Lecture #13: Checkpoint Protocols

15-721 Advanced Database Systems (Spring 2018)
http://15721.courses.cs.cmu.edu/spring2018/
Carnegie Mellon University
Prof. Andy Pavlo

1 In-Memory Checkpoints

There are different approaches for how the DBMS can create a new checkpoint for an in-memory database. The choice of approach in a DBMS is tightly coupled with its concurrency control scheme. The checkpoint thread(s) scans each table and writes out data asynchronously to disk.

Ideal Checkpoint Properties: [5]

- Do **not** slow down regular transaction processing.
- Do **not** introduce unacceptable latency spikes.
- Do **not** require excessive memory overhead.

1.1 Checkpoint Types

Approach #1 – Consistent Checkpoints:

- Represents a consistent snapshot of the database at some point in time.
- No uncommitted changes.
- No additional processing during recovery. Still have to replay the log if not a complete shutdown of the system.

Approach #2 – Fuzzy Checkpoints:

- The snapshot could contain records updated from transactions that have not finished yet.
- Must do additional processing to remove those changes in the log.
- Write a `BEGIN` and `END` record to the log to know when the checkpoint was active.

1.2 Checkpoint Contents

Approach #1 – Complete Checkpoint:

- Write out every tuple in every table regardless of whether were modified since the last checkpoint.
- Nearly every system does this approach.

Approach #2 – Delta Checkpoint:

- Write out only the tuples that were modified since the last checkpoint.
- Can merge checkpoints together in the background.

1.3 Checkpoint Frequency

Taking a checkpoint too often causes the runtime performance to degrade However, waiting a long time between checkpoints is just as bad.

Approach #1 – Time-based

- Periodically take a new snapshot after some amount of time.

Approach #2 – Log-File Size Threshold

- Take a new snapshot after the DBMS has written a certain amount of data to the log file.

Approach #3 – On Shutdown

- Take a snapshot whenever the administrator tells the system to shutdown.

2 Checkpoint Algorithms

There are several approaches for implementing a checkpoint algorithm for an in-memory DBMS [1].

Checkpoint implementation primitives:

- **Bulk State Copying:** Pause transaction execution to take a snapshot.
- **Locking/Latching:** Use latches to isolate the checkpoint thread from the worker threads if they operate on shared regions.
- **Bulk BitMap Reset:** If DBMS uses BitMap to track dirty region, it must perform a bulk reset at the start of a new checkpoint.
- **Memory Usage:** To avoid asynchronous writes, the method may need to allocate additional memory for data copies

2.1 Naive Snapshots

Block all transactions, and create a consistent copy of the entire database in a new location in memory and write the contents to disk.

Two approaches for copying

- Do it yourself (tuple data only).
- Let the OS do it for your (everything).

The **HyPer** DBMS originally used OS fork snapshots [3]:

- Create a snapshot of the database by forking the DBMS process.
- Child process contains a consistent checkpoint if there are no active transactions.
- Otherwise, use the in-memory undo log to roll back transactions in the child process.
- Continue processing transactions in the parent process.

2.2 Copy-on-Update Snapshots

During the checkpoint, transactions create new copies of data instead of overwriting it. Copies can be at different granularities (block, tuple). The checkpoint thread then skips anything that was created after it started. Old data is pruned after it has been written to disk.

This is trivial to do in an multi-versioning DBMS with snapshot isolation.

Although **VoltDB** is not a multi-versioned DBMS, it still supports consistent checkpoints [4]:

- A special transaction starts a checkpoint and switches the DBMS into copy-on-update mode.
- Changes are no long made in-place to tables. The DBMS tracks whether a tuple has been inserted, deleted, or modified since the checkpoint started.
- A separate thread scans the tables and writes tuples out to the snapshot on disk. It also ignores anything changed after the checkpoint and cleans up old versions.

2.3 Wait-Free ZigZag

The DBMS maintains two copies of the entire database. Each write only updates one copy. This requires two BitMaps to keep track of what copy a transaction should read/write from per tuple.

Avoids the overhead of having to create copies on the fly as in the copy-on-update approach.

2.4 Wait-Free PingPong

Trade extra memory + CPU to avoid pauses at the end of the checkpoint. The DBMS maintains two copies of the entire database at all times plus a third "base" copy. Pointer indicates which copy is the current master. At the end of the checkpoint, swap these pointers.

3 Shared Memory Restarts

Not all DBMS restarts are due to crashes:

- Updating OS libraries
- Hardware upgrades/fixes
- **Updating DBMS software**

Thus, we need a way to be able to quickly restart the DBMS without having to re-read the entire database from disk again.

Facebook's **Scuba** DBMS is in-memory DBMS for time-series event analysis and anomaly detection. They want to be able to restart the DBMS to upgrade it to a new version without having to load the last checkpoint from disk back into memory [2].

The main idea is to decouple the in-memory database lifetime from the process lifetime using shared memory (SHM). By storing the database shared memory, the DBMS process can restart and the memory contents will survive.

Approach #1 – Shared Memory Heaps:

- All data is allocated in SHM during normal operations.
- Have to write a custom allocator to subdivide memory segments for thread safety and scalability. Cannot use lazy allocation of backing pages with SHM.

Approach #2 – Copy on Shutdown:

- All data is allocated in local memory during normal operations.
- When the admin initiates a restart command, the node halts ingesting updates and the DBMS copies data from heap to SHM. Once snapshot finishes, the DBMS restarts.
- On start up, the DBMS checks to see whether there is a valid database in SHM to copy into its heap. Otherwise, the DBMS restarts from disk.

References

[1] T. Cao, M. V. Salles, B. Sowell, Y. Yue, A. Demers, J. Gehrke, and W. White. Fast checkpoint recovery algorithms for frequently consistent applications. In *SIGMOD '11 Proceedings of the 2011 ACM SIGMOD International Conference on Management of data*, pages 265–276, June 2011. URL https://dl.acm.org/citation.cfm?id=1989352.

[2] A. Goel, B. Chopra, C. Gerea, D. Matani, J. Mezler, F. U. Haq, and J. Wiener. Fast database restarts at facebook. In *SIGMOD '14 Proceedings of the 2014 ACM SIGMOD International Conference on Management of data*, pages 541–549, June 2014. URL https://dl.acm.org/citation.cfm?id=2595642.

[3] A. Kemper and T. Neumann. Hyper: A hybrid oltp&olap main memory database system based on virtual memory snapshots. In *ICDE '11 Proceedings of the 2011 IEEE 27th International Conference on Data Engineering*, pages 195–206, April 2011. URL https://dl.acm.org/citation.cfm?id=2005619.

[4] N. Malviya, A. Weisberg, S. Madden, and M. Stonebraker. Rethinking main memory oltp recovery. In *2014 IEEE 30th International Conference on Data Engineering*, pages 604–615, March 2014. doi: 10.1109/ICDE.2014.6816685.

[5] K. Ren, T. Diamond, D. J. Abadi, and A. Thomson. Low-overhead asynchronous checkpointing in main-memory database systems. In *SIGMOD '16 Proceedings of the 2016 International Conference on Management of Data*, pages 1539–1551, June 2016. URL https://dl.acm.org/citation.cfm?id=2915966.