Tuple Locking redesigned

Kuntal Ghosh
(Senior Software Engineer)

PGCon 2019
31.05.2019

Introduction

- At EnterpriseDB, we are working on a new storage format called ‘ZHeap’, which will provide better control over bloat.
- In ZHeap, whenever possible, we handle an UPDATE by moving the old row version to an undo log, and putting the new row version in the place previously occupied by the old one.
- The ability to lock a tuple is an important feature of any database storage engine (for concurrency).
- How we have implemented *tuple locking* in ZHeap.

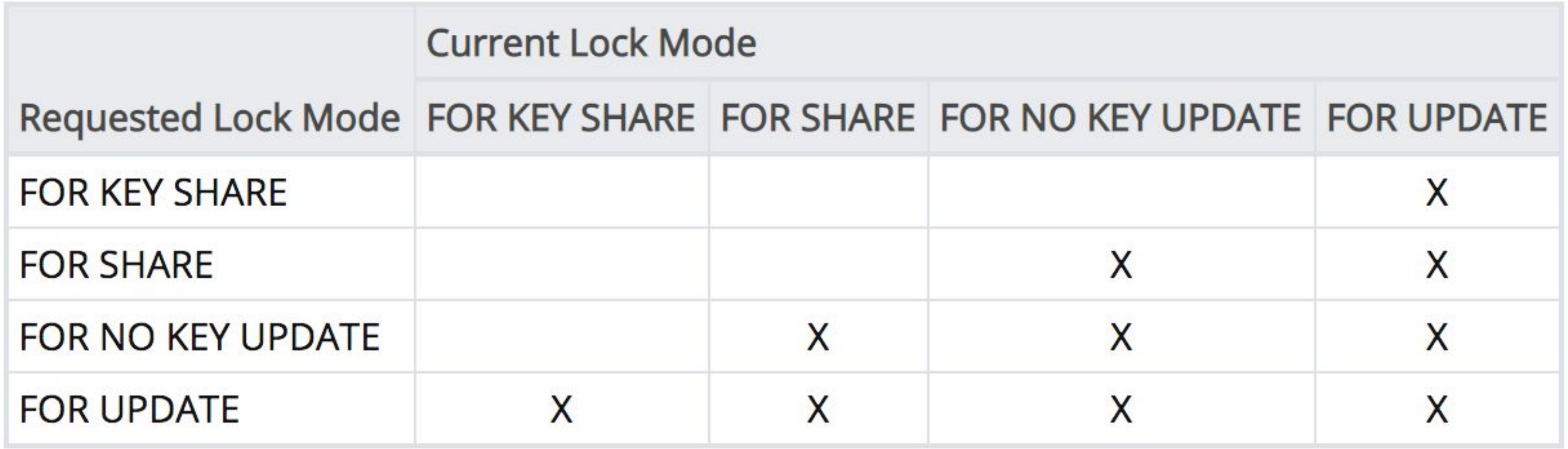
(Secret - We’ve not used Multixact machinery)

What is Tuple Locking?

Tuple Locking

- Tuple locking is needed to ensure that no two transaction can update the same tuple at the same time.
- This guarantee must persist till the end of the transactions.
- Different tuple locking modes in PG
    - FOR UPDATE
    - FOR NO KEY UPDATE
    - FOR SHARE
    - FOR KEY SHARE

Tuple Locking



	Current Lock Mode			
Requested Lock Mode	FOR KEY SHARE	FOR SHARE	FOR NO KEY UPDATE	FOR UPDATE
FOR KEY SHARE				X
FOR SHARE			X	X
FOR NO KEY UPDATE		X	X	X
FOR UPDATE	X	X	X	X

https://www.postgresql.org/docs/11/explicit-locking.html

How does tuple locking work in heap?

Tuple Locking in heap

- If we want to lock a newly inserted tuple, we can store the locking information on the tuple and set xmax as our transaction.
- But, what will happen if there are more lockers?
- We certainly can’t store them in the tuple.
    - Lack of space

https://www.pgcon.org/2012/schedule/attachments/246_Improving%20Foreign%20Key%20Concurrency.slides.pdf

Tuple Locking in heap - MultiXact

- We create an array of locking Xids along with their lock mode and we associate the array entry with an uint4 key - called MultiXactId.
- We set the tuple xmax as the MultiXactId.
- We also mark the infomask to indicate that the xmax is a MultiXactId.

https://www.pgcon.org/2012/schedule/attachments/246_Improving%20Foreign%20Key%20Concurrency.slides.pdf

Tuple Locking in heap

- If xmax is invalid
  - Grab the lock, set xmax as locker’s xid
- If xmax is valid
  - If it conflicts with current lockmode
    - Grab the lockers list and sleep on it. Restart the process when you’re awaken
  - Else, note the locker. And,
    - if a single xact, create a multixact with the two, set it as the xmax
    - if a multixact, expand it by adding yourself, create a new multixact entry and set it as the xmax

https://www.pgcon.org/2012/schedule/attachments/246_Improving%20Foreign%20Key%20Concurrency.slides.pdf

Tuple Locking in heap

How to follow the update chain

- When locking a tuple, we've to lock the future versions of the tuple.
- Failing to lock the updated row would allow a future transaction to delete the updated row when the locking transaction is still running.
- Needs a separate WAL record.
- This step is required in EvalPlanQual path as well.

https://www.pgcon.org/2012/schedule/attachments/246_Improving%20Foreign%20Key%20Concurrency.slides.pdf

Tuple Locking in heap - Bottlenecks

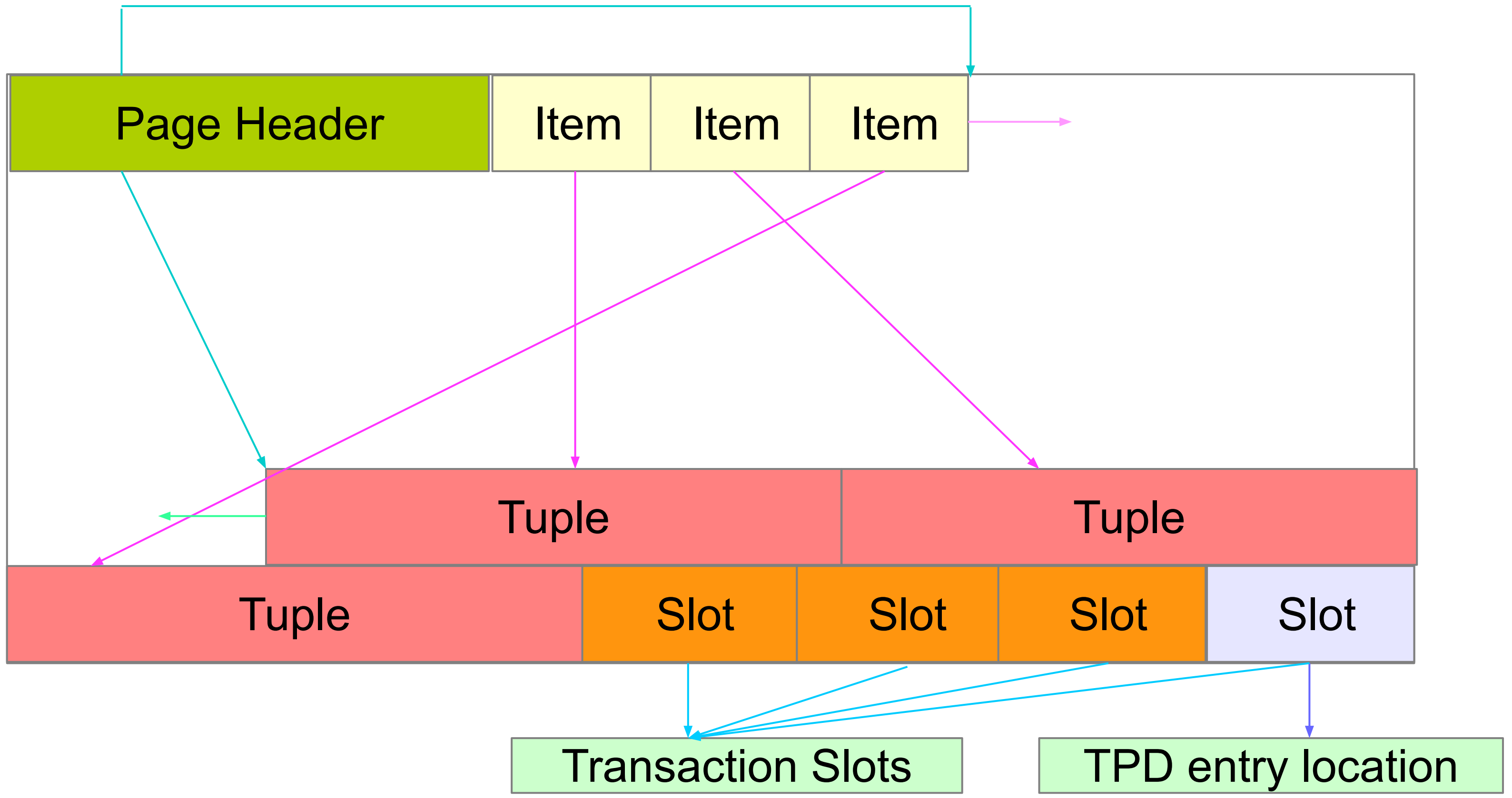
- Each new locker on a tuple combines the existing lockers along with itself and creates a new MultiXact entry.
- The 32 bit counter MultiXact Id is maintained for each MultiXact entry which requires efficient aging management, storage cleanup, and wraparound handling.
- Since any update in heap is non-inplace update, we’ve to follow the update chain and lock future versions.
- Whenever a new multixact entry is created, a new WAL record is inserted.
- MultiXact may trigger full-table vacuum.
- And the list goes on ....

ZHeap page and tuple format

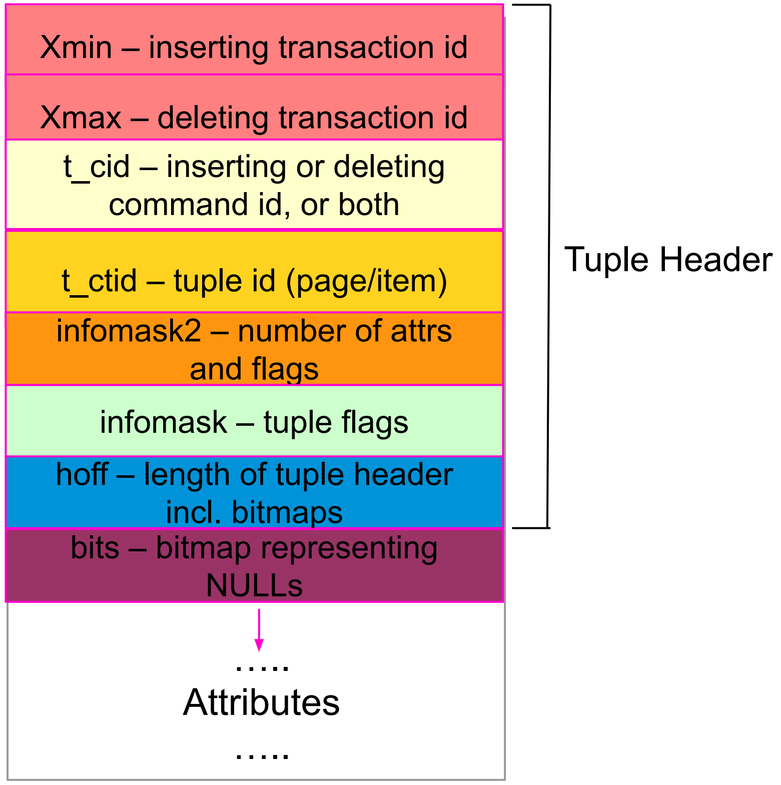
ZHeap: Page Format

- Each ZHeap page has fixed set of transaction slots containing transaction info (64-bit transaction id and the latest undo record pointer of that transaction).
- Each transaction slot occupies 16 bytes.
- As of now, the number of slots are configurable and default value of same is four.
- We allow the transaction slots to be reused after the transaction becomes too old to matter (older than oldest xid having undo), committed or rolled back. This allows us to operate without having too many slots.

ZHeap: Page Format

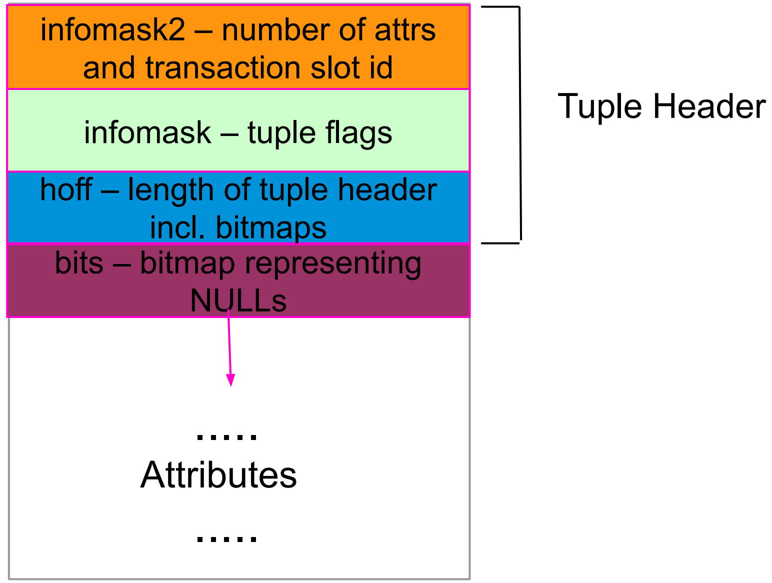


ZHeap: Tuple Format



Xmin – inserting transaction id
Xmax – deleting transaction id
t_cid – inserting or deleting command id, or both
t_ctid – tuple id (page/item)
infomask2 – number of attrs and flags
infomask – tuple flags
hoff – length of tuple header incl. bitmaps
bits – bitmap representing NULLs
…..
Attributes
…..
Tuple Header

heap Tuple



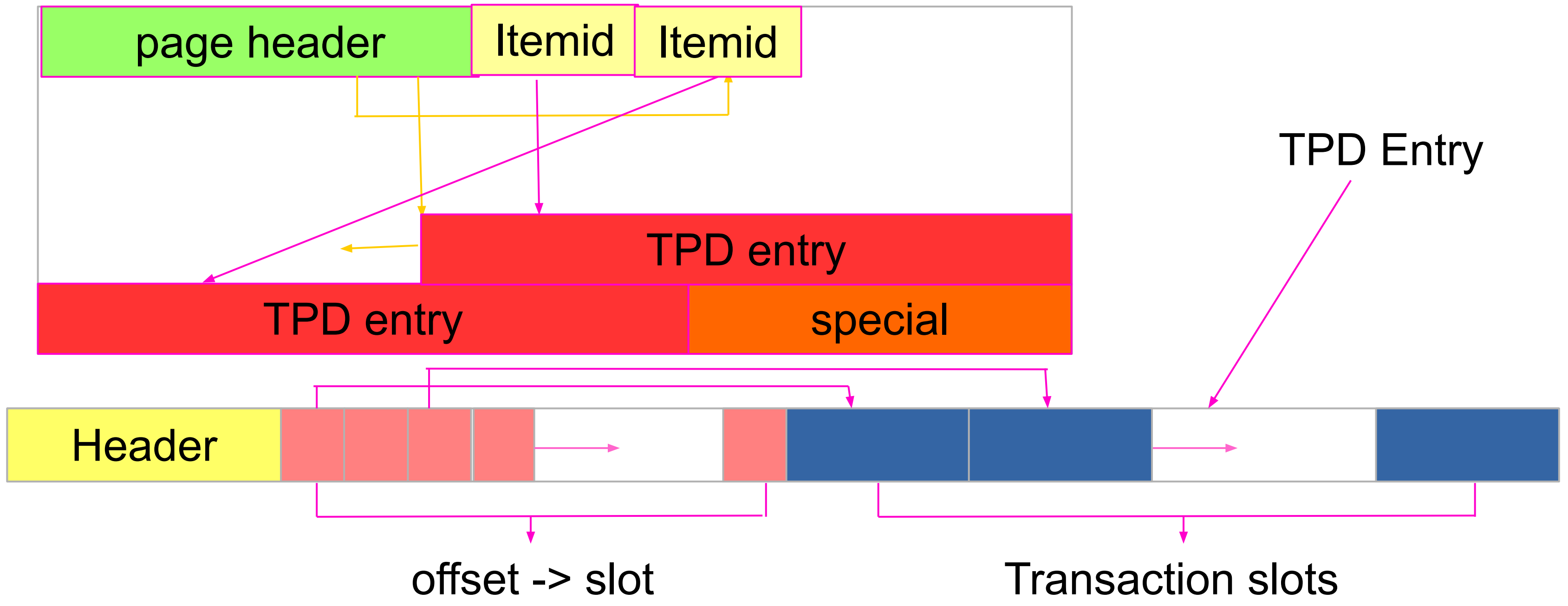
infomask2 – number of attrs and transaction slot id
infomask – tuple flags
hoff – length of tuple header incl. bitmaps
bits – bitmap representing NULLs
…..
Attributes
…..
Tuple Header

ZHeap Tuple

TPD

- TPD is nothing but temporary data page consisting of extended transaction slots from zheap pages.
- Why we need TPD?
  - In the ZHeap page we have fixed number of transaction slots which can lead to deadlock.
  - To support cases where a large number of transactions acquire SHARE or KEY SHARE locks on a single page.
- The TPD overflow pages will be stored in the ZHeap itself, interleaved with regular pages.
- The idea of putting TPD in zheap was of **Andres Freund**

TPD



Tuple headers normally point to the transaction slot responsible for the last modification, but since there aren't enough bits available to do this in the case where a TPD is used, an offset -> slot mapping is stored in the TPD entry itself.

UNDO retention

- UNDO data needs to be retained till the active transactions need to see old versions
  - All transactions which are in-progress
  - For aborted transactions till the time UNDO actions have been performed
  - For committed transactions till the time they are all-visible
- We could reduce the time period for which UNDO needs to be retained in category 3 by implementing “snapshot too old”.
- We consider undo for a transaction to be discardable once its XID is smaller than oldestXmin.

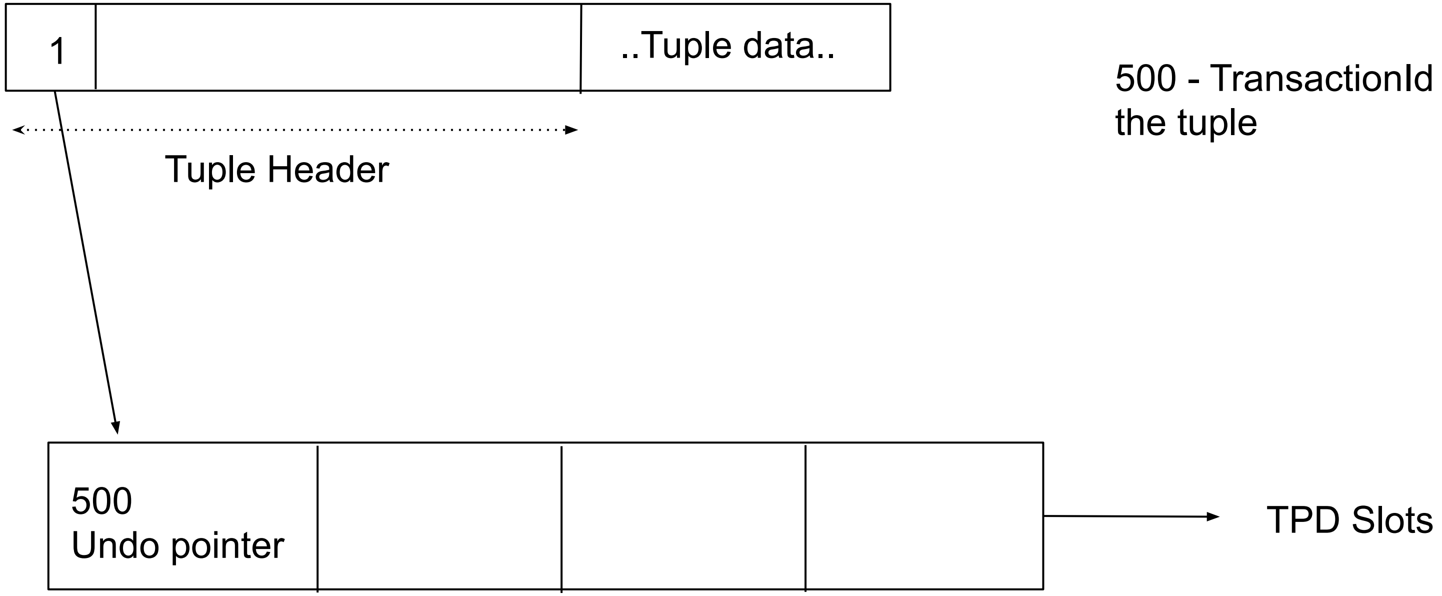
How does tuple locking work in ZHeap?

Tuple Locking in ZHeap

If we want to lock a newly inserted tuple,

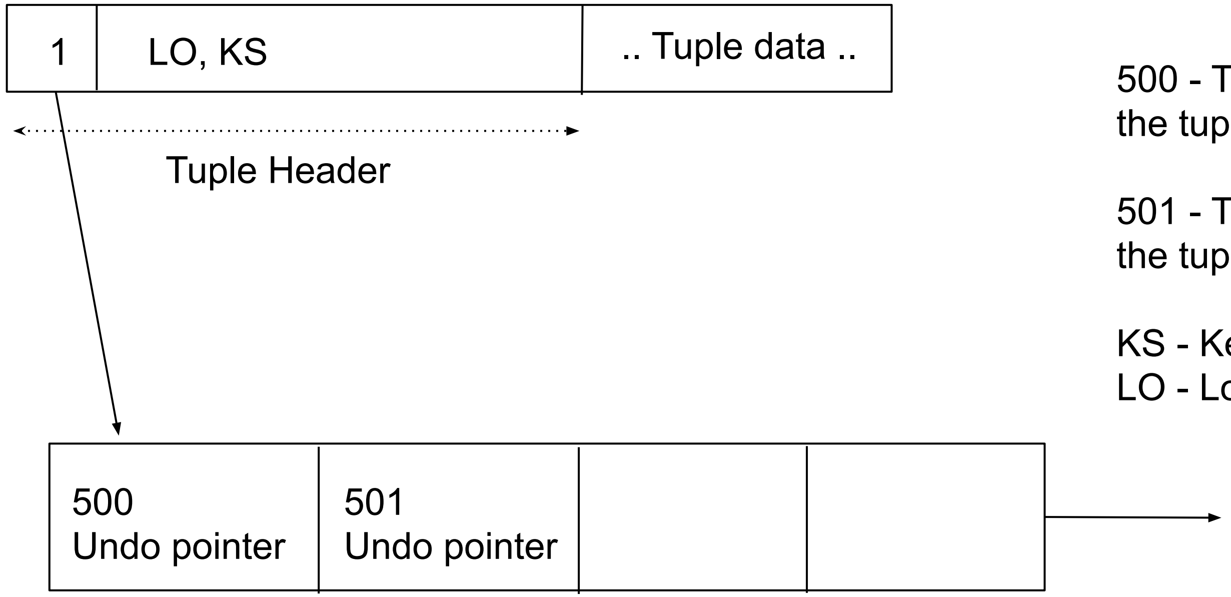
- store lock mode and lock-only flag in the tuple infomask
- insert an undo record which mainly includes
  - locking information
  - transaction information for the tuple
- store the 64-bit xid and the undo record pointer in a transaction slot
- We don’t set the transaction slot of the tuple to the locker’s transaction slot.
  - We always keep the slot of the latest xid that has inserted/updated/deleted the tuple.

Tuple Locking in ZHeap



500 - TransactionId that inserted the tuple

Tuple Locking in ZHeap



TPD Slots

500 - TransactionId that inserted the tuple

501 - TransactionId that locked the tuple in key-share mode

KS - Key Share lock mode
LO - Lock Only flag

Tuple Locking in ZHeap

If the tuple is modified,

- If the lock-mode doesn’t conflict with current lock mode, grab the lock as before. In this case, we store the highest lock mode on the tuple.
- Else, make a list of xids that has locked the tuple in conflicting lock modes

```
xid_list = {};
for each slot in the page including TPD entry (if any)
do
        if (slot->xid is in progress)
                Visit its undo chain and check if it has locked the tuple in a conflicting lock-mode;
                if yes add xid to xid_list; endif
        endif
done
```

Tuple Locking in ZHeap

If the tuple is modified,

- If the lock-mode doesn’t conflict with current lock-mode, grab the lock as before. In this case, we store the highest lock mode on the tuple.
- Else, make a list of xids that has locked the tuple in conflicting lock modes
    - If the list is not empty, sleep on it. Restart the process when you’re awaken
    - If the list is empty, go ahead and grab the lock. In this case, we store the current lock mode on the tuple.

Tuple Locking in ZHeap

Follow the update chain

- If the tuple is in-place updated, don’t have to follow the update chain. Since, we find the conflicting xids from undo using ctid, we can fetch lockers of all the versions from undo.
  - Most favourable case
  - Reduce write amplifications
- If the tuple is non-in-place updated, we’ve to lock all future versions as well.
- This step is required in EvalPlanQual path as well.

Tuple Locking in ZHeap - visibility

- No need to retrieve the lockers from undo for the cases that wouldn’t block.
- Otherwise, we’ve to traverse undo chains to collect lockers.
- We’ve kept a multi-locker flag on the tuple to indicate whether the tuple has a single locker or multiple lockers.
- For single locker case, as soon as we get the locker, no need to process other undo chains.

Tuple Locking in ZHeap - rollbacks

- If the tuple has only one locker
  - We clear the transaction slot corresponding to the aborted transaction.
  - We reset the lock mode on the tuple.
  - We keep the previous inserter's/updater's slot as it is. Because, while locking a tuple we don't modify the existing the transaction slot on the tuple.
- If the tuple has multiple lockers
  - We clear the transaction slot corresponding to the aborted transaction.
  - We don't apply any rollback operations on the tuple.
  - We keep the previous inserter's/updater's slot as it is.
  - For any subsequent operation that wants to lock the tuple, only keeps the lock modes of in-progress transactions.

Tuple Locking in ZHeap - WAL

- To lock a tuple, we emit a single WAL record with enough information so that we can form the undo record from the same during recovery.
- We also need to emit a WAL record while applying the rollback actions.

Tuple Locking in ZHeap - advantages

- Removed the requirement of 32-bit MultiXactId
  - No MultiXact wraparound issues
- Undo records corresponding to all-visible lockers are discarded by the discard worker.
  - No access needed on ZHeap pages
- For each new locker, one undo record is inserted containing current lock mode only.
  - In heap, we combine previous lockers and lock modes in the newly created pg_multixact entry.
- Other advantages of not using MultiXact system

Tuple Locking in ZHeap

Performance

- Hardware
  - 128 cores 2.13 GHz GenuineIntel processors with 500GB RAM
- Test
  - Pgbench with scale factor 1000
  - Two scripts with 0.5 weight in each for 15 minutes
    - Script 1: UPDATE pgbench_accounts WHERE aid = :aid;
    - Script 2: SELECT FROM pgbench_accounts WHERE aid = :aid FOR KEY SHARE;
- Results
  - Heap - 71531
  - ZHeap (with 16 slots) - 70667

Tuple Locking in ZHeap

Scope of improvements

- While locking a tuple in exclusive mode, we can store the transaction slot on the tuple. In that case, we don’t have to traverse all the undo chains in a page to get the locker.
- We write undo records for locking each tuple. So, for the statements like *select * from foo for share* generates an undo record for each tuple.
- While collecting conflicting lockers, we need to traverse the undo chains of all in-progress transactions.

Who?

- Amit Kapila (development lead)
- Dilip Kumar
- Kuntal Ghosh

A special thanks to Robert Haas, Andres Freund and Thomas Munro who have provided a lot of valuable design inputs.

Questions?

Thanks!

https://github.com/EnterpriseDB/zheap