Fine-Grain Checkpointing with In-cache-Line Logging

Main idea: How to keep data structures consistent after a system crash for non-volatile main memory?

For lock based data-structures, what happens when locks are released? A lock is a value in memory that is atomically updated. This is used in multithreaded applications.In order to make sure that concurrent threads don't make changes at once, locks are in place to ensure atomic updates and consistent data structures.

NVM with locks: If the variable of lock is stored in main memory, then saving that value can preserve state.

Flushing every single write is impractical since we have to wait for that operation to complete before moving on. Transaction logging is somewhat better but still involves flushing. Checkpointing is better of the 3 since it saves everything at periodic intervals to durable memory.

This paper: Fine grained checkpointing + in-cache-line logging

Fine grained checkpointing is checkpointing at frequent intervals. In-cache line logging allows us to restore the data structure to the previous checkpoint stored at the beginning of the epoch.

Masstree leaf node:

If insert(k,v) = (100, [1,2,3]) stored at address location A000, then insert(k,v) = (100,xA000)

After storing K,V pair, the permutation parameter is updated. It does not matter how many inserts we do, since we can revert the permutation field.

Delete: update the permutation field to mark a deletion or to undo a deletion.

The structure is very specific to the data structure discussed. It makes it less scalable.

How would this relate to other data structures?

This is something to think about. How do we evaluate the efficiency trade-off if they had used a different data structure? No comparisons have been made.

If there are a lot of writes after a system crash, then it will increase recovery time.

In the benchmark, they have tested only inserts and not updates.