Research Statement

Wyatt Lloyd

Vision

My research addresses emerging problems in the massive-scale distributed systems that support big data. The recent and dramatic growth in the demands on these systems—on their ability to store, process, and manage large data volumes—makes this area ripe for new research. This new scale, e.g., all of Twitter's tweets, requires massively distributed systems with hundreds or thousands or more machines cooperating to provide the necessary capacity and throughput.

To handle the magnitude of this data, recent systems from industry and academia have stressed performance and scalability at the cost of strong semantic properties, e.g., linearizability and transactions, yet these same properties make systems easier to use and understand. My research questions whether such sacrifices are necessary and seeks to identify properties that make these systems easier to program to and reason about that are compatible with massive scale. I pursue properties that are rigorously defined for the clarity they bring to programmers using the systems, e.g., causal consistency instead of eventual, or guaranteeing low latency for all operations.

Looking at my research from another direction, it reexamines conventional design decisions and approaches given the new reality of big data. Centralized approaches that were simple, straightforward, and effective in the small systems of the past quickly hit bottlenecks that prevent scaling to the large systems of today. In contrast, I design systems that provide strong properties while keeping all operations distributed, so they can scale to the even larger systems of tomorrow.

My focus on scalability is maintained throughout the research process; I include it as a primary design requirement and then build prototypes that are tested with real data at scale. While building and experimentally verifying that a design is scalable is useful and necessary in itself, I have also found that doing so often exposes unanticipated bottlenecks and can reveal important new research topics, as discussed in my future directions. This focus on designing and building massive-scale systems that provide strong, rigorous properties defines my niche as a researcher.

Dissertation Research

Geo-replicated storage systems provide the backend for massive-scale websites like Facebook, storing data that includes your profile, friends list, and status updates. These storage systems seek to provide an "always-on" experience where operations always complete quickly, because of a widely demonstrated link between page load times, user engagement, and revenue. We term systems that can handle such data at scale and provide an always-on experience as *ALPS systems*, because they provide four key properties: Availability, Low latency, Partition tolerance, and Scalability.

Previous ALPS systems such as Amazon's Dynamo, LinkedIn's Project Voldemort, and Facebook's Cassandra (in some configurations) all made large usability sacrifices in pursuit of their scale and performance goals. These sacrifices—such as providing only eventual consistency, which gives no ordering guarantees about operations—make it hard for programmers to reason about the system and result in an end-user experience that is far from ideal. My dissertation research shows that these sacrifices are not fundamental; stronger consistency and semantics are achievable for ALPS storage systems.

Yet, known theoretical results show that low latency and the strongest types of consistency are incompatible.[1] Knowing this, the recent spate of low-latency geo-replicated systems settled for the weakest semantic property, eventual consistency. The first part of my dissertation research, COPS, shows that this sacrifice is not necessary. COPS is the first ALPS system to provide causal consistency, a middle ground between the two extremes where operations always appear in an order consistent with potential causality, e.g., all of a user's operations and all conversations between users appear in their original order. This improvement in consistency makes the distributed storage more intuitive for programmers to use and gives end users more of the experience they expect.

One key technical contribution in COPS is a design focused on the scalability of clusters where the keyspace is partitioned across nodes, replication is done in parallel from all nodes in each cluster, and then nodes use dependency metadata associated with the updates to issue distributed checks that ensure operations are always applied in the correct causal order. Naively, dependency metadata grows exponentially and throttles performance. COPS avoids this problem using multiple types of garbage collection and by exploiting the transitive structure inherent in the graph of potential causality.

COPS's use of distributed metadata runs counter to the traditional wisdom for enforcing causal consistency, which was to exchange logs of operations and then replay them at other locations. The log-exchange approach works well and admits a simple implementation when all data can reside on a single machine. With the new realities of massive scale where data is spread across many machines, however, the log-exchange approach breaks down as logging updates to all machines in a cluster into one place becomes a bottleneck. In contrast, because of COPS's distributed design, it is the first scalable and causally consistent system.

The big data in ALPS systems is spread across thousands of nodes, yet previous systems provided only inconsistent batch operations to read and write data across multiple nodes. Eiger, the second part of my dissertation research, shows that much stronger semantics are possible. These stronger semantics include read-only and write-only transactions that consistently read or write data spread across all the nodes in a cluster. Eiger's semantics also include counter columns and the hierarchical column-family data model used in BigTable and Cassandra, which makes building applications atop it much simpler. The limited transactions and rich data model in Eiger do not come at the expense of high scalability or performance, and designing the system to ensure this was one of Eiger's main challenges.

In particular, to guarantee low latency Eiger must eschew locks and blocking, the typical techniques used for transactions. And to enable scaling it must avoid the centralization that is also typical for transactions. Eiger overcomes both these challenges using distributed algorithms that utilize logical-time-validity metadata and temporarily maintaining multiple versions of the data. In addition, its algorithms for read-only and write-only transactions are designed to work together using indirection to ensure that clients obtain a consistent, up-to-date view of the system and can atomically update data spread across many nodes.

My dissertation research shows that ALPS systems do not need to settle for eventual consistency and weak semantics. Taken together, Eiger and COPS show that causal consistency and stronger semantics are possible for low-latency geo-replicated storage.

[1]This incompatibility is an implication of Brewer's famed CAP theorem from 2000, which was formalized shortly after by Gilbert and Lynch. Its first proof, however, was a lesser-known result from 1988 by Lipton and Sandberg.

Future Directions

Fully-Distributed General Transactions

While working on transactional algorithms for the low-latency geo-replicated setting of COPS and Eiger, I began to design algorithms for fully-distributed general transactions. General transactions are widely recognized as easy for programmers to reason about and necessary for certain scenarios, e.g., banking. Due to the fundamental trade-off between strong consistency and low latency, they are hard to reason about in—and perhaps incompatible with—the geo-replicated and low-latency setting of COPS and Eiger. They are, however, compatible with a low-latency single-datacenter setting or with a non-low-latency geo-replicated setting.

Currently, transactions are either scalable or general, but not both. My research and Google's recent work on Spanner, which provides read-once-then-write transactions, are examples of the former. On the other hand, examples of the latter include traditional databases with general transactions that are restricted to a small subset (shard) of the data and/or are scheduled by a master node. Developing and verifying fully-distributed transactions across large numbers of nodes will bridge this divide and provide scalability for general transactions.

Scalable Transport for Massively-Distributed Systems

Current transport-layer protocols, e.g., TCP and UDP, are ill-suited for massive-scale distributed systems. TCP was designed to provide a reliable stream of data between two machines; UDP was designed to provide unreliable datagrams in the same setting. In contrast, the communication patterns of massive-scale distributed systems typically have large numbers of parallel, asynchronous RPCs between many machines.

One example of this mismatch is that while running experiments for Eiger with hundreds of nodes, I found I could not achieve perfect linear scaling of throughput as cluster size increased due to the overhead from the increasing number of TCP connections on each node. As another example, I've learned from industrial colleagues that it is common practice to aggregate connections from multiple processes on the same machine to reduce connection overhead and to increase batching. Yet another example is Facebook's modification to memcached that uses TCP for writes, but uses UDP for reads so they can build their own retransmission protocol atop it that is aware of how they batch reads (multigets).

All of these issues stem from a mismatch between what current transport layers provide and how massive-scale distributed systems communicate. While there has been much recent work on improving TCP for datacenter usage—e.g., DCTCP that improves congestion control, or the entire "Data Centers: Latency" session at SIGCOMM 2012 that improved flow completion times—a more fundamental approach is necessary and possible. Datacenter services provide a rare opportunity to deploy a clean-slate transport layer because they are in a single administrative domain and can be upgraded en masse. Along with networking colleagues, I am interested in exploring what new transport layer properties, abstractions, and mechanisms can better match the performance or programmability requirements of massive-scale distributed systems.

Making Programming Distributed Storage Make Sense

Strong consistency and general transactions, while incompatible with low-latency geo-replication, are well understood by programmers and easier for them to reason about than weaker consistency and limited transactions. My dissertation research starts to bridge this gap, but there is still much to do before programming massive-scale distributed storage truly "makes sense."

This introduces two exciting avenues of research. One is, how can we make it easier for programmers to reason about and use low-latency primitives? My current approach has been to make the primitives as strong as possible and I believe this direction will bear more fruit. But, we will also need ways to make it easier for programmers to express themselves. Perhaps a query language will help? Or, a compiler that will deconstruct general transactions into limited ones?

The other avenue of research is, how can we present a single interface that is simple for programmers to reason about that provides access to strong-but-slow general transactions and fast-but-weaker limited transactions? Should transactions and their results be typed so we can reason about their use throughout a program? Would a domain-specific language help? Can we allow programmers to write the simplest code initially and then only specialize it if dictated by performance?

Each of these avenues provides an interesting mix of programming languages and distributed systems problems. I look forward to collaborating on them with PL colleagues and I believe solving them will have a large and lasting impact on the way web services are built and the way programmers interact with big data.

In the last 20 years the field of distributed systems has changed dramatically. The field moved from systems on the order of 10s of nodes in one administrative domain, to peer-to-peer systems with 1000s of nodes in many administrative domains, to datacenter services with a small number of datacenters each with 1000s of nodes within it that are back in a single administrative domain. With billions of edge devices that are increasingly capable, I am intrigued to see how they fit into the distributed systems of the future.

Increasingly, distributed systems problems have connections to networking, databases, programming languages, algorithms, and security. I look forward to working with colleagues in these areas in my future research career, as I believe that many of the most productive types of research come from inter-area collaboration.