Research Statement

Jialin Li

The unprecedented scale of today’s datacenter applications presents tremendous challenges to the design of systems infrastructure. These systems need to handle the massive scale of user traffic, remain highly available despite failures, keep data strongly consistent, and meet stringent performance requirements.

My research focuses on building distributed systems that offer strong semantics and high performance. In particular, my research has pioneered a new approach to designing distributed systems: co-designing distributed systems with the datacenter network. Using this approach, I have built strongly consistent replication systems and distributed transactional systems with dramatic performance improvement, and distributed storage systems with provable load balancing guarantees.

Co-designing distributed systems with datacenter networks calls for a careful separation of logic between the network layer and the application layer. To that end, I analyze the core protocols in these systems, and identify key abstractions that 1) can be efficiently implemented in network devices, given the limited hardware resources and restricted computational model, and 2) can result in significant end-to-end performance improvements. As an example, in the NOPaxos work we built an ordering abstraction using network sequencing, and co-designed a new replication protocol around it to largely eliminate the coordination overhead of consensus.

1 Current Work

To meet the increasing demand of datacenter applications, distributed storage systems are partitioned for scalability and replicated for availability. To provide strong data consistency, these systems run a number of complex distributed protocols on each user request – consensus protocol such as Paxos for replication, two-phase commit protocol for atomic commitment, and two-phase locking for isolation. Each protocol involves extensive coordination among servers, limiting the overall system scalability and imposing high latency overhead.

My research addresses the above challenges by co-designing distributed systems with the datacenter network. Taking advantage of new generation programmable switches, we have built several novel network-level primitives that offer strong guarantees. We then leveraged these primitives in the protocol and system design, leading to dramatic reduction in coordination overhead for state machine replication and distributed transactional systems. These projects have already had impact: Cavium adapted their switch architecture to support NOPaxos-style sequencing; Google and Microsoft are currently looking into NOPaxos, and Microsoft and Huawei are working on their own implementations of Eris.

1.1 Co-Designing State Machine Replication with the Datacenter Network: Speculative Paxos and NOPaxos

Distributed protocols like Paxos and Viewstamped Replication require extensive coordination due to their assumption of a completely asynchronous network where packets can be arbitrarily delayed, dropped, or reordered. Although these are reasonable assumptions about the Internet, datacenter networks are more predictable, reliable, extensible, and programmable.

We exploited these properties of the datacenter network in Speculative Paxos [10] and designed a new network level primitive, the Mostly-Ordered Multicast. Mostly-Ordered Multicast...
provides a best-effort guarantee that all multicast receivers will receive messages in the same order with high probability. We efficiently engineered this network primitive by leveraging the network topology and flexible packet forwarding enabled by software-defined networking.

We then co-designed a new replication protocol, Speculative Paxos, which relies on the ordering property of Mostly-Ordered Multicast. Replicas assume requests are ordered in the common case, and speculatively execute operations without any explicit coordination. This allows the protocol to commit client requests with minimum latency and eliminates leader bottleneck. The resulting system achieves 40% lower latency and 2.6× higher throughput than leader-based Paxos.

In Network-Ordered Paxos [6], we took this approach a step further. Instead of a best-effort ordering property, we designed a stronger network primitive, the Ordered Unreliable Multicast (OUM), which guarantees message ordering but may drop messages. NOPaxos replicas then only need to agree on which messages are dropped, but not the order of messages. We achieved a near-zero-overhead implementation of OUM by building a sequencer directly in the data plane of a programmable switch ASIC.

NOPaxos not only outperforms Speculative Paxos and other replication protocols, but more importantly, it achieves throughput within 2% and latency within 16µs of an unreplicated system, demonstrating that there need not be a trade-off between strong consistency and maximum performance.

1.2 Coordination-Free Consistent Distributed Transactions: Eris

In Eris [5], we looked beyond state machine replication, and considered distributed transactions in sharded storage systems. To provide both data consistency and fault tolerance, these systems commonly use a layered approach: a replication protocol within each shard, an atomic commitment protocol and a concurrency control protocol across shards. As each protocol requires explicit coordination, committing a single client transaction in these systems involves multiple rounds of coordination, limiting the overall system performance.

Eris takes a new approach to distributed transaction processing. We leveraged properties of an important class of transactions called independent transactions, and designed a network multi-sequencing layer to consistently order independent transactions both within and across shards. Eris then uses this multi-sequencing layer to build a unified replication and transactional protocol that avoids both intra-shard and cross-shard coordination. As a result, despite offering strong transactionality, consistency, and fault tolerance, Eris achieves performance within 3% of a non-transactional, unreplicated system on the TPC-C benchmark.

1.3 Dynamic Load Balancing for Distributed Storage Systems: Pegasus

Distributed storage systems face the challenge of highly skewed and dynamic workloads. Popular data items often receive orders of magnitude higher traffic than others, and data popularity changes constantly. The resulting server load imbalance can lead to a significant drop in system throughput and higher tail latency.

In Pegasus [7], we used data replication to provide dynamic load balancing for distributed storage systems. Our key observation is that selectively replicating a small number of popular objects can achieve a surprisingly high degree of load balancing. Realizing selective replication, however, is a challenging task: it requires tracking the set of replicated objects and which servers they are stored on, as well as guaranteeing data consistency. Leveraging the programmability in reconfigurable switches, Pegasus introduces a novel solution that implements an in-network coherence directory directly in the switch data plane. Pegasus leverages the ToR switch’s central view of request traffic to dynamically replicate popular objects based on workload changes and performs load-aware forwarding based on real-time server load information.
The resulting system shows significant load balancing improvements: compared to consistent hashing, Pegasus reduces the tail latency for highly skewed workloads by up to 97%, and increases the throughput by 9\times. Pegasus is also able to achieve these benefits across a large set of workloads with different read/write ratios and skewness, while reacting quickly to object popularity changes.

### 1.4 High Performance Operating Systems Design for the Datacenter

The strict performance SLOs of datacenter applications present new challenges to the design of datacenter operating systems. To ensure fast responses, servers need to provide both low median latency and low tail latency. However, server applications today incur significant OS kernel overhead for packet processing. At the same time, current operating systems offer poor latency predictability: the 99th percentile latencies can be 1000\times higher than the median [8].

Our study [8] explored the hardware, operating system, and application-level sources of poor tail latency. We used classical queueing theory models to establish ideal latency distributions, and compared them to actual latency measurements of networked services. Our results showed that these services running on Linux exhibit much worse tail latency than the queueing model predicts. Using fine-grained instrumentations, we systematically identified causes of excessive tail latency. These findings led us to apply various system level techniques, resulting in tail latency improvement by two orders of magnitude.

In Arrakis [9], we explored a new operating system design that moves the kernel out of the I/O data plane while still maintaining the same security and isolation guarantees of a traditional OS. Leveraging recent I/O virtualization technologies, Arrakis allocates protected virtual device instances directly to applications. Application I/O operations then require no kernel intervention, and Arrakis provides coarse-grained control plane interfaces to manage these virtual instances. Our new OS design results in a throughput improvement of 3\times for Memcached and 9\times for Redis.

### 2 Future Directions

Recent network, hardware, and application trends in datacenters have introduced many new challenges – and exciting opportunities – in systems research. I will continue to work on leveraging new hardware and application trends to build systems with strong semantics and high performance. More concretely, I’m looking at the following two potential future research directions.

#### 2.1 Resource Disaggregation in Datacenters

Resource disaggregation has lately drawn interest from both industry and academia [11, 2, 4, 3]. Moving away from the server-centric architecture offers many benefits, including more flexible resource allocation and better resource utilization, independent scaling of hardware resources based on application requirements, and stronger fault tolerance against individual component failures.

Many challenges, however, still remain to be addressed. Even with network technology advances, accessing remote resources will still be inherently slower than local accesses due to fundamental physical constraints. To maximize performance, it is therefore crucial to be able to exploit application locality of reference. I plan to conduct measurement studies of popular datacenter applications and analyze their workloads, access patterns, and working set. I will use findings in these studies to design better caching mechanisms for the disaggregation system such as workload-aware prefetching, more optimal eviction policies, and better cache memory allocation schemes.
To deliver acceptable performance, many datacenter applications such as Memcached, GraphLab, and Spark require remote resource access latency to be consistently below 5µs \[2\]. Any non-trivial queuing delay in the network and overhead in the end-host network stack would lead to severe performance degradation. I plan to design new transport protocols and congestion control mechanisms that achieve low end-to-end median and tail latency. Towards this goal, I will study network traffic workloads in the disaggregated datacenter environment, and use queueing theory models to derive latency distributions for different scheduling policies in switches and end-hosts.

To enable transparent distribution of compute resources across physical machine boundaries, cache memory on compute nodes needs to be kept coherent and consistent. Traditionally, providing cache coherence in a distributed setting has been considered impractical due to the high cost of running distributed coherence protocols. Co-designing the coherence protocol with programmable network hardware has the potential to close this gap. Towards this goal, I plan to design an in-network coherence protocol that minimizes latency and server processing overhead. To address the issue of limited on-chip memory on programmable switches, I plan to distribute coherence directory entries to the compute node FPGA NICs, which have access to larger on-board DRAM.

2.2 System Software for Programmable Hardware

Over the last few years, various forms of programmable accelerators have been widely deployed in datacenters, including GPUs for massively parallel computations, TPUs for deep neural networks, FPGAs for general-purpose accelerations, and reconfigurable switches for network packet processing. Comparing to general-purpose CPUs, programmable accelerators offer higher energy efficiency, massive degree of parallelism, and acceleration for specific classes of computation (e.g. matrix multiplication). As the performance improvement in CPUs has largely stalled, we will likely to see even more adoptions of heterogeneous, domain-specific hardware in the future.

Despite their wide deployment, system software support for these hardware is still rudimentary, making resource sharing and multiprogramming a formidable task in a multi-tenancy environment. For instance, Barefoot Tofino \[12\] only supports running a single P4 program at a time; Amazon EC2 F1 \[1\] instances assign the entire FPGA fabric to a single user FPGA image; Google’s TPU has a simple, deterministic execution model, with no support for multiprocessing and context switching.

Designing system software for programmable hardware calls for collaboration with researchers in hardware, architecture, programming languages, and compilers. Unlike general-purpose processors where resources are abstracted away by the ISA, these accelerators currently present low-level hardware-specific components to programmers and compilers. I hope to design common hardware abstraction layers and intermediate representations, such that system software can efficiently multiplex hardware resources in a fine-grained manner. I also plan to develop hardware-independent programming models for better portability, and co-design compiler techniques to generate hardware-specific optimizations based on hardware characteristics.
References


