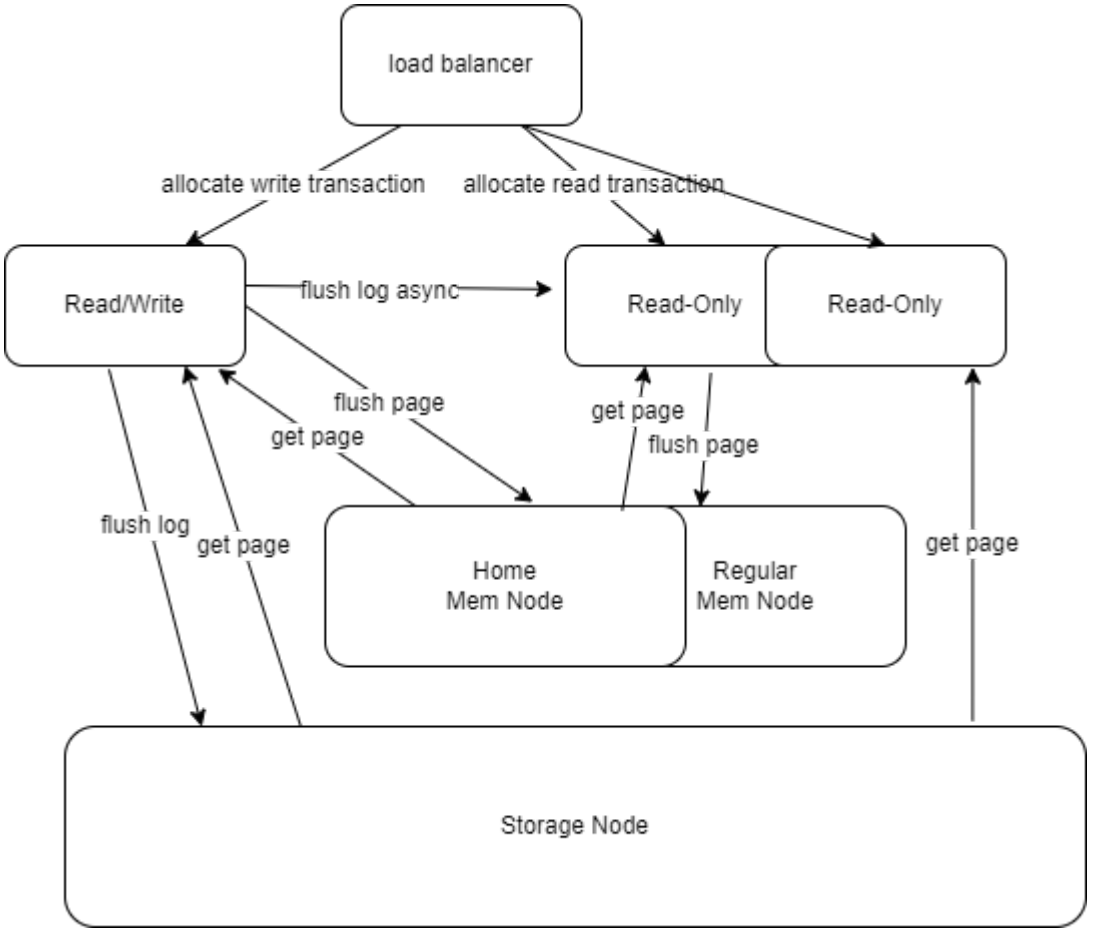
Memory Disaggregation Database

Key words:

- recovery
- multi-version
- log replication
- lock
- cache consistency
- storage

Overview

High Level Design



Low level Design

1. Components
    a. Load balancer
    b. Compute nodes
        i. Read/Write node
            - Pages in local memory: page data, page id, LSN
            - xlog
            - LRU list: all pages in local memory

ii. Read-Only nodes
- pages in local memory
- xlog
- LRU list

c. control memory node
- Page array (PA): all page data in memory pool. For different versions of each page, only a base page is stored.
- Page Address Table (PAT): hashmap maintaining each page's location in memory pool and LSN.
- Page reference directory (PRD): hashmap maintaining a list of locations in compute nodes for each page
- Page Latch Table (PLT): global lock for each page, indicating in which LSN intervals the page is exclusive.
- LRU list: pages with 0 reference

d. Regular memory nodes
- Page array

e. Storage (OpenAurora)
- Page Data (RocksDB)
- Log files
- Background log replay process
- parse process
- get-page service

2. Operations
   a. Load page
      i. Check whether it is in local memory and valid. If so, return.
      ii. Check whether a base version of the page is in mempool and its LSN is not greater than the requested one. If so, fetch it. Evict a local cold page if necessary. Replay local xlog so that its version corresponds to requested LSN. And update PRD and LRU lists.
      iii. Otherwise, fetch the requested version of the page from storage (as in open aurora), and replay local xlog so that its version corresponds to the requested LSN. Also load it into mempool as the base version. Evict cold pages locally and in mempool if necessary. Update information in the local node and mempool.
   b. Evict page from local memory
      i. With LRU list, using LRU with midpoint.
      ii. Update PRD and LRU list in mempool.
   c. Evict page from mempool
      i. With LRU list, using LRU without midpoint.
      ii. Update information in mempool.
   d. Read
      i. Start Transaction
      ii. Check lock for b+tree consistency
      iii. Load the corresponding pages of version LSN_start.
      iv. Read data locally
      v. End Transaction
   e. Write
      i. Start Transaction

        ii. Load the corresponding pages of the newest version.
        iii. Obtain locks from PLT for b+tree consistency.
        iv. Flush xlog to storage.
        v. Write to local memory.
        vi. Commit.
        vii. Asynchronously flush xlog to mempool/RO nodes.
3. Recovery
    a. R/W node fail
        i. Promote the R-O node with the largest LSN to R/W
        ii. New R/W fetches the xlog it has not received from storage
        iii. New R/W replays xlog.
        iv. fix PRD
        v. fix PLT
        vi. Asynchronously rollback uncommitted transactions
    b. R-O node fail
        i. Start a new R-O node
        ii. Load balancer reallocates the read transactions
    c. regular memory node fail
        i. Start a new regular mem node
        ii. control mem node scans PAT, and loads corresponding pages into it from the storage node.
    d. control memory node fail
        i. If its metadata is backed up synchronously in another mem node, then promote the regular mem node to a control mem node.
        ii. Otherwise, recover all compute nodes and mem nodes from the storage node.
4. MVCC
    a. Snapshot isolation: previous versions of a record are constructed with undo logs

Todo:

- Do we flush xlog to mempool and replay xlog in mempool? If not, what if a page is frequently loaded from mempool but never evicted from mempool?
    - If the page's LSN does not update, the replay process becomes more and more time-consuming.
    - If xlog is flushed to mempool, and the minimum of RO nodes' newest LSN is sent to mempool asynchronously, mempool could replay xlog to min{LSN_newest}.
- Read a version of a page that is b+tree inconsistent. How to maintain PLT?
    - Choice 1: For each page, PLT maintains the LSN intervals where the page latch is locked.
    - Choice 2: PLT maintains page latch in the original way, while a bit in each page is allocated to indicate whether the page latch is locked at that LSN, and requiring and releasing page latch would flip the bit.

notes:

- mvcc: undo -> redo
- page latch for b+tree: ignore for now
- r/w node updates version of base page in mempool if min{lsn_{ro}} > lsn_{base}
- buffer pool cache miss: -> memory node