Research Proposal: Advancing Fault-Tolerant Distributed Systems through Speculative Execution

Weihai Shen, PhD student at Stony Brook University

My research focuses on **speedy and fault-tolerant distributed database systems**, which form the backbone of many large-scale online services. These systems face a persistent challenge: the ever-growing volume of data continuously imposes heavier workloads and increasing demands on system **performance**. This scale requires the infrastructure systems to support very high throughput. Meanwhile, **strong fault tolerance and rapid failure recovery** are also required and crucial in those infrastructure systems, especially for data-sensitive services, where multiple nodes (e.g., hundreds of thousands of shards in Spanner) perform transaction execution or store data coordinately, and various types of failures can cause some nodes to slow down or fail resulting in incomplete transaction execution or data loss. Strong fault-tolerance and fast failure recovery can ensure the infrastructure system (e.g., Google Cloud) is able to continue proceeding promptly without correctness issues in case of a minority of nodes crashing. Quorum-based replication was the prevalent approach for strong fault-tolerance. However, it requires a considerable network communication latency which increases the contention footprint resulting in poor performance, for example, Google's Spanner (Corbett et al., OSDI'12) using Paxos to replicate each step in transaction execution can only achieve several thousand transactions per second on the online transaction processing workload. Recent optimizations, like COCO (Yi et al., VLDB'21), improve throughput via group commit but a single shard failure would halt the entire system.

The tradeoff between performance and (strict) correctness is an eternal topic in fault-tolerant system design and implementation. In chasing performance, systems from academia and industry choose to (1) give up correctness properties, such as *serializable transactions and linearizable replication*; (2) compromise strong fault-tolerance, such as the multi-core database system on the single node only with logging support (e.g., Silo, Tu et al., SOSP'13) or (3) depend on kernel bypass and advanced network hardware (e.g., Meerkat, Szekeres, Eurosys'20) or (4) block the entire system during failures.

Digging into the root reason for the poor performance in replicated distributed systems, servers are mostly idle due to waiting for the coordination between partitions (e.g., two-phase commit) and replication among replicas (e.g., Paxos) to complete, but which should be avoided completely. Although the top-level observation about the idle waits is easy to state, it was highly non-trivial to eliminate this wait. For example, the protocol in Spanner requires that the system has to wait for committing one transaction until all coordination and replication are done.

My approach differs from the conventional designs: I am pursuing performance without giving up on strong fault-tolerance and fast failure recovery. The main philosophy is to explore the possible concurrency and pipelining chances left on the table by exisiting designs. In particular, my research uses *asynchronous replication* and *vectorized method* to avoid idle waits while maximizing the pipeline to mask the cost of distributed coordination protocols without causing more aborts and blocking.

Recent Research and Vision

Maximizing the performance. Over the last decade, the number of CPU cores on a server has increased significantly (from 8 to several hundred). A large number of past works focus on optimizing concurrency control protocol to build multi-core transactional systems on a single node. The throughput of a single-node database can be easily over 1M transactions per second on the online transaction

processing workload. However, single-node databases consider improving reliability by supporting checkpoints to disks and recovery from them after rebooting from a crash. This provides a weaker fault-tolerance guarantee than a replication-based solution chosen by academia and industry.

Strong fault-tolerance is achieved at expense of sacrificing performance. One straightforward way to mitigate this performance loss is having more concurrent transactions. In hope, the long latency in the distributed coordination and replication protocols is roughly masked and the system can achieve similar throughput to a non-replicated transactional system. However, this approach can't increase the throughput easily. The main reason for this is that the replication protocol is closely tied to the transaction execution and commit protocols (e.g., Spanner as mentioned above) which will cause more aborts by increasing the chance of conflicting accesses.

Our approach, Rolis [2, 1], aims at masking 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. In more detail, Rolis first speculatively executes all transactions on the leader. Then, each thread of the leader creates an independent Paxos stream and uses it to replicate its transaction logs *asynchronously*, including the serialization order, to a corresponding thread at each follower. Finally, the followers deterministically replay the logs to arrive at the same state. The execution, replication, and replay are carefully designed to be scalable and serializable. Our evaluation shows that Rolis can achieve similar throughput to the single-node non-replicated databases while having strong fault-tolerance same as replicated counterparts.

Rolis targets a multi-core transaction on a single shard. In reality, a partitioned database is preferred. For example, Google's Spanner is able to support hundreds of thousands of shards. Spanner applies Paxos to replicate the critical steps in two-phase locking and two-phase commit; increasing the number of outstanding transactions will not improve the system's performance, but will instead cause more aborts by increasing the chance of conflicting accesses. Note that Spanner can work perfectly only when transactions in many applications are not conflicted with each other and well-partitioned. To optimize the performance, COCO tries to optimize throughput by grouping transactions into epochs and only persisting transactions at the epoch boundaries. While throughput is pretty high, the downside is that the entire system can be blocked in two cases: (1) every 10ms, during the 2PC synchronization needed for epoch durability and epoch advancement; and (2) if there is a shard failure. Our solution, Warbler, utilizes vector watermark to decouple replication, which is similar to COCO, but further decouple each shard's impact as much as possible. This allows many new transactions to execute without interruption from the failed shard. Also, under normal conditions, there's no 2PC coordination needed in Warbler for advancing epochs.

Making systems programming more accessible. Building correct, high-performance systems are known to be hard and error-prone. A major issue is that the protocols in systems lead to non-deterministic state changes and complex branching conditions based on different events (e.g., time-outs). Traditionally, these systems are built with an asynchronous coding style with event-driven callbacks but often lead to "callback hell" that makes code hard to follow and maintain. Converting to synchronous coding styles (e.g., using coroutines) is challenging because of the complex branching conditions. Our work, Depfast [3], is an expressive framework for developing various systems. DepFast provides a set of event abstraction (e.g. *QuorumEvent* or *TimeoutEvent*), to enable building systems in a synchronous style. With Depfast, the researchers or engineers can code a system in a synchronous programming style which is more natural and less error-prone without losing the performance.

Future Research Directions

Speculation (pipelining) is a powerful principle in system design, and we have identified a recurring pattern in how it is leveraged across many systems. My near-term goal is to **distill this pattern**

into a reusable abstraction that enables correct and seamless speculation optimization in diverse systems.. First, we will craft a general, pluggable design that can be applied uniformly to a wide spectrum of systems—ranging from serverless platforms to disk-based consensus protocols and two-phase commit (2PC) in distributed databases. Second, we aim to elevate this abstraction into the programming model itself, so that speculative execution becomes completely transparent to developers. DepFast is only the first step in this direction, and we intend to push the idea much further. These considerations introduce new opportunities in protocol and system design to boost performance and accessibility further which I plan to pursue in my future research.

References

[1] **W. Shen**, Y. Cui, S. Angel, S. Sen, and S. Mu. Warbler: Speculative distributed transactions with geo-replication. In 19th USENIX Symposium on Operating Systems Design and Implementation (OSDI 25), 2025.

[2] **W. Shen**, A. Khanna, S. Angel, S. Sen, and S. Mu. Rolis: a software approach to efficiently replicating multi-core transactions. In Proceedings of the Seventeenth European Conference on Computer Systems (Eurosys), 2022.

[3] X. Luo, **W. Shen**, S. Mu, and T. Xu. Depfast: Orchestrating code of quorum systems. In 2022 USENIX Annual Technical Conference (USENIX ATC), 2022.