Eventually Durable Replicated State Machines

Kriti Kathuria, Ken Salem

University of Waterloo
first.last@uwaterloo.ca

Consider a replicated key-value (KV) store as an example of a replicated state machine. It consists of a key-value store at each site and a replicated log. The interface it exposes is *put(key, value)* and *get(key)*. *put* creates an entry in the replicated log, and once the entry has achieved majority replication, creates/updates the key-value pair. *get* returns the latest value of the key. The KV store guarantees that if a *put* succeeds, a subsequent *get* will see its effect regardless of failures. This write durability can be guaranteed because the *put* does not return until it has achieved majority replication. Therefore, durability is an expensive guarantee.

Hence, latency-sensitive applications may choose to forgo durability for better performance in an ad-hoc manner, like acknowledging put operations without waiting for them to fully replicate. We present Eventually Durable (ED) replicated state machines and establish a principled approach for the applications to reason about performance/durability tradeoffs they already make.

Like a regular KV store, an ED KV store also consists of a key-value store at each site and a replicated log. It also exposes a *put* and a *get* interface. *get* works like in the regular KV store. *put*, on the other hand, does not guarantee durability when it returns. A put may become durable eventually, after it returns. In the meantime, the application may proceed under the assumption that the put will eventually become durable. That is, the application can speculate on the eventual durability of the put operation.

Additionally, the ED KV store provides a *sync* operation which returns after all preceding puts, and as a result, values read by preceding gets, have become durable. *sync* is a tool for the application to resolve speculation and thus, manage the risks associated with durability speculation.

Applications accept the risk that the ED KV store will lose some acknowledged puts in the event of a failure. The ED model provides clear failure semantics that allow applications to reason about possible data loss. Specifically, the ED model guarantees that a failure will result in the resolution of all existing speculation — the speculative entries that survive the failure will become durable and will survive forever. The entries that get lost will never reappear. In this way, there will only ever be a single speculative "future" for the ED KV store.

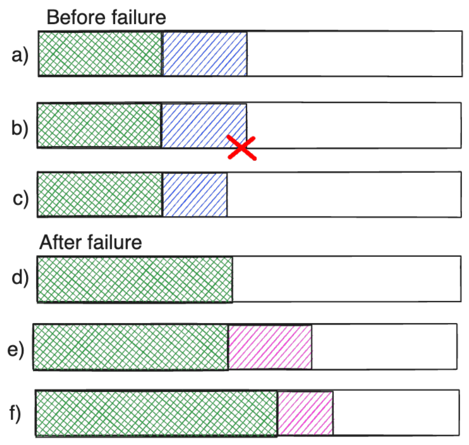

Figure 1: Behaviour of the replicated log

Fig.1 illustrates the behavior ED KV store's underlying replicated log across a failure. Green depicts the durable portion of the log and blue/pink depicts the speculative portion. (a, b, c) show the log before a failure, and (d, e, f) show the log after the failure. At first, the log contains some durable entries and some speculative entries (a). Then there is a failure (b) due to which some speculative entries disappear (c). After the failure, the surviving pre-failure speculation has become durable (d). The application starts speculating again (e) and as time progresses, a prefix of the new speculative entries is made durable, and more speculative entries are added (f).

To support an ED replicated state machine, we have developed an ED variant of the Raft consensus algorithm[1]. ED Raft acknowledges new log proposals without waiting for replication. Failures lead to the loss of speculative log proposals. We show that ED Raft supports the failure semantics described above.

References

[1] Diego Ongaro and John Ousterhout. In search of an understandable consensus algorithm. In *Proceedings of the 2014 USENIX Conference on USENIX Annual Technical Conference*, USENIX ATC'14, page 305–320, USA, 2014. USENIX Association.