Coordinating Stable Crash Tolerance for Stream Processing Systems

George Stamatiadis
Paul Ezhilchelvan

Newcastle University

February 2019

Crash-tolerance provisioning in stream processing systems commonly requires operator nodes to store recovery related information, such as checkpoints and upstream backup, and use that information to aid the recovery of neighbouring operator nodes. This requirement inevitably injects a trade-off between storage overheads and checkpointing interruptions during normal, crash-free operations: the more often the checkpoints are taken and publicised, the smaller will be the storage overhead for upstream backup and the swifter will be crash recovery; frequent checkpointing and subsequent indications to upstream operators, however, slows down the normal operation.

Furthermore, current approaches do not adequately address the issues that arise when an operator processes multiple input streams from upstream operators. The outputs produced and passed on downstream by such an operator depends not just on upstream inputs but also on how these inputs interleaved at that operator while being processed. When that operator crashes and recovers, it must maintain the same interleaving order in reaching the pre-crash state from the check-pointed state; otherwise, post-crash outputs can be different to their pre-crash counterparts and downstream operators, though not crashed, need to rollback for recovery to be consistent. This leads to a domino effect in the downstream end. Worse still, the end user will have to accept new outputs that are different from the ones received prior to crash. One of the aims of this investigation is to ensure that an operator's outputs to a downstream neighbour or to the end user be *stable* despite crash recovery: an output, once delivered, may be re-issued but are never substituted for a different one. In addition, we will seek to minimise crash-recovery related storage and processing overheads on operator nodes. Meeting these aims requires taking a novel approach that is fundamentally different to the existing ones.

We will present a crash-tolerant architecture wherein the operator nodes continually flush-out recovery information they accrue, as message piggybacks to a reliable *house-keeper* system, whenever they output to a downstream entity. House-Keeper then coordinates crash-recovery using the recovery information it holds for the entire system. This leads to negligible storage and processing overhead on operator nodes, stable output delivery and to provably-minimal recovery latencies. For example, in the absence of multiple crashes, a crashed node needs to re-start only from its latest checkpoint and, during recovery, no operative node needs to roll-back and outputs are always guaranteed to be stable.