DynaStar: Optimized Dynamic Partitioning for Scalable State Machine Replication

Appendix: Correctness

ACM Reference Format:
. 2018. DynaStar: Optimized Dynamic Partitioning for, Scalable State Machine Replication: Appendix: Correctness. In *Proceedings of ACM Conference (Conference'17).* ACM, New York, NY, USA, 1 page. https://doi.org/10.1145/nnnnnnn.nnnnnnn

Correctness criterion

DynaStar ensures linearizable executions. An execution is *linearizable* if there is a way to reorder the client commands in a sequence that (i) respects the semantics of the commands, as defined in their sequential specifications, and (ii) respects the real-time precedence of commands.

Correctness proof

To prove that DynaStar ensures linearizability, we must show that for any execution σ of the system, there is a total order π on client commands that (i) respects the semantics of the commands, as defined in their sequential specifications, and (ii) respects the real-time precedence of commands (§). Let π be a total order of operations in σ that respects $<$, the order atomic multicast induces on commands.

To argue that π respects the semantics of commands, let C_i be the i-th command in π and p a process in partition $\mathcal{P}_p$ that executes C_i. We claim that when p executes C_i, it has updated values of variables in $vars(C_i)$, the variables accessed by C_i. We prove the claim by induction on i. The base step trivially holds from the fact that variables are initialized correctly. Let $v \in vars(C_i)$, C_v be the last client command before C_i in π that accesses v, and q a process in $\mathcal{P}_q$ that executes C_v. From the inductive hypothesis, q has an updated value of v when it executes C_v. There are two cases to consider: (a) $p = q$. In this case, p obviously has an updated value of v when it executes C_i since no other command accesses v between C_v and C_i. (b) $p \neq q$. Since processes in the same partition execute the same commands, it must be that $\mathcal{P}_p \neq \mathcal{P}_q$. From the algorithm, when q executes C_v, $v \in \mathcal{P}_q$ and when p executes C_i, $v \in \mathcal{P}_p$. Thus, q executed a command to move v to another partition after executing C_v and p executed a command to move v to $\mathcal{P}_p$ before executing C_i. Since there is no command that accesses v between C_v and C_i in π, q has an updated v when it executes C_v (from inductive hypothesis), and p receives the value of v at q, it follows that p has an updated v when it executes C_i.

We now argue that there is a total order π that respects the real-time precedence of commands in σ. Assume C_i ends before C_j starts, or more precisely, the time C_i ends at a client is smaller than the time C_j starts at a client, $t^{cli}_{end}(C_i) < t^{cli}_{start}(C_j)$. Since the time C_i ends at the server from which the client receives the response for C_i is smaller than the time C_i ends at the client, $t^{srv}_{end}(C_i) < t^{cli}_{end}(C_i)$, and the time C_j starts at the client is smaller than the time C_j starts at the first server, $t^{cli}_{start}(C_j) < t^{srv}_{start}(C_j)$, we conclude that $t^{srv}_{end}(C_i) < t^{srv}_{start}(C_j)$.

We must show that either $C_i < C_j$; or neither $C_i < C_j$ nor $C_j < C_i$. For a contradiction, assume that $C_j < C_i$ and let C_j be executed by partition $\mathcal{P}_j$.

There are two cases:

(a) C_i is a client command executed by $\mathcal{P}_j$. In this case, since C_i only starts after C_j at a server, it follows that $t^{srv}_{end}(C_j) < t^{srv}_{start}(C_i)$, a contradiction.

(b) C_i is a client command executed by $\mathcal{P}_i$ that first involves a move of variables $vars$ from $\mathcal{P}_j$ to $\mathcal{P}_i$. At $\mathcal{P}_j$, $t^{srv}_{end}(C_j) < t^{srv}_{start}(global(vars, \mathcal{P}_j, \mathcal{P}_i))$ since the move is only executed after C_j ends. Since the move only finishes after variables in $vars$ are in $\mathcal{P}_i$ and C_i can be executed, it must be that $t^{srv}_{end}(global(vars, \mathcal{P}_j, \mathcal{P}_i)) < t^{srv}_{start}(C_i)$. We conclude that $t^{srv}_{end}(C_j) < t^{srv}_{start}(C_i)$, a contradiction.

Therefore, either $C_i < C_j$ and from the definition of π, C_i precedes C_j or neither $C_i < C_j$ nor $C_j < C_i$, and there is a total order in which C_i precedes C_j.

For termination, we argue that every correct client eventually receives a response for every command C that it issues. This assumes that every partition (including the oracle partition) is always operational, despite the failure of some servers in the partition. For a contradiction, assume that some correct client submits a command C that is not executed. Atomic multicast ensures that C is delivered by the involved partition. Therefore, C is delivered at a partition that does not contain all the variables needed by C. As a consequence, the client retries with the oracle, which moves all variables to a single partition and requests the destination partition to execute C, a contradiction that concludes our argument.