Report 4: Rethink the Sync

Dalia Camacho García-Formentí,
Instituto Tecnológico Autónomo de México (ITAM)
Mexico City, Mexico

I. Summary

In the article 'Rethink the Sync' [1], Nightingale *et al.* introduce the concept of *external synchrony* which refers to executions, that may or may not be synchronous, but to the user it is indistinguishable from a synchronous execution.

Compared to previous approaches external synchrony aims to be consistent from the users point of view when the system crashes (durability), but without aggravating performance. To achieve the latter external synchrony is user-centric instead of application-centric. Thus, only outputs sent to external devices require a specific order, this corresponds to the causal order in which events happened. Moreover, when a user observes an output, the necessary changes are already saved in the disk.

It is important that changes are committed in the disk before showing them to the user, because if they remain in cache and a crash occurs before rewriting the cache line in the disk, information in the disk will not be consistent with what the user observed. To overcome this problem the authors suggested either disabling the cache drive, or flushing the cache every 5 seconds and whenever an external output is required (output-triggered commits).

To avoid performance loss, several modifications are grouped before committing to disk. This was done to avoid multiple writes to disk by taking advantage of spatial and temporal locality. Another technique they used was buffering the output to external devices until changes to disk were committed. By buffering the outputs, subsequent operations could continue to execute without being dependent on showing the output to the user.

For the implementation Nightingale *et al.* used *Speculator*, which predicts and begins the following execution, before the previous ends. Speculator saves a checkpoint to return to in case the prediction is incorrect, this checkpoint is erased if the prediction is correct. No speculative results are sent to external devices before they are confirmed to be correct.

Speculator was modified so it could manage external synchrony. It keeps a log of changes done to external file systems with a commit dependency. Once changes are committed to disk, the commit dependency is removed. The modifications also include the propagation of commit dependencies to processes with shared memory.

External synchrony was implemented in a modified version of *ext3*, which they named *xsyncfs*. The journaling mode was used to maintain causal order, since modifications are added to the journal in the order of completion.

They evaluated durability when a loss of power occurred, in the cases of external synchrony, asynchrony, synchrony, and synchrony with write barriers that flush the cache whenever a write instruction appears. Only external synchrony and synchrony with write barriers were durable.

They evaluated performance for four benchmarks, in terms of speed they found that *xsyncfs* is very similar to the asynchronous version of *ext3*, and better than the synchronous versions in both PostMark and Apache benchmarks. In terms of transactions per minute in the MySQL benchmark xsyncfs is better than ext3 with write barriers for less threads. However, as the number of threads increases the performance of xsyncfs worsens, while the performance of ext3 with write barriers increments. Finally in terms of throughput in the SPECweb99 benchmark ext3 in the synchronous and asynchronous modalities was better than xsyncfs.

II. Analysis

External synchrony, the method proposed by the Nightingale *et al.*, is focused mainly to achieve durability. This means that what the user observes is what is on the disk, and if the system crashes or a power loss occurs things are consistent on reboot. The idea behind this seems quite good, because the user changes, that have been saved to the disk, in their causal order.

In terms of durability it seems quite good, but there could still be some cases in which durability is not ensured. As examples of possible issues consider that durability fails if the system crashes or power fails once changes are in the disk, but before displaying them; or if errors occur while writing or displaying the results; or if speculative execution results fatal. Although these issues may not occur often, an analysis of their frequency may help to verify if they are negligible. If they are, then external synchrony is a good guarantee of durability with a general better performance than usual synchrony with write barriers.

For multiple clients performance diminishes, because external outputs are required more often and output triggered commits do not allow for large groupings. They only evaluated throughput up to 20 threads, but it would have been best to do it for more threads to compare if external synchrony converges to synchrony with write barriers, or if it reaches a point where synchrony with write barriers is better, or if external synchrony is always at least slightly better.

Performance is better in the asynchronous model and depending on the application usual synchrony may be better than external synchrony. Thus, one should evaluate what is best for the system and which properties are desired. If durability is not that important, because the user does not receive many external outputs, then the asynchronous model would be better. However, if users are constantly observing results, then durability is more important and external synchrony is a better approach.

Overall, external synchrony is an interesting idea that makes sense for the user experience and it can avoid durability issues.

References

[1] E. B. Nightingale, K. Veeraraghavan, P. M. Chen, and J. Flinn, "Rethink the sync," in *Proceedings of the 7th Symposium on Operating Systems Design and Implementation*, OSDI '06, (Berkeley, CA, USA), pp. 1–14, USENIX Association, 2006.