The Snapshot Index: An I/O Optimal Access Method For Timeslice Queries

Vassilis J. Tsotras and Nickolas Kangelaris

Abhishek Srivastava
Student ID: 861307778

March 9, 2017
CS 236, Winter 2017

The problem:
In this paper the author talks about the shortcomings of storing the evolutions of databases in a conventional way and present a model to store them in efficient manner with respect to their computational cost and space consumed.

The contribution:
The authors present *Snapshot Index* for a transaction time environment which uses a database state and its evolution i.e operations applies to the database, to provide an efficient way to apply insert, update and search operation using timeslices as a parameter in the database.

The method:
The objects in temporal databases usually store three parameters to represent them. An *Oid*(time invariant key), time variant attributes and Lifespan interval [start,end). At insertion we do not know the end time so when we delete it's end value can be update. Object is considered "alive" until is deleted and this property is useful in snapshot indexing since we are trying to optimize pure-timeslices queries.

The authors discusses many approaches in the paper with pros and cons of each process. The two most common approaches he present is storing the current state of the database for every time there is an evolution or storing the updates to the databases into a "log" and use it generate database current state.

The databases presented time(t), object id(oid) and operation(op) in the blocks of specific sizes. It is stored sequentially in the acceptor block. For updating existing records in a constant time it uses hashing schemes. While inserting a data its hash is added to the lookup table with its pointer pointing the record, while deleting records are updated and its hashing is removed. Thus hash lookup table always represent the current state of database and solves the problem of reconstructing state a particular time.

A block "usefulness" (**I-useful** mode) and (**II-useful** mode) is calculated based on few criteria related to their start and end time. If a block does not belong to any of the useful mode it is considered to be *non-useful*. This helps in reducing the problem of reconstructing a current state at a particular time.

A *meta-evolution* parameter is also used to further improve the performance which stores meta-changes of a record and is used in Access Forest method. Access forest contains a doubly link list and an array. At a particular time the list contains blocks with "alive" meta-evolution. Based on "lifespan" blocks are represented as a nodes which facilitates finding "alive" blocks for given query.

Comments:
The paper presents one of the way to store the temporal database in efficient manner and process the pure-timeslice queries in an efficient manner. Space, Update processing and I/O complexity cost are few of the parameters used in its evaluation.

However, there are some drawbacks of the proposed model:

- It is restricted to pure-timeslice and not evaluation was done for Range-timeslice and Pure-exact match queries.
- It can be expanded to bi-temporal databases with its access methods.