EV: Replicating Multithreaded Servers

Manos Kapritsos*, Yang Wang*, Vivien Quema†,
Allen Clement‡, Lorenzo Alvisi*, Mike Dahlin*
*University of Texas at Austin †Grenoble INP ‡MPI-SWS

Abstract: This poster presents EV, a new Execute-Verify architecture that allows state machine replication to scale to multi-core servers. EV departs from the traditional agree-execute architecture of state machine replication: replicas first concurrently and nondeterministically execute requests; then they verify, agree, and converge on the state and the outputs produced by a correct replica. EV minimizes divergence through a mixer stage that applies application-specific rules to organize requests into batches of requests that are unlikely to interfere. Our evaluation suggests that EV's unique ability to combine execution independence with nondeterminism enables high-performance replication for multi-core servers while offering tolerance to a wide range of faults, including elusive concurrency bugs.

1 Motivation

State machine replication is a powerful fault-tolerance technique [3, 10, 16]. Historically, the essential idea is for replicas to deterministically process the same sequence of requests so that they traverse the same sequence of internal states and produce the same sequence of outputs.

Multi-core servers pose a challenge to this approach. To take advantage of parallel hardware, modern servers execute multiple requests in parallel. However, if different servers interleave requests' instructions in different ways, the servers' states and outputs may diverge. As a result, most state machine replication systems implement sequential execution where a replica finishes executing one request before beginning to execute the next [3, 11, 12, 17, 18, 20].

At first glance, a promising idea is to leverage recent efforts to enforce deterministic parallel execution. Unfortunately, existing techniques may require rewriting applications around new synchronization abstractions [2, 14] or may incur high overheads [6, 7, 19].

The EV replication architecture allows parallel request execution with low overhead. EV does this by eliminating the requirement that execution be deterministic. Instead, EV allows execution to be internally nondeterministic, but it speculates that the results of parallel execution (including the system's important state [15] and outputs) will match across replicas.

To execute nondeterministically without violating the safety requirement of replica coordination, EV turns on its head the tenet of traditional state machine replication: traditionally, deterministic replicas first *agree* on the order in which requests are to be executed and then *execute* them [3, 11, 12, 16, 20]. In EV, replicas first speculatively *execute* requests concurrently and nondeterministically; they then *verify* and agree on the state and the outputs produced by a correct replica. If replicas diverge, EV guarantees safety and liveness by rolling back and sequentially and deterministically re-executing the requests.

Critical to EV's performance are mechanisms that ensure that despite nondeterminism, replicas seldom diverge, and that, if they do, divergence is efficiently detected and reconciled through state-transfer and fine-grained rollbacks. EV minimizes divergence through a *mixer* stage that applies application-specific rules to organize requests into batches of requests that are unlikely to interfere. Note that if the underlying program is correct under unreplicated parallel execution, then delaying agreement until after execution and falling back on sequential re-execution guarantees that replication remains safe and live even if the mixer allows interfering requests in the same batch.

EV's execute-verify architecture also allows us to use replication to protect against nondeterministic concurrency bugs ("Heisenbugs" [8].) EV thus explores a region of the design space that falls short of Byzantine fault tolerance but that strengthens guarantees compared to standard crash-tolerance. EV's robustness stems from two sources. First, EV's *mixer* reduces the likelihood of triggering latent concurrency bugs by running only unlikely-to-interfere requests in parallel [9, 13]. Second, EV's *execute-verify* architecture allows it to detect and recover from cases where concurrency causes executions to diverge regardless of whether the divergence was the result of different legal choices by different replicas or was caused by a Heisenbug.

This poster refines the fundamental assumptions of state machine replication. For decades, the traditional view has been that if all replicas execute independently, then producing the same sequences of outputs to a set of clients requires them to be deterministic [10, 16]. EV leverages the observation that although deterministic

execution of a sequence of inputs is sufficient to produce identical outputs across replicas, it is not a fundamental requirement to maintaining execution independence.

2 Result summary

The practical consequence of refining our assumptions is that EV can use its execute-verify architecture to bring together *nondeterminism* and *independence* to improve the replication of multi-core servers:

1. *Nondeterminism → EV provides high-performance replication of multi-core servers.* EV gains performance by avoiding the overhead of enforcing determinism. For example, in our experiments with the TPC-W benchmark, with 16 cores EV achieves a 7.2*x* throughput speedup for synchronous, crash-tolerant server-pair replication, while the original unreplicated server achieves a 8.8*x* throughput speedup. Additionally, in our experiments EV has competitive overheads and better scalability than the Remus primary-backup system [5], and EV outperforms DPG [1], a replication system that handles multi-core servers by ensuring determinism across executions.

2. *Independence → EV masks a wide range of faults.* EV's architecture is general, and our prototype supports tunable fault tolerance [4], retaining state machine replication's ability to be configured to tolerate crash, omission, or Byzantine faults. Notably, even when an EV system is configured to tolerate crash or omission failures, it can also mask some concurrency failures. Although we do not claim that our experimental results are general, we find them promising: for the TPC-W benchmark running on the H2 database, executing requests in parallel on an unreplicated server triggered a previously undiagnosed concurrency bug in H2 73 times in a span of 750K requests. Under EV, our mixer *eliminated* all manifestations of this bug. Furthermore, when we altered our mixer to allow batches of conflicting requests, EV detected and corrected this bug 82% of the times it manifested.

References

[1] T. Bergan, N. Hunt, L. Ceze, and S. D. Gribble. Deterministic process groups in dOS. In *OSDI*, 2010.

[2] E. D. Berger, T. Yang, T. Liu, and G. Novark. Grace: safe multithreaded programming for C/C++. In *OOPSLA*, 2009.

[3] M. Castro and B. Liskov. Practical Byzantine fault tolerance and proactive recovery. *ACM Trans. Comput. Syst.*, 2002.

[4] A. Clement, M. Kapritsos, S. Lee, Y. Wang, L. Alvisi, M. Dahlin, and T. Riche. Upright cluster services. In *SOSP*, 2009.

[5] B. Cully, G. Lefebvre, D. Meyer, M. Feeley, N. Hutchinson, and A. Warfield. Remus: High availability via asynchronous virtual machine replication. In *NSDI*, 2008.

[6] J. Devietti, B. Lucia, L. Ceze, and M. Oskin. DMP: deterministic shared memory multiprocessing. In *ASPLOS*, 2009.

[7] G. Dunlap, D. Lucchetti, M. Fetterman, and P. Chen. Execution replay for multiprocessor virtual machines. In *VEE*, 2008.

[8] J. Gray. Why do computers stop and what can be done about it. Technical Report 85.7, Tandem Computers, June 1985.

[9] R. Kotla and M. Dahlin. High throughput Byzantine fault tolerance. In *DSN*, 2004.

[10] L. Lamport. Time, clocks, and the ordering of events in a distributed system. *CACM*, 1978.

[11] L. Lamport. The part-time parliament. *ACM Trans. Comput. Syst.*, 1998.

[12] Y. Mao, F. P. Junqueira, and K. Marzullo. Mencius: building efficient replicated state machines for WANs. In *OSDI*, 2008.

[13] C. Reis, J. Dunagan, H. Wang, O. Dubrovsky, and S. Esmeir. BrowserShield: Vulnerability-driven filtering of dynamic HTML. In *OSDI*, 2006.

[14] J. Robert L. Bocchino, V. S. Adve, D. Dig, S. V. Adve, S. Heumann, R. Komuravelli, J. Overbey, P. Simmons, H. Sung, and M. Vakilian. A type and effect system for deterministic parallel Java. In *OOPSLA*, 2009.

[15] R. Rodrigues, M. Castro, and B. Liskov. BASE: using abstraction to improve fault tolerance. In *SOSP*, Oct. 2001.

[16] F. B. Schneider. Implementing fault-tolerant services using the state machine approach: a tutorial. *ACM Computing Surveys*, 1990.

[17] A. Singh, P. Fonseca, P. Kuznetsov, R. Rodrigues, and P. Maniatis. Zeno: Eventually consistent Byzantine-fault tolerance. In *NSDI*, 2009.

[18] R. van Renesse, K. P. Birman, and S. Maffeis. Horus: A flexible group communication system. *CACM*, 1996.

[19] K. Veeraraghavan, D. Lee, B. Wester, J. Ouyang, P. Chen, J. Flinn, and S. Narayanasamy. DoublePlay: parallelizing sequential logging and replay. In *ASPLOS*, page 15, 2011.

[20] J. Yin, J.-P. Martin, A. Venkataramani, L. Alvisi, and M. Dahlin. Separating agreement from execution for Byzantine fault tolerant services. In *SOSP*, Oct. 2003.