Add/Drop Index (Concurrently) in Peloton

Rong Huang  rhuang@andrew.cmu.edu
Xingyu Jin    xingyuj1@andrew.cmu.edu
Ziheng Liao  zihengl@andrew.cmu.edu
Recap: what have we done?

- Motivation
- Goal
  - 75%: Lock everything
  - 100%: Concurrency pain
  - 125%: Multithreading? For real?

Testing: from unit to SQL

Code quality: of course we are good

Future: more index, more fun
It’s Broken.
75%: Support add/drop index correctly in Peloton, using locking mechanisms.

For drop index: Implementations already exist, yet buggy in some ways. We fixed those bugs.

For add index: Implemented lock-based version, based on our centralized lock-manager.
LOCK MANAGER

Centralized: get a static copy of it just like you get the instance of catalog

Per-table lock: lock the tables based on their oid

Two modes: you can use it as a scope lock, which will unlocks when the scope ends. You can also lock the table until current transaction ends.

Currently supports only read_write lock, but can be extended.
75%: LOCK EVERYTHING

TXN 1
BEGIN
...
...
INSERT INTO A
...
COMMIT

TXN 2
BEGIN
...
...
CREATE INDEX
...
COMMIT

TXN 3
BEGIN
...
...
INSERT INTO A
...
COMMIT

Lock Manager (table A’s slot)

Shared

Exclusive

Shared
100%: Support add index **correctly** in Peloton, by doing it **concurrently**

We managed to come up with a solution that deals with possible race conditions. No lock is used in our implementation. Also, we added support for parsing the query “CREATE INDEX CONCURRENTLY”.

Notice that concurrent create index will not block other txns, but itself still blocks (same behavior as Postgres).
100%: HANDLE CONCURRENCE

TXN 1
BEGIN
... INSERT INTO A ... COMMIT

TXN 2
BEGIN
... CREATE INDEX ... COMMIT

TXN 3
BEGIN
INSERT INTO A ... COMMIT
100%: HANDLE CONCURRENCY

BEGIN
...  
**INSERT INTO A**
...
COMMIT

BEGIN
...
**CREATE INDEX**
...
COMMIT

BEGIN
**INSERT INTO A**
COMMIT

**Concurrent txns list**

**TXN 1**

**TXN 2**

**TXN 3**

**Index: insertion record**
GOAL

125%: Support add index **correctly** in Peloton, by doing it **concurrently**, and **in parallel**

Parallel add index depends largely on multi-threaded sequential scan, and Prashanth is still working on it.
Unit tests: tests for lock manager, tests for add/drop index

SQL tests with junit: launch parallel transactions, testing concurrent operations with or without enforced ordering of events.
Utilizing object lifecycle to unlock scoped locks automatically
Utilizing shared ptr to avoid other transactions holding deleted locks
Utilizing concurrent data structure (tbb’s concurrent_unordered_set) to avoid race conditions
Strength: put new functionalities in proper places; design with minimum impact to existing code
Weakness: didn’t comment code as we develop, but rather add them later on
PELOTON IS BUGGY!

Some known bugs that has been fixed by our team:

- DropIndex won’t actually drop index in table_catalog (function call typo)
- PopulateIndex will add entries to already existed indexes
- PopulateIndex will cause transaction abort if the index is empty
PELTON IS BUGGY!

Some still existing bugs:

• Insert/Delete/Update actions in transaction -> goes through codegen modules, like codegen::Inserter, codegen::Deleter, etc.

• Insert/Delete/Update action not in transaction -> goes through executors (marked as deprecated)

• Weird behavior that Peloton kept switching between executor and their corresponding codegen version. When it uses executor, bugs caused segmentation fault when accessing bw_tree index in certain ways. This bug originated from cmu-db/peloton, and as we were told that those executors are going to be deprecated, we didn’t take effort in fixing them.
FUTURE WORK

Populate index is currently an executor and marked as deprecated. Should add it as one of the codegen modules.

Support create index in parallel, after parallel sequential scan is sorted out.

More on indexing: add indexes to tables, just like Cicada.

Fix related bugs that has not been resolved.
Q&A

Thank you!