Research Statement

Peter Bailis

Today, large-scale data-intensive computing is more ubiquitous than ever. After decades of academic research, distributed system design and massive-scale data management are real concerns faced by everyday practitioners. Data-intensive services today now include billion-user, geo-replicated applications such as Facebook and Google. Coupled with the increasing interconnectivity of computing systems, from retail inventories to biosensors, and the availability of elastic cloud computing, these services have placed increased demands on today's data management solutions. Requirements for low latency, "always on," and scalable operation have led to a substantial restructuring of commodity database systems, resulting in a host of emerging research challenges. My research focuses on the core problems in these large-scale distributed data management systems.

In my dissertation research, I have investigated the design of coordination-free, asynchronous distributed database systems and algorithms. Specifically, when is it possible to achieve the benefits of coordination-free execution—including available, low latency, and scalable operation—and when is coordination required? Answering this question enables *coordination avoidance*: the use of as little coordination (i.e., blocking synchronization between concurrent operations [3]) as possible. When this coordination avoidance is possible, how can we realize its benefits, and what are the performance implications? Practically, coordination avoidance is a useful tool in designing systems and algorithms that circumvent the overheads found in traditional data management systems. For example, we can often avoid wide-area network delays that are lower-bounded by the speed of light and avoid the use of potentially expensive consensus and atomic commitment protocols, leading to regular order-of-magnitude performance gains.

I have investigated coordination avoidance in a number of systems contexts. As an example, prior research in data management has left a wide gap between easy-to-use but expensive mechanisms like serializable transactions and less understood but cheap and increasingly adopted approaches such as eventual consistency. One of my goals has been to understand when we can preserve the latency, scalability, and availability benefits of these weaker models (due to their lack of coordination) while providing database functionality and semantics traditionally associated with more coordination-intensive designs (e.g., transactional isolation [2], integrity constraints [3], indexes and views [7]). I have also studied the use of coordination avoidance in large-scale statistical and "Big Data" analytics in asynchronous distributed model training [10] and in building low-latency data product prediction pipelines [18]. Overall, this research demonstrates both the power and the limitations of coordination-free data management systems.

My Approach

I believe we are experiencing a renaissance in distributed data management: the number of problems that are both of practical interest and are fundamentally interesting as research questions is at an all-time high. In my research, I take industry's near-insatiable demand for more capable data management solutions as a prominent source of inspiration and collaboration. I accordingly favor a principled approach to studying and building practical data management systems that spans theory and practice. My approach to problem selection is heavily influenced by the practitioner community through personal interaction, meetups and talks, and social media. Although sifting through and distinguishing fundamental challenges from implementation issues requires a large sample size, the challenges I have gleaned from this community motivate every chapter of my dissertation research, from the widespread use of eventually consistent database deployments [9, 14] to prevailing weak isolation levels [2] and application-level invariants [3, 4]. This methodology rewards problems within the intersection of modern (or anticipated) practitioner concerns and open, challenging systems issues in data management research.

Having identified a problem in this intersection, I take a principled but systems-oriented approach to solving it. I typically seek an analytical model for system behavior, including upper bounds on performance and availability, often drawing from more theoretical literature (e.g., probabilistic quorums [14], rule-based rewrite systems [3], formal models of weak isolation [2]). I subsequently validate these models using software prototypes and then design and implement new algorithms and systems to fill the gap in functionality, performance, or both. I strongly believe that building and evaluating actual software artifacts is necessary to fully understand and evaluate systems solutions, and, incidentally, often leads to interesting new research. This frequently leads to the opportunity to contribute technology back to the open source community [7, 15]. Focusing on practice helps to identify the core research issues underlying demands on real-world data management infrastructure and is instrumental in building systems for a plausible future.

Large Scale Data Serving and Transaction Processing

The body of my dissertation research has focused on the problem of building low latency, scalable, and available distributed databases and transaction processing systems. Per above, given the rise of "weakly consistent" data storage, I have sought to understand when and why asynchronous, coordination-free data management solutions work in practice and when stronger solutions are required. This has led to a number of coordination-avoiding systems and algorithms that achieve useful functionality without sacrificing the benefits of coordination-free execution.

Probabilistically Bounded Staleness. While academia has developed a range of "strong" consistency guarantees such as serializability and linearizability, many industrial users of distributed databases today use a particularly weak, coordination-free model called eventual consistency. For the start of my dissertation studies, I wanted to understand why eventual consistency was so popular, despite its seeming lack of guarantees (i.e., lack of *safety* properties)—was it because industry was simply ignoring the ill effects of these models, or was eventual consistency somehow "good enough" in practice? I developed a technique called Probabilistically Bounded Staleness (PBS) that explains both how and why, in production, eventual consistency often delivers strong guarantees with low latency and without the use of coordination [14, 16, 17]. PBS provides a statistical *expectation* of safety in the form of staleness (w.r.t. regular register semantics) in terms of either real-time or versions using simple protocol analysis and Monte Carlo methods. Using PBS, I analyzed traces from LinkedIn and Yammer and found staleness was often limited to tens to a few hundred milliseconds, with meaningfully reduced tail latencies (especially with SSD-backed servers). I later extended PBS analysis to predict multi-key and causal consistency [17]. Accordingly, PBS provides some of the first quantitative evidence for *why* eventual consistency is so widely deployed, despite its weak guarantees. Our PBS implementation in Apache Cassandra was integrated into the 1.2.0 release [15].

Bolt-On Consistency. While PBS showed how eventually consistent stores often deliver consistent results, these stores still sometimes surface "inconsistent" behavior. Could we prevent this behavior without negating these stores' benefits, and could we do so without directly modifying existing systems? To answer these questions, I developed a thin *bolt-on* "shim" layer that "upgrades" the safety properties offered by existing stores while preserving their coordination-free benefits [9]. This allows re-use of the engineering efforts within existing data stores while allowing the shim designer to focus entirely on safety. My shim provides causal consistency, one of the strongest semantics that is still compatible with coordination-free execution. A core challenge in this work was encoding causal dependency information: using existing algorithms, an underlying eventually consistent store would "lose" history due to concurrent overwrites. The causal shim avoids this problem by using a novel encoding of each write's history (as a causally consistent cut). Across datacenters, this allows order-of-magnitude lower latencies than a strongly consistent configuration and comparable (sometimes lower) latency to eventual consistency—all without modifying the underlying store.

During this work, I also examined the use of application-provided causal dependencies (i.e., *explicit causality*) [5] in lieu of classic process-level causal dependencies. Via a study of several real-world applications, including Twitter reply chains, online blog comments, and bulletin board discussions, I demonstrated the potential for multiple order-of-magnitude reductions in causality graph size by tracking explicit causality, which I applied in the bolt-on shim.

Highly Available Transactions. After my study of coordination-free causal consistency, I began to wonder what other consistency semantics were achievable via coordination-free implementations. While transactional (or multi-item, multi-operation) guarantees are often associated with coordination-intensive, unavailable implementations (and, indeed, the standard textbook semantics, serializability, requires coordination), I decided to investigate whether this was a fundamental limitation. In a survey of 18 "ACID" and "NewSQL" databases, I found that only three offered serializability by default and only nine offered it as as an option at all. Instead, these databases offered weaker models [6]. When I further investigated these weaker models, I found that many were actually achievable under coordination-free execution (and thus as *Highly Available Transactions*, or HATs) [2]. For example, common isolation models such as Read Committed (default in eight databases) and ANSI Repeatable Read isolation are achievable as HATs. As intuition, many guarantees, like Read Committed, arbitrated *visibility* but not concurrency (much like causal consistency). A key challenge in this work was separating coordinated *implementations* (e.g., Read Committed via short read locks) from each model's semantics. The resulting HAT taxonomy is one of the first unified treatments of weak isolation, distributed register consistency, session guarantees, and coordination requirements in the literature. Using these results, I built a HAT prototype that achieves up to two order-of-magnitude latency reductions when deployed across datacenters.

RAMP Transactions. While performing the HAT analysis, I encountered a set of applications that were underserved by existing coordination-free algorithms. That is, while "strong" models like Repeatable Read and Snapshot Isolation are sufficient for enforcing foreign key integrity constraints, secondary indexing, and multi-get and multi-put operations,

there was no coordination-free mechanism for enforcing them. Accordingly, a number of practitioner reports on systems (e.g., from Facebook, Google, LinkedIn, and Yahoo!) specifically highlight these use cases as scenarios where, lacking a coordination-free algorithm, system architects explicitly sacrificed correctness for latency and availability.

In response, I proposed a new isolation model—called Read Atomic (RA) isolation—and set of scalable, coordination-free algorithms—called Read Atomic Multi-Partition (RAMP) Transactions—for addressing the isolation requirements of these use cases [7]. Informally, RA guarantees *atomic visibility* of updates: once one write from a transaction is visible, all writes will be visible. Existing protocols for achieving RA (or stronger), such as distributed locking, couple atomic visibility with mutual exclusion; RAMP achieves the former without the cost of the latter. RAMP uses limited multi-versioning to allow clients to operate concurrently over the same data items while ensuring that readers can correctly detect and repair incomplete writes. Across a range of workloads (including high contention scenarios), RAMP transactions incur limited overhead and outperform existing mechanisms for achieving RA while scaling linearly.

Invariant Confluence. While RAMP and HAT provided examples of "correctness" without coordination, I began to seek a more general principle for coordination-avoiding execution. Namely, I examined the use of *invariants*, or declarative, data-centric correctness criteria, as a basis for coordination avoidance (in contrast with low-level read/write races [1]). I adapted a property from rule-based rewriting systems called invariant confluence (I-confluence) to provide a simple test for understanding when coordination is strictly required [3]. Informally, invariant confluence ensures that the set of reachable database states (or histories) is closed under *merge*, or version exchange. If a set of transactions and invariants is I-confluent, then the invariants can be preserved during coordination-free transaction execution. If not, coordination is required to prevent data corruption in at least one execution of the transactions.

To apply I-confluence, I examined the integrity constraints offered by databases today—including the foreign key constraints addressed by RAMP but also row-level check constraints, uniqueness constraints, and constraints on abstract data types—and classified each as I-confluent or not. Many were I-confluent, so I subsequently applied this classification to a number of database workloads from the OLTPBenchmark suite. Many invariants in these workloads passed the I-confluence test as well. For example, in the TPC-C benchmark, the gold standard for transaction processing performance, ten of twelve invariants were I-confluent under the workload. Given this classification, I developed a database prototype and coordination-avoiding execution strategy for TPC-C that, on a cluster of 200 servers, achieves a 25-fold improvement in throughput over the prior best result (over 12.7M New-Order transactions per second).

Feral Concurrency Control. While I was able to find I-confluent database integrity constraints and benchmarks, were *real* applications I-confluent? Moreover, were invariants an actual practical choice of correctness criteria? To answer these questions, I examined open source web applications. I found that popular web programming frameworks—including Ruby on Rails, Django, and Spring/Hibernate—have introduced support for *validations*, or support for declarative, application-level invariants. I subsequently analyzed the use of validations in 67 of the most popular Ruby on Rails applications on GitHub and found that, in fact, validation usage was fourteen times more common than the use of database transactions. Moreover, 76% (of over 9700 validations) were I-confluent [4]. However, the remainder were *not* I-confluent and therefore required coordination. For these invariants, I profiled the incidence of invariant violations (similar to PBS) both with and without validations. This highlighted the applicability of I-confluence and unearthed a surprising practitioner trend away from transactions and towards invariants via validations.

Practical surveys. During my dissertation work, I co-authored two practitioner-oriented surveys appearing in ACM Queue and CACM; each highlights tenets of my research approach. The first describes issues and trends within the theory and practice of coordination-free system designs [8]. This article exemplifies a low-overhead approach to technology transfer I regularly engage in both on my blog and via social media. Organizing recent literature in an approachable manner can be tremendously valuable to practical research consumers. Given the amount of work that goes into each of my research projects, I am happy to spend a few hours organizing results in a friendly, easy-to-digest presentation (or demonstration, like http://pbs.cs.berkeley.edu/#demo). The second article (co-authored with a practicing engineer) collects data on the incidence of network communication partitions and their impact on deployed software in the wild [11]. This exemplifies, in my opinion, the importance of working *with* industry; while the importance of partitions is hotly contested within academia and industry, our collected data—a corpus I could not have assembled without Kyle—unequivocally highlights the danger of ignoring communication failures in system design.

Beyond Traditional Data Management: "Big Data" and Asynchrony

In my fourth and final year of graduate studies, I began to study coordination-avoiding techniques in contexts beyond traditional data management systems—namely, complex analytics and prediction tasks.

Asynchronous Convex Optimization via Plasma. A growing set of theoretical results in convex optimization highlights the benefit of coordination-free execution mechanisms such as updating a shared weight vector without synchronization: these mechanisms can lead to statistical convergence more quickly than traditional, coordinated approaches. Unfortunately, these results are at architectural odds with the resurgence of interest in large-scale dataflow architectures for large-scale training: systems such as MapReduce and Spark (as well as frameworks using these systems, like Mahout and MLlib) present an inherently synchronous BSP-like programming model. In response, with colleagues in the AMPLab, I developed support for single-stage, fine-grained and asynchronous data sharing between BSP tasks—a pattern we call *Bulk Asynchronous Parallel* [10]. Our BAP prototype implementation, called Plasma, runs within Spark and demonstrates up to two order-of-magnitude improvements in learning tasks via the use of asynchrony (in learning terms, via dual averaging and asynchronous ADMM). This allows seamless integration of asynchronous learning tasks within existing workflows without requiring the addition of yet another "Big Data" learning system.

Low Latency Serving of Data Products with Velox. As a member of a multi-member team in the AMPLab, I started the Velox project, which is designed to provide low latency and scalable serving of complex, statistically intensive *data products* [18]. A range of increasingly standardized analysis tools have facilitated machine learning tasks such as model training at large scale. However, once computed—often in batch—these models must yet again be transformed and loaded into yet another set of systems in order to be used in tasks including personalized recommendations, ad targeting, and anomaly detection. These pipelines are often ad-hoc, manually run, and require considerable engineering overhead. Our goal in Velox is to simplify this process, in effect doing for model serving what MapReduce did for model training. It is too expensive (in terms of latency and resources) to compute a given model from scratch on demand, so Velox treats model prediction as an instance of asynchronous materialized view maintenance. By exposing a declarative interface for model specification, refresh, and prediction, Velox automates the data product pipeline. By exploiting—as in Plasma—the statistical nature of the problem, Velox performs cost-based analysis to determine exactly what parts of the model to materialize and re-train. Velox is an alpha component of the Berkeley Data Analytics Stack (BDAS).

Other research. My prior research examined operating system scheduling policies for processor cooling [13], operating systems for unmanned aerial vehicle swarms [19], and multi-agent behavior in bee foraging models [12].

Future Research

In the future, I plan to continue my research in the field of data management, with a focus on large-scale and distributed systems. The demands of practical deployments raise a number of challenges for future database systems, including:

Bespoke Data Management. As my dissertation work highlights, there is rarely a unilateral "best" choice in concurrency control: rather, the optimal choice often depends on application semantics, workload characteristics, and dataset distribution. This trend is prevalent across almost every layer of a data management system, from storage to query execution. However, historically, much of the work on automated systems optimization has focused on queries, rather than the whole systems stack. This has, in part, contributed to the wide-spread use of "polyglot persistence"—or a separate system for each (class of) workload(s). To what extent can we automate this broader class of design choices on behalf of end users? My work on coordination avoidance hints at a combination of language design and runtime adaptivity, and I see a wide range of cross-stack optimization throughout the database architecture. Database systems have thrived due to declarativity, and adopting declarative policies that simultaneously provide adaptivity for coordination, execution, data layout and partitioning enables many interesting research opportunities.

Turning up data volume. By volume, an increasing proportion of today's data comes as insertion of facts external to traditional databases—such as log data, sensor readings, and other outside measurements. Much of this data is not human-generated: for example, gathering eight bytes (e.g., a bank balance, GPS coordinate, or combined heart rate, weight, and height) from each of the approximately seven billion humans on Earth today would require only 56 gigabytes. But if we consider the trend towards automated data collection—for example, many biometric signals over time or via ambient readings from the environment—and volume explodes. I see many opportunities for interesting research in this shift. For example, as data volumes continue to grow, storage prices drop, and heterogeneous storage media become more prevalent, how can we support peta-scale, multi-tiered aggressive data aging? Can we simultaneously support both

low-latency "rifle-shot" queries over hot data and higher-latency complex analytics within the same system? Addressing the needs of these emerging workloads will require significant architectural restructuring.

Service Composition and Consolidation. As the Velox project highlights, today's large-scale services are built via a patchwork of interconnected data-intensive services. This ad-hoc assembly is often a manual process, with limited tooling and support for understanding and optimizing behavior across services. I see tremendous value in better understanding service-based composition. As an example, these services are, in effect, sophisticated, specialized query processors, where the input language is a DSL expressed via calls to front-end web servers and the query processor is spread across multiple mid-tier and back-end services. While traditional databases are relegated to the final, rear tier, revisiting these service-oriented architectures as new forms of database architectures yields opportunities in query planning, language design, and execution optimization. Moreover, as projects within the Hadoop stack have shown, exposing programming abstractions at a variety of levels—from low-level dataflow to languages like Pig, UDFs, and SQL—brings many advantages yet is much less prominently supported in classic data management architectures.

Large-Scale Statistical Analytics. My initial foray into large-scale statistical analytics with the Plasma and Velox projects highlighted the power of leveraging statistical robustness in many common "Big Data" procedures. I am interested in further exploring the relationship between coordination-avoiding database primitives—such as Escrow transactions, the demarcation protocol, and weak isolation levels—and these emerging computational tasks. There is a range of interesting questions here, from algorithmic advances to systems support for these computation patterns.

References

[1] Peter Alvaro, Peter Bailis, Neil Conway, and Joseph M. Hellerstein. Consistency Without Borders. In *SoCC*, 2013.

[2] Peter Bailis, Aaron Davidson, Alan Fekete, Ali Ghodsi, Joseph M. Hellerstein, and Ion Stoica. Highly Available Transactions: Virtues and Limitations. In *VLDB*, 2014.

[3] Peter Bailis, Alan Fekete, Michael J. Franklin, Ali Ghodsi, Joseph M. Hellerstein, and Ion Stoica. Coordination Avoidance in Database Systems. In *VLDB*, 2015.

[4] Peter Bailis, Alan Fekete, Michael J. Franklin, Ali Ghodsi, Joseph M. Hellerstein, and Ion Stoica. Feral Concurrency Control: An Empirical Investigation of Modern Application Integrity. In *SIGMOD*, 2015.

[5] Peter Bailis, Alan Fekete, Ali Ghodsi, Joseph M. Hellerstein, and Ion Stoica. The Potential Dangers of Causal Consistency and an Explicit Solution. In *SoCC*, 2012.

[6] Peter Bailis, Alan Fekete, Ali Ghodsi, Joseph M. Hellerstein, and Ion Stoica. HAT, not CAP: Towards Highly Available Transactions. In *HotOS*, 2013.

[7] Peter Bailis, Alan Fekete, Ali Ghodsi, Joseph M. Hellerstein, and Ion Stoica. Scalable Atomic Visibility with RAMP Transactions. In *SIGMOD*, 2014. INVITED TO ACM TODS "BEST OF SIGMOD 2014".

[8] Peter Bailis and Ali Ghodsi. Eventual Consistency Today: Limitations, Extensions, and Beyond. *ACM Queue*, 11(3), March 2013. Also appears in *Communications of the ACM* 56(3):55-63, May 2013.

[9] Peter Bailis, Ali Ghodsi, Joseph M. Hellerstein, and Ion Stoica. Bolt-on Causal Consistency. In *SIGMOD*, 2013.

[10] Peter Bailis, Joseph E. Gonzalez, Michael J. Franklin, Joseph M. Hellerstein, Michael I. Jordan, and Ion Stoica. Asynchronous Complex Analytics in a Distributed Dataflow Architecture. Under review.

[11] Peter Bailis and Kyle Kingsbury. The Network is Reliable: An Informal Survey of Real-World Communications Failures. *ACM Queue*, 12(7), July 2014. Also appears in *Communications of the ACM* 57(9):48-55, September 2014.

[12] Peter Bailis, Radhika Nagpal, and Justin Werfel. Positional Communication and Private Information in Honeybee Foraging Models. In *ANTS*, 2010. BEST STUDENT PAPER.

[13] Peter Bailis, Vijay Janapa Reddi, Sanjay Gandhi, David Brooks, and Margo Seltzer. Dimetrodon: Processor-level Preventive Thermal Management via Idle Cycle Injection. In *DAC*, 2011.

[14] Peter Bailis, Shivaram Venkataraman, Michael J. Franklin, Joseph M. Hellerstein, and Ion Stoica. Probabilistically Bounded Staleness for Practical Partial Quorums. In *VLDB*, 2012. INVITED TO VLDB JOURNAL "BEST OF VLDB 2012" AND CACM RESEARCH HIGHLIGHT.

[15] Peter Bailis, Shivaram Venkataraman, Michael J. Franklin, Joseph M. Hellerstein, and Ion Stoica. PBS at Work: Advancing Data Management with Consistency Metrics. In *SIGMOD*, 2013.

[16] Peter Bailis, Shivaram Venkataraman, Michael J. Franklin, Joseph M. Hellerstein, and Ion Stoica. Quantifying Eventual Consistency with PBS. *Communications of the CACM (Research Highlight)*, 57(8):93–102, August 2014.

[17] Peter Bailis, Shivaram Venkataraman, Michael J Franklin, Joseph M Hellerstein, and Ion Stoica. Quantifying Eventual Consistency with PBS. *The VLDB Journal*, 23(2):279–302, 2014. "Best of VLDB 2012" Special Issue.

[18] Dan Crankshaw, Peter Bailis, Joseph E. Gonzalez, Haoyuan Li, Zhao Zhang, Michael J. Franklin, Ali Ghodsi, and Michael I. Jordan. The Missing Piece in Complex Analytics: Low Latency, Scalable Model Management and Serving with Velox. In *CIDR*, 2015.

[19] Karthik Dantu, Bryan Kate, Jason Waterman, Peter Bailis, and Matt Welsh. Programming Micro-aerial Vehicle Swarms with Karma. In *SenSys*, 2011.