State Checkpointing in Main-Memory Database Systems

Matthew Antoszyk: Yale University, 2014

Due to the importance and sensitivity of information contained within databases, which includes data related to business transactions, stock portfolios, bank accounts, and personal information, it is imperative that databases guarantee the durability of committed transactions. One influential recovery protocol originally conceived in 1992 is ARIES, which guarantees durability through write-ahead logging of undo and redo records. However, as it becomes more routine for complete datasets to fit within the main memory of one machine or a cluster of machines, the cost of writing all log records out to stable storage before transactions can commit is becoming a bottleneck to the throughput of transaction processing systems. With these main memory database systems, there is a need for a more efficient algorithm for ensuring durability.

In addition to avoiding frequent writes to disk, there are a few other desirable properties for the checkpointing algorithm of a recovery protocol for main memory databases to possess. First, the algorithm should not cause intermittent spikes in database latency. Ideally, latency should not fluctuate significantly as the recovery algorithm changes state. Second, the checkpointing process should not be extremely work-intensive. In other words, a system running at full capacity should not see a sharp drop-off in total throughput when the checkpointing algorithm is introduced. Finally, the checkpointing algorithm should require a minimal amount of additional memory overhead when compared to a database system running without the checkpointing algorithm. This means that complete multi-versioning, as is used in MVCC in PostgreSQL, would be prohibitively expensive.

Last semester a classmate and I implemented the checkpointing algorithm proposed by Thaddeus Diamond et. al. in the Calvin codebase written and maintained by Alex Thompson. Our results were very encouraging. CALC was shown to beat a naïve checkpointing algorithm (one that quiesces the entire database) without decreasing throughput significantly when compared to performance when no checkpoint is running. My project will be a continuation of this work.

CALC consists of 5 phases: rest, prepare, resolve, capture, and cleanup, which we will briefly describe. During the rest phase the checkpointing process is dormant. In the prepare phase newly begun transactions make updates to a live version of database records but maintain overwritten records in case they are needed as stable versions in the capture phase (which will be the case if the transaction that updated a particular record doesn’t commit until the resolve phase). The start of the resolve phase marks a virtual point of consistency. Records last updated by transactions that committed before the start of the resolve phase will have their values at that time reflected in the checkpoint, while all other records will have their stable values, which were their values
before they were altered by any transaction that hadn’t committed by the start of the resolve phase, reflected in the checkpoint instead. During the capture phase, a background process is spawned to scan database records and write the appropriate stable and live records to disk. Finally, in the cleanup phase all remaining stable versions that were not removed in the capture phase are removed.

One immediately apparent disadvantage to the CALC algorithm is that every checkpoint requires a complete checkpoint of the entire database. One proposed optimization by Thaddeus Diamond et. al. is a partial checkpointing version (pCALC). If there are relatively few records updated in between virtual points of consistency, it can be highly advantageous to keep track of those records to avoid unnecessary work involving unchanged records in the database. Two potential data structures for storing the records that have been updated include a hash table and a bloom filter. With a bloom filter there is a tradeoff: on the one hand, the overhead per update to the bloom filter is less than with the hash table, but on the other hand, the bloom filter can lead to false positives, causing records to be checked unnecessarily during the capture phase. One final aspect to note about the pCALC algorithm is that transactions that begin in the prepare phase can be added unconditionally to the hash table or bloom filter because if the transaction doesn’t commit before the resolve phase, in the capture phase additional records may have to be checked unnecessarily, but this will not lead to inaccurate results.

My research will contribute to his ongoing work and includes the following steps. First, Kun Ren has created a more robust version of the Calvin codebase. Thus I will need to port our implementation of CALC from Alex’s code into the new codebase. This will also offer me the opportunity to rethink and refine my implantation, improving the code that was written during the first attempt. The next goal is to rerun our earlier trials on Amazon EC2 instances. Then, I plan to the pCALC algorithm, with a hash table or with a bloom filter or both depending on performance, in Calvin, and compare performance between these algorithms.

In order to make this research relevant to the general database community I will need to have other comparators as well. I will need to compare against the naïve implementation that obtains an exclusive lock on the entire database, an Interleaved Ping-Pong checkpointing algorithm that relies on physical points of consistency, and a partial snapshot version of the Interleaved Ping-Pong checkpointing algorithm. Performance metrics that I plan to compare the algorithms on include latency over time, throughput over time under peak load, and memory usage. I plan on running the algorithms on the microbenchmark testing framkework. In the microbenchmark most transactions will operate on a small number of records making a small number of updates, while a minor percentage of transactions will conduct writes on many records. At the completion of the project, I plan to deliver a variant of Calvin equipped to run the CALC algorithm, the pCALC algorithm, thorough results for the CALC algorithm, each of
the pCALC algorithms, each of the IPP algorithms, and the naïve snapshot algorithm on performance metrics such as latency, throughput, and memory usage.

I will write up the results in a paper to be submitted to the SIGMOND database conference by the December 10th deadline. The paper will suggest CALC as a valuable checkpointing method for main memory resident databases that is similar in concept to copy-on-write. Copy-on-write is the current standard for backing up an OS. It’s not immediate applicable databases since a database needs to guarantee which transactions have completed at the time of the checkpoint. The virtual point of consistency in CALC makes this guarantee possible and thus makes a version of the often-used copy-on-write procedure available for databases.

There is a final avenue of research that I plan to explore and potentially integrate into my project. I currently am not aware how cloud services (EC2, Azure, etc..) checkpoint their databases. That is, after my initial research I could not determine the current database industry standard, likely because it is proprietary information. Ideally, I would be able to discover more information on what current services use in order to implement another comparator algorithm that is most relevant to the database community today.