pages we need clustering: controlled copying
- If a page has less than $uB$ alive objects, we artificially delete the remaining alive objects and copy them to the acceptor bock
Optimal Solution

- We can prove that the SI is optimal for pure-timeslice queries:
  - $O(n)$ space, $O(a/B + \log_B n)$ query and $O(1)$ update (expected, using hashing)

Range-timeslice queries

- Multi-version B-tree: next!