Research Statement

Shuai Mu
Department of Computer Science
Stony Brook University (SBU)

"A distributed system is one in which the failure of a computer you didn't even know existed can render your own computer unusable. "

Leslie Lamport

1 Summary

I work in distributed systems, in particular distributed database systems, which are the foundation of many large-scale online services. As a distributed systems person, the first lesson I learned and now give in distributed systems classes is: do not use a distributed system if the problem fits into a single machine, because distributed systems are hard to build, use, and maintain. However, distributed systems are the necessary solution when the data scale increases. Almost every programmer today needs to interact with distributed systems. How to make distributed systems more usable is a prominent question.

My entire research career is driven by this goal: how to make distributed systems as easy to use as a single machine? I hope programmers use distributed systems in an extremely simple way, as if writing a single threaded program, while enjoying all the benefits such as scalability and performance. The major challenge I find in achieving this goal is the *data consistency* issue brought up by failures and concurrency. There are different solutions to the data consistency issue, e.g., adopting a relaxed data consistency for easier designs in the cost of exposing eccentric behaviors to users. My approach takes no compromises—the user observes zero data inconsistency. My research advocates for *linearizable* replication and *(strictly-)serializable* transaction, which hide all the failure and concurrency details from the users and provide a strongly consistent view.

If we see the data consistency issue as a trade-off between correctness and performance, choosing the strongest data consistency means facing more challenges in building a high-performance system. To that end, a large portion of my research works focus on achieving a higher performance in linearizable (serializable) systems. I have explored a range of approaches, including 1) adopting SmartNIC network acceleration (§2.1), 2) utilizing modern multi-core machines (§2.2), 3) designing more efficient distributed protocols (§2.3). In addition to the performance works, I am also excited to explore other aspects in this theme, including: 4) implementing distributed protocols in practice (§2.4), 5) auditing black-box serializability (§2.5).

Executive Summary.

- I have published 16 conference, 2 workshop papers, over 60% of which were published after I joined Stony Brook in 2018. At the time of writing, my works have been cited over 750 times, with an h-index of 13.
- My "home" community is (distributed) systems, i.e., I publish mostly in conferences such as SOSP/OSDI/NSDI/EuroSys/ATC. I have actively contributed to the success of these conferences by serving on their TPCs on multiple occasions.
- I have helped secure, as PI or Co-PI, more than $7.5 Million in research funding across 6 grants, including the NSF Career Award. Of these, I am PI on 4 of them, and about $1.7 Million is budgeted for my exclusive use.

2 Recent Works

2.1 Accelerating Distributed Systems with FPGA SmartNICs

Linearizable replication is usually achieved by state machine replication (SMR) backed by a consensus protocol. To provide a transparent replication to users, the implementation of the SMR needs to achieve ultra-low latency, in microseconds. Traditional implementations are limited by the kernel/userspace stack as well as commodity network devices. We tackle this challenge by applying FPGA SmartNICs, using them to accelerate an SMR system with the Raft consensus protocol. FPGA is powerful in performance: it is capable to provide single-digit microsecond latency at line rate and is insensitive to the amount of offered load. But it is also notorious for the difficulty in programming. For example, in our experience writing a line of code for FPGA can take 10-100 times more effort than software C++ code.

Therefore, our approach does not implement the entire SMR system in hardware; instead, it is a hybrid software/hardware system. We name the system Waverunner (as opposed to Raft). We make the observation that, despite the complexity of SMR, the most common routine—the data replication—is actually simple. The complex parts (leader election, failure recovery, etc.) are rarely used in modern datacenters where failures are only occasional. These complex routines are not performance critical; their software implementations are fast enough and do not need acceleration. Therefore, our system uses FPGA assistance to accelerate data replication, and leaves the rest to the traditional software implementation of SMR. Our Waverunner approach is beneficial in both the common and the rare case situations. In the common case, the system runs at the speed of the network, with a 99th percentile latency of 1.8 µs achieved without batching on minimum-size packets at network line rate (85.5 Gbps in our evaluation). In rare cases, to handle uncommon situations such as leader failure and failure recovery, the system uses traditional software to guarantee correctness, which is much easier to develop and maintain than hardware-based implementations. Overall, our experience confirms Waverunner as an effective and practical solution for hardware accelerated SMR—achieving most of the benefits of hardware acceleration with minimum added complexity and implementation effort.

Waverunner was published at NSDI '23 [1]. When building Waverunner, an interesting question is raised: can this hybrid approach be applied to more distributed system protocols? We try to answer this question by investing several widely-used types of distributed protocols, including distributed hash tables, consensus protocols, and Byzantine protocols. We find all these protocols share the character of only running a small part in the common routines, and therefore they are also fit for our hybrid acceleration approach. A deeper reason for this phenomenon is that all these distributed protocols need to consider the failure cases. Failure tolerance and failure recovery are typically very complex, and therefore they take a large portion in the design, but are not often run. This observation leads us to building a generalized framework for accelerating common distributed protocols (§3).

2.2 Building Speedy Multi-core Transactional Systems

Databases in recent years typically come in two flavors: distributed databases and single-machine multi-core databases. Compared to distributed databases, multi-core databases are relatively new, but tend to be faster and achieve higher throughput owing to the lack of network delay. I have worked on building fast multi-core databases since PhD/Postdoc [2, 3].

Despite being fast, multi-core databases have one weakness compared to (replicated) distributed database: they cannot continue operating in the presence of crash failures. Recent works, including our own Waverunner work, attempt to fix this by providing faster replication utilizing advanced network techniques such as RDMA or FPGA. However, these techniques have limitations: they require special hardware, and have special needs on the deployment, e.g., the machines have to be on the same or adjacent racks.

My latest work in this line tries to explore the following research question: can we significantly improve the throughput of replicated transactions (the bottleneck in making distributed databases perform as fast as multi-core ones) with a cleverer replication protocol between a multi-core leader and its multi-core replicas? Our solution, Rolis, aims to mask the high cost of replication by ensuring that cores are always doing useful work and not waiting for each other or for other replicas. Rolis achieves this by not mixing the multi-core concurrency control with multi-machine replication, as is traditionally done by systems that use Paxos to replicate the transaction commit protocol. Instead, Rolis takes an "execute-replicate-replay" approach. Rolis first speculatively executes the transaction on the leader machine, and then replicates the per-thread transaction log to the followers using a novel protocol that leverages independent Paxos instances to avoid coordination, while still allowing followers to safely replay. The execution, replication, and replay are carefully designed to be scalable and have nearly zero coordination overhead across cores. Our evaluation shows that Rolis can achieve 1.03M TPS (transactions per second) on the TPC-C workload, using a 3-replica setup where each server has 32 cores. This throughput result is orders of magnitude higher than the traditional software approaches we tested (e.g., 2PL), and is comparable to state-of-the-art, fault-tolerant, in-memory storage systems built using kernel bypass and advanced networking hardware, even though Rolis runs on commodity machines.

2.3 Designing More Efficient Distributed Protocols

Since my PhD time, I have worked on designing better concurrency control and consensus protocols, including a concurrency control Rococo that exploits workload semantics in ordering transactions [4], a consolidated protocol that combines consensus and concurrency control [5], and a latency-optimal protocol for read-only transactions [6]. My most recent work on this line aims to reduce the cost for achieving strict serializability in many common cases. Although strict serializability can greatly simplify application development, only a few concurrency control techniques can provide strict serializability and they are expensive. Common techniques include two-phase locking, optimistic concurrency control, and transaction (re-)ordering. They incur high overheads which manifest in extra rounds of messages, distributed lock management, blocking, and excessive aborts.

We have observed that these overheads are unnecessary for many datacenter workloads where transactions do not interleave: e.g., many of them are dominated by reads, and the interleaving of reads that return the same value does not affect correctness. In many cases, the transactions arrive at servers in an order that trivially satisfies the strict-serializability requirement. We call these transactions *naturally consistent*. Therefore, we developed Natural Concurrency Control (NCC), a new concurrency control technique that guarantees strict serializability and ensures minimal costs—i.e., one-round latency, lock-free, and non-blocking execution—in the common case. NCC's design insight is to execute naturally consistent transactions in the order they arrive, as if they were non-transactional operations, while guaranteeing correctness without interfering with transaction execution.

NCC has three key designs: 1) Non-blocking execution ensures that servers execute transactions in a way that is similar to executing non-transactional operations. 2) Decoupled response management separates the execution of requests from the sending of their responses, ensuring that only correct results are returned. 3) Timestamp-based consistency checking uses timestamps to verify transactions' results, without interfering with execution. We compare NCC with common strictly serializable techniques: OCC, 2PL, and TR, and several non-strict serializable protocols. With read-dominated workloads from Google and Facebook, NCC significantly outperforms 2PL, OCC, and TR with 2-10× lower latency and 2-20× higher throughput, and closely matches the performance of a non-strict serializable protocol. NCC was published at OSDI '23 [7].

While designing the consistency-checking component, we identified a correctness pitfall in timestamp-based, strictly serializable techniques. That is, these techniques sometimes fail to guard against an

execution order that is total but incorrectly inverts the real-time ordering between transactions, thus violating strict serializability. We call this the timestamp-inversion pitfall. Timestamp inversion is subtle because it can happen only if a transaction interleaves with a set of non-conflicting transactions that have real-time order relationships. The pitfall is fundamental as we find it affects several prior systems, which, as a result, do not provide strict serializability as claimed.

2.4 Implementing Distributed Protocols in Practice

Starting from 2014, I had frequent discussions with the lead of MongoDB's replication team. They needed to implement a consensus-based replication module to replace the old ad-hoc replication implementation. This is a rare experience as it reveals the real-world challenges in implementing a consensus protocol.

For MongoDB, the biggest challenge in this process is backward compatibility. In its older versions, MongoDB made the design choice of using client query interface for transmitting request logs between replicas. That is, the leader stores all request logs in a special database table, and a follower directly queries from this table to retrieve the latest log. This is very different from a common distributed database design where a leader would "push" logs to followers by initiating messages; hence we name MongoDB's approach as a "pull"-based model. The pull-based model makes the old replication implementation very simple (e.g., no need for adding message handlers) and gives flexibility to its performance (e.g., a follower can pull from another follower, following chain-style replication).

Our solution started with adopting the recent and popular Raft, but ended up basically inventing a new protocol with many heavy modifications. During our development we found that any unthoughtful changes to the protocol would easily introduce new corner cases that would break the correctness of the system. To ensure correctness, extensive verification and testing are done on the protocol, including model checking using TLA+, fuzz testing, and fault-injection testing. The developed protocol preserves the feature of allowing data pulling between any two replicas. This work was published at NSDI '23 [8].

2.5 Auditing Black-box Strict Serializability

Similar to other infrastructure services, databases are moving to the cloud. Cloud databases often offer serializability guarantees. But cloud databases are complicated black boxes, running in a different administrative domain from the clients. Thus, clients might like to know whether the databases are meeting their contract. To that end, we developed Cobra, aiming to audit the serializability of a black-box database.

Cobra is the first system that combines (a) black-box checking, of (b) serializability, while (c) scaling to real-world online transactional processing workloads. The core technical challenge is that the underlying search problem is computationally expensive. Cobra tames that problem by starting with a suitable SMT solver. It then introduces several new techniques, including a new encoding of the validity condition; hardware acceleration to prune inputs to the solver; and a transaction segmentation mechanism that enables scaling and garbage collection. Cobra imposes modest overhead on clients, improves over baselines by 10× in audit cost, and (unlike the baselines) supports continuous auditing. Our artifact can handle 2000 transactions/sec, equivalent to 170M transactions/day. Cobra was published at OSDI '20 [9].

Based on Cobra's work, we are developing a more generic framework for auditing a wide range of practical databases. Practical databases also offer other isolation levels, e.g., snapshot isolation (SI). There is not yet a fast checker that can audit SI soundly. We developed Viper to solve this issue, as our first step to the more generic framework. Viper introduced a few new techniques, including a new representation of dependency graph we call BC-graph, and an optimization to accelerate checking SI by leveraging heuristics. Viper is the first checker that can audit all major SI variants soundly, and improves over baselines by 15×. Viper was published at EuroSys '23 [10].

3 Future Research Directions

Automatic FPGA acceleration for distributed system protocols. As discussed above, we find that distributed protocols expose a common characteristic that the mostly common run code only takes a small part in the whole system. Therefore, we are investigating a more general approach that can be applied to a rich body of distributed protocols. A prior work iPipe (Liu et al. SIGCOMM '19) has demonstrated this possibility via building a general actor framework with ARM-based SmartNICs. Users can write the code once and the framework will move the code between the host and the NIC dynamically at runtime. This work shares a similar goal but unfortunately its method cannot be applied to the FPGA world because FPGA has a completely different architecture from the host. Therefore, we plan to take a different approach. We provide an RPC-like interface for the users, which the users are already familiar with (e.g., gRPC). Additionally, the users need to annotate some functions. We will construct a compiler to compile these functions into code that can be further compiled into our FPGA hardware. The system will also build abstractions for shared data to bridge the host and the FPGA. This way the user can develop the system as if writing a normal piece of software, and the framework will automatically accelerate the system with FPGA SmartNICs.

Hybrid automatic verification of distributed systems. Distributed systems and protocols are complex and bug-prone. How to find and reduce the bugs in distributed systems is an important problem and is pursued by many researchers. The ultimate solution to this problem is through formal verification. A formally verified system provides mechanically checkable proof that the implementation exactly meets its specification. However, writing the implementation and the proof in a formal verification language (e.g., Coq, Dafny) takes a lot more effort than write common C++ code. This is very similar to the fact that FPGA code is hard to write in the hardware acceleration project. Therefore, we attempt to apply our observation again in the formal verification field, and purse a hybrid approach for a distributed protocol. For the complex but rarely run part, we write an automatic translation tool that can translate from specification to (slow) implementation with little human intervention. For the straightforward and commonly run part, we manually optimize the implementation and write the proof. Some preliminary results show this direction is promising. It can reduce more than 80% of human effort while preserve most of the system's performance in the common case.

Scalable distributed multi-core transactions. A major limitation of Rolis is that it does not support sharding so it cannot scale to different machines. We are exploring this question: can we support distributed (and replicated) transaction while preserving the performance of the multi-core machines? Decoupling concurrency control and replication in a distributed setting turns out to be more challenging than we initially thought in our preliminary exploration. In Rolis, because there is only a single shard, the failure of the leader leads to a follower taking over. Since the follower takes over, we do not to concern about the residual state on the failed leader because it is considered failed and outvoted by the rest of the replicas. If we extend the system design to directly support sharding, then the leader expands to the leaders group, and machines in this group speculatively execute transactions before the transaction logs are replicated. If any machine in the group fails, the system has to drop all the machines in the group because the system cannot recover a single node failure; it has to switch to a group of new leaders. This will halt the entire system for seconds to minutes, and it is not acceptable in a large-scale deployment. Addressing this problem is the primary challenge in future work on this project.

References

[1] Mohammadreza Alimadadi, Yida Wu, Sergey Madaminov, Shenghsun Cho, Michael Ferdman, Peter Milder, and Shuai Mu. Waverunner: An elegant approach to hardware acceleration of state machine replication. In *Proceedings of USENIX Conference on Networked Systems Design and Implementation (NSDI)*, April 2023.

[2] Zhaoguo Wang, Shuai Mu, Yang Cui, Han Yi, Haibo Chen, and Jinyang Li. Scaling multicore databases via constrained parallel execution. In *Proceedings of ACM International Conference on Management of Data (SIGMOD)*, June 2016.

[3] Shuai Mu, Sebastian Angel, and Dennis Shasha. Deferred runtime pipelining for contentious multicore software transactions. In *Proceedings of ACM European Conference on Computer Systems (EuroSys)*, March 2019.

[4] Shuai Mu, Yang Cui, Yang Zhang, Wyatt Lloyd, and Jinyang Li. Extracting more concurrency from distributed transactions. In *Proceedings of USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, October 2014.

[5] Shuai Mu, Lamont Nelson, Wyatt Lloyd, and Jinyang Li. Consolidating concurrency control and consensus for commits under conflicts. In *Proceedings of USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, November 2016.

[6] Haonan Lu, Christopher Hodsdon, Khiem Ngo, Shuai Mu, and Wyatt Lloyd. The SNOW theorem and latency-optimal read-only transactions. In *Proceedings of USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, November 2016.

[7] Haonan Lu, Shuai Mu, Wyatt Lloyd, and Siddhartha Sen. NCC: Natural concurrency control for strictly serializable datastores by avoiding the timestamp-inversion pitfall. In *Proceedings of USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, July 2023.

[8] Siyuan Zhou and Shuai Mu. Fault-tolerant replication with pull-based consensus in MongoDB. In *Proceedings of USENIX Conference on Networked Systems Design and Implementation (NSDI)*, February 2021.

[9] Cheng Tan, Changgeng Zhao, Shuai Mu, and Michael Walfish. Cobra: Making transactional key-value stores verifiably serializable. In *Proceedings of USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, November 2020.

[10] Jian Zhang, Ye Ji, Shuai Mu, and Cheng Tan. Viper: A fast snapshot isolation checker. In *Proceedings of ACM European Conference on Computer Systems (EuroSys)*, May 2023.