CS631 Project Report

Extension to LSM Tree Index in PostgreSQL

Abhinav Garg
22m0750
Sunanda Somwase
22m2107
Tanvi Keluskar
22m0782
Shivansh Dhiman
22m2120

Under the guidance of
Prof. S. Sudarshan

Department of Computer Science and Engineering
Indian Institute of Technology, Bombay
Nov 27, 2022

Objective

Our Aim is to optimize the LSM index extension using Stepped Merged LSM in Postgresql[1]. The later consists of multiple tree structure at each level except first and last level leading to write optimise data-structures.

Key Idea

- Changing existing lsm extension to support Stepped Merged version of LSM
- Implement 3 level index structure of L0, L1 and L2 where L1 is array of 2 index tree.

System Architecture

Figure 1 shows the architecture of stepped merged LSM index implementation followed by us. It consists of Main L0 index block which consists of no. of inserts in L0,L2 and L11 in array. The address array of L1 index tress. The reference architecture of LSM is followed from [2] implementation.

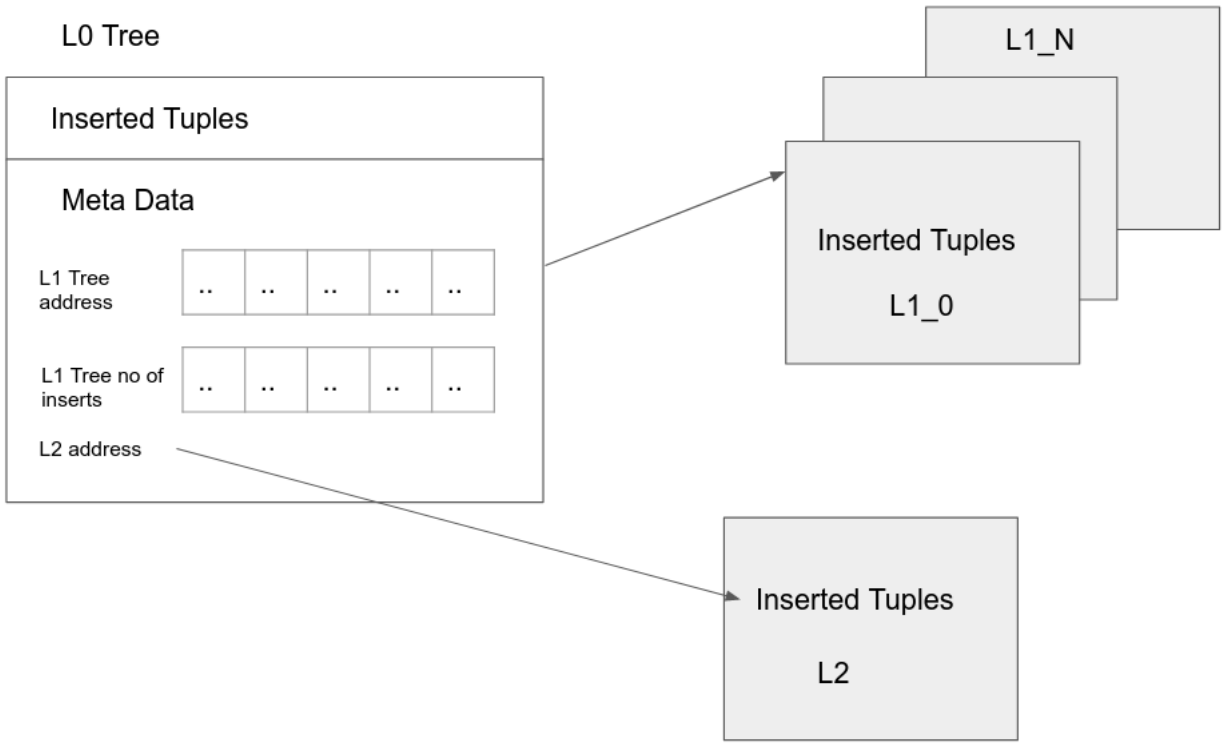

Figure 1: Stepped merged LSM Implementation

Steps to Install the LSM Extension

- Set the following terminal variables:
  - export PATH=/usr/local/pgsql/bin:$PATH
  - export PG_CONFIG=POSTGRES_INSTAlLDIR/bin/pg_config
- Copy the lsm folder in contrib folder of postgres
- To use the PGXS infrastructure for your extension run the following commands which will copy the lsm.control and lsm-x.x.sql in /git/data/share/postgresql/extension folder
  - make USE_PGXS=1
  - make install USE_PGXS=1
- Start the Postgres Server using command :
  - pg_ctl -D $PGDATA -l logfile start
- Enter into the database and create extension using command :
  - create extension lsm;

Implementation Details

We have implemented Stepped Merged LSM tree by creating LSM index. LSM tree consists of three levels.

- Level L0: The size of L0 is configurable, for testing purpose, we have set it to 2. On ambuild routine call, **lsmbuild** function will be called. This function will create lsm_tree index.
- Level L1: The L1 level of step merge LSM tree, has n number of index trees of equal size. Each tree at L1 level is k time larger than tree at L0. This k is also a configurable parameter. That means we can have n number of index trees at L1 level with the size k time L0. For creating L1[i] if not exist is created using function **lsm_create_l1_tree_if_not_exits**
  When L0 gets full, we flush the data from L0 to L1[0]. Now when L1[0] is also full and L0 gets full again, we flush L0 into L1[1] and so on. This is implemented inside **lsm_insert and lsm_merge_indexes** function.

- Level L2: Now when all n trees from L1 level gets full, and L0 is full, all trees from L1 level are flushed into L2, and L0 is flushed to L1[0]. Creation of L2 if not exists is implemented in **lsm_create_l2_if_not_exits**.

We merge indexes, using the merge indexes function. We need to truncate the index which was being merged. That is done use **lsm_truncate_index** for truncating L0 and **lsm_truncate_not_l0_index** is used to truncate any other tree which is not L0 tree. Need of two different functions for truncation is, the process when we truncate L0 is different than the process of truncating any other trees. As when we truncate L0,LSM meta is reinitialised and for the other case we do not reinitialise it.

Files Modified

- Changed lsm.c and lsm.h files to add functionalities related to stepped merged lsm indices.
- Files which get modified after make of the extension:

  POSTGRES_INSTDIR/share/extension/lsm.control

  POSTGRES_INSTDIR/share/extension/lsm–1.0.sql

  POSTGRES_INSTDIR/lib/lsm.so

Run Through

We have 2 level LSM tree with level-0 L0 containing 1 index tree. Level-1 containing n=2 index trees and Level-2 containing 1 index tree. The size of L1 is twice that of L0.

Steps for creating LSM index and table

- Create LSM extension using command :
  create extension lsm;
- Create a table test:
  create table test(k bigint, val bigint);
- Create a lsm based index :
  create index lsm_index on test using lsm(k);

Steps for inserting values into table

Insert following entries into the table.For example purpose consider L0 size as 2 tuples and L1 as 4 tuples, L2 as infinite and no. of levels in L1 as 2 i.e we have 2 L1 trees L1_0 and L1_1

- insert into test values (1,10);
  This will add corresponding index entry in L0.

- insert into test values (2,10);
  This will add corresponding index entry in L0. And the size of L0 becomes 2 (full capacity).Since the size of L0 is full,this will first make a copy of L0, create a data block for L1_0 index tree, add corresponding address for L1_0 entry in L0 meta data. Then copy all tuples from L0 to L1_0 and truncate L0 original page block and replace it with its copy.L0 will be empty at this point.And L1_0 has these entries of L0

- insert into test values (3,10);
  Add this new entry in L0 data block.

- insert into test values (4,10);
  this will add corresponding index entry in L0. Now L0 has 2 values (current and previous insert).This will again initiate same process and the entries will get appended in L1_0 data block and so it has 4 tuples in it reaching its maximum capacity.

- insert into test values (5,10);
  Adds this in L0

- insert into test values (6,10);
  insert into test values (7,10);
  This will add corresponding index entry in L0 add initiate copying data from L0 to L1_1 following same procedure.

- insert into test values (8,10);
  This will insert in L0

- Insert of the 11th tuple : insert into test values (11,10);
  At this point L0,L1_0 and L1_1 all are at its maximum capicity. So 1st the L1_0 and L1_1 sequentially are flushed into L2 index block and truncated. Then L0 entries are appended to L1_0 as per above procedure.

This above process is followed everytime.

Further work that can be implemented

- Increasing concurrency by using locking strategies of RowExclusiveLock
- Searching using stepped merged LSM
- Searching in LSM index.
- Application of bloom filter for efficient searching.

Learning

- Page layout mechanism of database.
- Metadata storage.
- Index creation, truncation, scaning, etc.
- Setting index attributes such as indisvalid.

Conclusion

We were able to understand the indexing in postgresql. Creation of generalized step merge tree with configurable sizes of trees. Our source code can be found here [3]

Bibliography

[1] Postgresql. [Online]. Available: https://github.com/postgres

[2] Lsm tree index. [Online]. Available: https://git.cse.iitb.ac.in/parasgarg/lsm

[3] Source code. [Online]. Available: https://github.com/Abhinav1717/LSM-TREE-IMPLEMENTATION