Brief Announcement: Coded State Machine - Scaling State Machine Execution under Byzantine Faults

Songze Li
University of Southern California
songzeli@usc.edu

Saeid Sahraei
University of Southern California
ss_805@usc.edu

Mingchao Yu
University of Southern California
mingchay@usc.edu

Salman Avestimehr
University of Southern California
avestimehr@ee.usc.edu

Sreeram Kannan
University of Washington
ksreeram@ee.washington.edu

Pramod Viswanath
University of Illinois at
Urbana-Champaign
pramodv@illinois.edu

ABSTRACT

We introduce Coded State Machine (CSM), an information-theoretic framework to securely and efficiently execute multiple state machines on Byzantine nodes. The standard method of solving this problem is using State Machine Replication, which achieves high security at the cost of low efficiency. CSM simultaneously achieves the optimal linear scaling in storage, throughput, and security with increasing network size. The storage is scaled via the design of Lagrange coded states and coded input commands that require the same storage size as their origins. The computational efficiency is scaled using a novel delegation algorithm, called INTERMIX, which is an information-theoretically verifiable matrix-vector multiplication algorithm of independent interest.

KEYWORDS

state machine execution, Byzantine faults, security, computational efficiency, information theory, verifiable computing

ACM Reference Format:

Songze Li, Saeid Sahraei, Mingchao Yu, Salman Avestimehr, Sreeram Kannan, and Pramod Viswanath. 2019. Brief Announcement: Coded State Machine - Scaling State Machine Execution under Byzantine Faults. In *2019 ACM Symposium on Principles of Distributed Computing (PODC '19)*, July 29-August 2, 2019, Toronto, ON, Canada. ACM, New York, NY, USA, 3 pages. <https://doi.org/10.1145/3293611.3331573>

1 INTRODUCTION

We consider the problem of securely implementing K independent and identical state machines over a network of $N \geq K$ untrusted compute nodes, some of which are subject to Byzantine faults. For such a system, there are three critical performance metrics: 1) security β , defined as the maximum number of malicious nodes that can be tolerated; 2) storage efficiency γ , defined as the total number of state machines that can be supported given that each node has a storage size of a single state; and 3) throughput λ , defined as the total number of input commands processed per unit time.

Permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for third-party components of this work must be honored. For all other uses, contact the owner/author(s).

PODC '19, July 29-August 2, 2019, Toronto, ON, Canada

© 2019 Copyright held by the owner/author(s).

ACM ISBN 978-1-4503-6217-7/19/07.

<https://doi.org/10.1145/3293611.3331573>

Information theoretically, all three metrics scale at most linearly with the network size N .

The canonical approach to deal with faulty nodes/processors is state machine replication (SMR) (see, e.g., [1, 5, 12, 13]). For *full replication*, all K state machines are replicated at all N nodes. This achieves a security of $\beta_{\text{full}} = \frac{N}{2}$, but a storage efficiency of $\gamma_{\text{full}} = 1$, and a throughput of $\lambda_{\text{full}} = \frac{1}{c(f)}$, where $c(f)$ is the computational complexity of the state transition function f . Alternatively, for *partial replication*, distinct state machines are replicated at a different disjoint subset of $q = N/K$ nodes. Partial replication improves the storage efficiency and throughput to $\gamma_{\text{partial}} = K$ and $\lambda_{\text{partial}} = \frac{K}{c(f)}$, which scale linearly with N by letting $K = \Theta(N)$. However, the security drops to $\beta_{\text{partial}} = \frac{q}{2}$. A key theoretical question in this context is whether this tradeoff between efficiency and security is fundamental or can be circumvented by careful algorithm design.

Table 1: Performance comparison of proposed CSM with SMR and information-theoretic limits in synchronous networks. Here $0 \leq \mu < \frac{1}{2}$ is some constant that denotes the fraction of malicious nodes. The state transition function f is an arbitrary multivariate polynomial with constant degree d . $c(f)$ and $c(\text{coding})$ are the computational complexities of f and the coding operations per node in CSM respectively.

	Security	Storage efficiency	Throughput
Full Replication	$\frac{N}{2}$	1	$\frac{1}{c(f)}$
Partial Replication	$\frac{N}{2K}$	K	$\frac{K}{c(f)}$
Information-Theoretic Limit	$\frac{N}{2}$	N	$\frac{N}{c(f)}$
Coded State Machine (CSM)	μN	$(1 - 2\mu)N/d + 1 - 1/d$	$\frac{(1-2\mu)N/d + 1 - 1/d}{c(f) + c(\text{coding})}$

Our Contributions. We propose “Coded State Machine” (CSM), which *simultaneously* achieves linear scaling for both storage efficiency and security, for both synchronous and partially synchronous networks (see Table 1). CSM also achieves an almost linear scaling for throughput under synchronous networks through the development of an interactive matrix-vector multiplication verification protocol named INTERMIX (specifically, INTERMIX helps to reduce the coding complexity per node $c(\text{coding})$ in Table 1 to $O(\log^2 N \log \log N)$).

Related Works. Conventional SMR research focuses on designing optimal protocols for the consensus phase (where the nodes aim to agree on a log of input commands submitted by clients) that tolerate the maximum number of faulty processors [5, 8, 13, 14, 19].

Concepts from coding theory have been utilized earlier to provide fault tolerance to state machine execution. Fused state machine is proposed in [2], where given n primary finite state machines, f fusion replicas are constructed to correct up to $\frac{f}{2}$ Byzantine faults, using closed partition lattice [15] of the reachable product state of the primary machines. Compared with CSM, while the fused state machine also achieves security scaling, the storage size at each replica increases with n , as opposed to a constant storage size in CSM. When the state machine operations are write and read, erasure codes have been exploited to minimize the storage costs given the coexistence of multiple versions of the data object, while ensuring that a reader can decode an atomically consistent version [3, 4, 11, 22].

The coding design of CSM is inspired by recent developments in *coded computing* [7, 16–18, 21, 23, 24], which leverage tools from coding theory to inject computation redundancy in order to improve latency, security, and privacy. In particular, as similarly done in Lagrange Coded Computing [23] to create coded data batches, CSM creates and processes coded states/commands using the Lagrange polynomial. However, in contrast to one-shot computation on static data in [23], the requirement of dynamically updating the local coded state that is compatible with the upcoming state transition poses new challenges in the design of CSM.

Compared to the existing verifiable matrix-vector multiplication schemes [9, 10, 20, 25], INTERMIX has two advantages. First, it is information-theoretically secure, i.e., no computational limitation assumptions are made on the nodes. Second, INTERMIX enables almost every node in the network (with the exception of a constant number of auditors) to verify the correctness of the results in constant time. In contrast, in the existing schemes the complexity of the verifier is at least linear in the size of the output.

2 MODEL

We consider operating K independent state machines with the same state transition function f on $N \geq K$ unreliable nodes. At round t , each machine k has state $S_k(t)$, and executes an input command $X_k(t)$ by computing $(S_k(t+1), Y_k(t)) = f(S_k(t), X_k(t))$, where $Y_k(t)$ is the output, and $S_k(t+1)$ is the state in next round.

Each state machine maintains a group of clients who continuously and concurrently submit their commands to be processed by broadcasting them to all nodes. For the communication between the nodes, we study two settings:

Synchronous network, with a fixed and *known* upper bound on the communication latency between any pair of nodes.

Partially synchronous network, with unbounded communication delay until an *unknown* global stabilization time (GST), after which the network becomes synchronous.

At round t , each node i locally stores some (possibly coded) data $Z_i(t)$, which is generated from the states $S_1(t), \dots, S_K(t)$.

Consensus Phase. For each t , all network nodes run some consensus protocol, via exchanging messages, to reach an agreement on a set of input commands $X_1(t), \dots, X_K(t)$ to process (the consensus setting is standard in the literature, e.g., [6]).

Execution Phase. Each node i computes some intermediate result from its stored data $Z_i(t)$, and the commands $X_1(t), \dots, X_K(t)$ agreed in the consensus phase, and communicates it to other nodes. Using the local computation result, and the results received from

Figure 1: Illustration of Coded State Machine. (a) Coded State: node i stores a coded state $\tilde{S}_i$. **(b) Coded Execution:** node i computes an intermediate result $f(\tilde{S}_i, \tilde{X}_i)$, which is given to a Reed-Solomon decoder to recover output results.

others, each node i recovers the state transition functions for a subset of machines, returns the recovered outputs to the intended clients, and updates its local storage to $Z_i(t+1)$.

Network nodes may be subject to *authenticated Byzantine faults*. That is, a faulty node can exhibit arbitrary behaviors that deviate from the above described protocol, but message impersonation is easily detectable.

We evaluate a state machine computation scheme P using the following performance metrics (defined in Section 1): security (β_P), storage efficiency (γ_P), and throughput (λ_P). Here we focus on the regime where computation latency is dominated by operations in the execution phase.

3 CODED STATE MACHINE ALGORITHM

CSM consists of two components: *Coded State* and *Coded Execution*.

Coded State. CSM views a (possibly coded) state variable as evaluating a polynomial $u(z)$ at a point. As illustrated in Figure 1(a), having specified $u(z)$ using the uncoded states $S_1, \dots, S_K$ via Lagrange interpolation at distinct points $\omega_1, \dots, \omega_K$, CSM generates N coded states $\tilde{S}_1, \dots, \tilde{S}_N$, by evaluating $u(z)$ at distinct points $\alpha_1, \dots, \alpha_N$, and stores $\tilde{S}_i$ at node i . Since each coded state has the same size as an uncoded state, the CSM scheme has a storage efficiency of $\gamma_{\text{CSM}} = K$.

Coded Execution. The consensus phase is performed the same as conventional SMR. In the execution phase, each node i generates a coded command $\tilde{X}_i$ from the agreed commands $X_1, \dots, X_K$ as done for the state encoding, and directly feeds $\tilde{S}_i$ and $\tilde{X}_i$ into the state transition function f . For any multivariate polynomial f , $f(\tilde{S}_i, \tilde{X}_i)$ can be viewed as evaluating a composite polynomial h of higher degree at the point α_i (see Figure 1(b)). From the intermediate computation results $h(\alpha_1), \dots, h(\alpha_N)$, a subset of which may be erroneous, CSM exploits efficient noisy interpolation techniques

like Reed-Solomon decoding to recover h , and evaluates it at ω_k to obtain the output $f(S_k, X_k) = h(\omega_k)$, for all $k = 1, \dots, K$. When μ fraction of the nodes are malicious, CSM can securely support up to $K = \frac{(1-2\mu)N-1}{d} + 1$ machines in a synchronous network, and up to $K = \frac{(1-3\mu)N-1}{d} + 1$ machines in a partially synchronous network.

Using CSM, both the storage efficiency and the security improves linearly with the network size N .

4 SCALING THROUGHPUT VIA DELEGATED VERIFIABLE COMPUTING

The throughput of CSM does not naturally scale with the network size N . This is because that the encoding and decoding, performed separately at each node, incur computational complexities that scale with N . Observing that polynomial interpolation and multi-point polynomial evaluation, as the building blocks of the CSM coding, are special matrix multiplications, we develop an information-theoretic protocol named INTERMIX that securely delegate the multiplication of a matrix A and a vector X to a single worker node, where efficient algorithms can be applied to reduce overall complexity.

As the first step of INTERMIX, a worker and a constant number of auditors are chosen randomly from the N nodes. Each auditor repeats the computation of $Y = AX$ and compares it with the result returned by the worker. As detailed in Algorithm 1, an (honest) auditor who detects a fraud will interactively enforce the worker to produce an inconsistent result which can be checked in constant time by the remaining nodes.

Algorithm 1 Algorithm run by an honest auditor

Input: $A, X, \hat{Y}$.
Output: One bit indicating whether the auditing process has succeeded or failed. A string ζ that localizes the failure.

```

1: if  $\hat{Y} = Y$  then
2:   Return True.
3: end if
4: Choose  $i$  such that  $\hat{Y}_i \neq Y_i$ . Set  $\zeta = [i]$ .
5: Set  $j = 1$ ,  $A^{(1)} = A_i$ ,  $X^{(1)} = X$ ,  $\hat{Y}^{(1)} = \hat{Y}_i$ ,  $s = \text{length}(A^{(1)})$ .
6: while  $(s > 1)$  do
7:   Let  $A^{(j,1)} = A_{1:\lfloor s/2 \rfloor}^{(j)}$ ,  $A^{(j,2)} = A_{\lfloor s/2 \rfloor+1:s}^{(j)}$ ,  $X^{(j,1)} = X_{1:\lfloor s/2 \rfloor}^{(j)}$ ,
      $X^{(j,2)} = X_{\lfloor s/2 \rfloor+1:s}^{(j)}$ .
8:   Request  $Y^{(j,1)} = A^{(j,1)}X^{(j,1)}$  and  $Y^{(j,2)} = A^{(j,2)}X^{(j,2)}$  from
     the worker. Receive  $\hat{Y}^{(j,1)}$ ,  $\hat{Y}^{(j,2)}$ .
9:   if  $\hat{Y}^{(j,1)} + \hat{Y}^{(j,2)} \neq \hat{Y}^{(j)}$  then
10:    Return (False,  $\zeta$ ).
11:  else
12:    Choose  $\ell \in \{1, 2\}$  such that  $\hat{Y}^{(j,\ell)} \neq Y^{(j,\ell)}$ . Let  $s =$ 
       $\text{length}(Y^{(j,\ell)})$  and  $\zeta = [\zeta, \ell]$ .
13:    Let  $Y^{(j+1)} = Y^{(j,\ell)}$ ,  $\hat{Y}^{(j+1)} = \hat{Y}^{(j,\ell)}$ ,  $A^{(j+1)} = A^{(j,\ell)}$  and
       $X^{(j+1)} = X^{(j,\ell)}$ . Increment  $j$ .
14:  end if
15: end while
16: Return (False,  $\zeta$ ).

```

Using INTERMIX, the coding complexity at each node reduces to $O(N \log^2 N \log \log N)$, enabling CSM to achieve throughput $\lambda_{\text{CSM}} = \Theta\left(\frac{N}{\log^2 N \log \log N}\right)$, which scales almost linearly with N .

ACKNOWLEDGMENTS

This work is supported by Defense Advanced Research Projects Agency under Contract No. HR001117C0053, NSF grants CCF-1703575, CCF-1763673, and CCF-1705007, ONR award No. N00014-16-1-2189, ARO awards W911NF1810400 and W911NF1810332, Distributed Technologies Research, and SNSF grant 178309. The views, opinions, and/or findings expressed are those of the author(s) and should not be interpreted as representing the official views or policies of the Department of Defense or the U.S. Government.

REFERENCES

- [1] Peter A Alsberg and John D Day. 1976. A principle for resilient sharing of distributed resources. In *Proceedings of the 2nd international conference on Software engineering*. IEEE Computer Society Press, 562–570.
- [2] Bharath Balasubramanian and Vijay K Garg. 2014. Fault tolerance in distributed systems using fused state machines. *Distributed computing* 27, 4 (2014), 287–311.
- [3] Christian Cachin and Stefano Tessaro. 2006. Optimal resilience for erasure-coded Byzantine distributed storage. In *IEEE DSN'06*. 115–124.
- [4] Viveck R Cadambe, Zhiying Wang, and Nancy Lynch. 2016. Information-theoretic lower bounds on the storage cost of shared memory emulation. In *Proceedings of the 2016 ACM Symposium on Principles of Distributed Computing*. ACM, 305–313.
- [5] Miguel Castro and Barbara Liskov. 2002. Practical Byzantine fault tolerance and proactive recovery. *ACM TOCS* 20, 4 (2002), 398–461.
- [6] Danny Dolev, Cynthia Dwork, and Larry Stockmeyer. 1987. On the minimal synchronism needed for distributed consensus. *JACM* 34, 1 (1987), 77–97.
- [7] Sanghamitra Dutta, Viveck Cadambe, and Pulkit Grover. 2016. Short-dot: Computing large linear transforms distributedly using coded short dot products. In *NIPS*. 2100–2108.
- [8] Cynthia Dwork, Nancy Lynch, and Larry Stockmeyer. 1988. Consensus in the presence of partial synchrony. *JACM* 35, 2 (1988), 288–323.
- [9] Kaoutar Elkhiyaoui, Melek Önen, Monir Azraoui, and Refik Molva. 2016. Efficient techniques for publicly verifiable delegation of computation. In *ASIA CCS*.
- [10] Dario Fiore and Rosario Gennaro. 2012. Publicly verifiable delegation of large polynomials and matrix computations, with applications. In *Proceedings of the 2012 ACM conference on Computer and communications security*. ACM, 501–512.
- [11] James Hendricks, Gregory R Ganger, and Michael K Reiter. 2007. Low-overhead byzantine fault-tolerant storage. In *ACM SIGOPS Operating Systems Review*, Vol. 41. ACM, 73–86.
- [12] Leslie Lamport. 1978. The implementation of reliable distributed multiprocess systems. *Computer networks* 2, 2 (1978), 95–114.
- [13] Leslie Lamport. 1998. The part-time parliament. *ACM TOCS* 16, 2 (1998), 133–169.
- [14] Leslie Lamport, Robert Shostak, and Marshall Pease. 1982. The Byzantine generals problem. *ACM TOPLAS* 4, 3 (1982), 382–401.
- [15] David Lee and Mihalis Yannakakis. 2002. Closed partition lattice and machine decomposition. *IEEE Trans. Comput.* 51, 2 (2002), 216–228.
- [16] Kangwook Lee, Maximilian Lam, Ramtin Pedarsani, Dimitris Papailiopoulos, and Kannan Ramchandran. 2018. Speeding up distributed machine learning using codes. *IEEE Transactions on Information Theory* 64, 3 (2018), 1514–1529.
- [17] Songze Li, Mohammad Ali Maddah-Ali, and A Salman Avestimehr. 2016. A Unified Coding Framework for Distributed Computing with Straggling Servers. *IEEE Workshop on Network Coding and Applications* (Sept. 2016).
- [18] S. Li, M. A. Maddah-Ali, Q. Yu, and A. S. Avestimehr. 2018. A Fundamental Tradeoff Between Computation and Communication in Distributed Computing. *IEEE Transactions on Information Theory* 64, 1 (Jan. 2018).
- [19] Brian M Oki and Barbara H Liskov. 1988. Viewstamped replication: A new primary copy method to support highly-available distributed systems. In *Proceedings of the seventh annual ACM Symposium on Principles of distributed computing*. 8–17.
- [20] Saeid Sahraei and A Salman Avestimehr. 2019. INTERPOL: Information Theoretically Verifiable Polynomial Evaluation. *arXiv preprint arXiv:1901.03379* (2019).
- [21] Rashish Tandon, Qi Lei, Alexandros G. Dimakis, and Nikos Karampatziakis. 2017. Gradient Coding: Avoiding Stragglers in Distributed Learning. In *Proceedings of the 34th International Conference on Machine Learning*. 3368–3376.
- [22] Zhiying Wang and Viveck R Cadambe. 2018. Multi-Version Coding-An Information-Theoretic Perspective of Consistent Distributed Storage. *IEEE Transactions on Information Theory* 64, 6 (2018), 4540–4561.
- [23] Qian Yu, Songze Li, Netanel Raviv, Seyed Mohammadreza Mousavi Kalan, Mahdi Soltanolkotabi, and A Salman Avestimehr. 2019. Lagrange Coded Computing: Optimal Design for Resiliency, Security, and Privacy. In *AISTATS*.
- [24] Qian Yu, Mohammad Ali Maddah-Ali, and A Salman Avestimehr. 2017. Polynomial Codes: an Optimal Design for High-Dimensional Coded Matrix Multiplication. In *NIPS*. 4406–4416.
- [25] Yihua Zhang and Marina Blanton. 2014. Efficient secure and verifiable outsourcing of matrix multiplications. In *International Conference on Information Security*. Springer, 158–178.