A Review of 'All about Eve: Execute-Verify Replication for Multi-Core Servers'

Prashanth R Duggirala

Summary

Paper by Manos Kapritsos, Yang Wang, Vivien Quema, Allen Clement, Lorenzo Alvisi and Mike Dahlin, 2012.

This paper presents a new architecture that allows state machine replication to scale to multi-core servers. Fault-tolerant execution on multi-core servers poses a new challenge, even for deterministic applications, because thread-level parallelism may introduce unpredictable differences between processes, if different servers interleave requests' instructions in different ways, the states and outputs of correct servers may diverge even if no faults occur. Eve refers to the Execute-Verify nature of the architecture. It departs from the traditional agree-execute architecture of state machine replication where replicas first execute groups of requests concurrently and then verify that they can reach an agreement on a state and output produced by a correct replica; if they can not, they roll back. Eve speculatively identifies groups i.e. batches of non-interfering operations and executes each batch in parallel. Afterwards it compares and verifies the outputs if they can reach an agreement on a state and output is produced by a correct replica. If too many replicas diverge so that a correct state/output cannot be identified, Eve guarantees safety and liveness by rolling back and sequentially and deterministically re-executing the requests. In Traditional architectures, deterministic replicas first agree on the order in which requests are to be executed and then execute them. Eve's performance depends on mechanisms that ensure that, despite the nondeterminism introduced by al-lowing parallel execution, replicas seldom diverge, and that, if they do, divergence is efficiently detected and reconciled. Eve minimizes divergence through a mixer stage that applies application-specific criteria to produce groups of requests that are unlikely to interfere, and it makes repair efficient through incremental state transfer and fine-grained rollbacks.

Eve consists of two stages, execution stage and verification stage. In execution stage, eve divides requests in batches, and lets replicas execute requests within a batch in parallel. Clients send their requests to the current pri-mary execution replica. The primary groups requests into batches, assigns each batch a sequence number,and sends them to all execution replicas. Each replica runs the same deterministic mixer algorithm to partition each batch received from the primary into the same ordered sequence of requests that the mixer believes can be executed in parallel with little likelihood that different interleavings will produce diverging results at distinct replicas. Each replica executes the parallel batches in the order specified by the mixer. In the verification stage, an agreement protocol is run to determine the final state and outputs of all correct replicas after each batch of requests. The main differences between PBFT and Eve are, in PBFT the replicas try to agree on the output of a single node, the primary but In Eve the object of agreement is the behavior of the execution replicas. Therefore, in Eve verification replicas use a quorum of tokens from the execution replicas as their "proposed" value. Also, in PBFT the replicas try to agree on the inputs to the state machine (the incoming requests and their order). Instead, in Eve replicas try to agree on the outputs of the state machine (the application state and the responses to the clients).

Strong points

1. By exploiting its unique ability to allow independent replicas to interleave requests non-deterministically, eve gains performance by avoiding the overhead of enforcing determinism.

2. Eve divides requests in batches, and lets replicas exe-cute requests within a batch in parallel, without requiring them to agree on the order of request execution within a batch. However, Eve takes steps to make it likely that replicas will produce identical final states and outputs for each

batch. Eve uses also uses a dynamic batching scheme to optimize the trade-off between latency and throughput.

3. The issues of using deterministic execution of multithreaded programs have been explained by the paper. Current techniques for deterministic multithreading either require hardware support or are too slow for production environments and the second issue is the semantic gap between modern SMR protocols and the techniques used to achieve deterministic multithreading.

Weak points

1. The algorithm relies heavily on the quality of the mixer, Eve is very sensitive to inaccurate mixers.

2. The experiments and benchmarks mentioned in the paper, no rollbacks were observed, but in the real world case there might be more cases of rollbacks and the performance improvements may not be as pronounced.

3. The algorithm works under the assumption that at any given point in time, the requests at each replica are same, which is not quite possible in a real world scenarios.

Comments expanding upon weak points

First, In the paper, it was mentioned that when they add false negatives while doing benchmarks, the number of rollbacks increases with both the underlying conflict rate and the false negative rate and the impact builds more quickly than one might expect because there is essentially a birthday "paradox"—if we have a 1% conflict rate and a 1% false negative rate, then the probability that any pair of conflicting requests be misclassified is 1 in 10000. But in a batch of 100 requests, each of these requests has about a 1% chance of being a party to a conflict, which means there is about a 39% chance that a batch of 100 requests contain an undetected conflict. Second, The paper also mentions the usage of H2 Database and TPC-W workload, nothing about their nature has been mentioned in the paper, this casts a doubt in my mind about the veracity of the performance improvements showcased by the paper. Third, In real world scenarios, it is highly likely that due to how the replicas are connected over the network, the requests may not be reaching each replica at the same time, but the paper assumes that the requests at each replica and their order is the same. This presents the requirement of some mechanism which ensures that the requests at every replica is the same at every point in time.

References:

Non-Determinism in Byzantine Fault-Tolerant Replication

High performance recovery for parallel state machine replication