# Challenges to Adopting Stronger Consistency at Scale

Phillipe Ajoux*, Nathan Bronson*, Sanjeev Kumar*, Wyatt Lloyd†*, Kaushik Veeraraghavan*
*Facebook, †University of Southern California

## Abstract

There have been many recent advances in distributed systems that provide stronger semantics for geo-replicated data stores like those underlying Facebook. These research systems provide a range of consistency models and transactional abilities while demonstrating good performance and scalability on experimental workloads. At Facebook we are excited by these lines of research, but fundamental and operational challenges currently make it infeasible to incorporate these advances into deployed systems. This paper describes some of these challenges with the hope that future advances will address them.

## 1 Introduction

Facebook is a social network that connects 1.35 billion people [19]. Facebook's social graph reflects its users, their relationships, the content they create, and the actions they take. The social graph is large and constantly growing and changing. Data from this graph is stored in several geo-replicated data stores and tightly integrated caching systems. Recent research results have shown how to provide stronger properties for the data layers that underly these systems, which has the potential to improve user experience and simplify application-level programming at Facebook. Results include work on scalable causal consistency [1, 4, 16, 17, 31, 32], strong consistency [12, 18, 22, 29, 36, 44], and transactions [5, 13, 27, 33, 41, 45, 47]. These ideas excite and inspire us, but we have not adopted them yet. Why not?

In this paper we identify barriers to our deployment of a strongly consistent or causally consistent data model. A system for ensuring consistency must: (1) integrate consistency across many stateful services, (2) tolerate high query amplification, (3) scale to handle linchpin objects, and (4) provide a net benefit to users.

Facebook relies on sharding, data replication, and caching to efficiently operate at scale. When a node or edge is added to the social graph, it is first written to a single MySQL instance. From there it is asynchronously replicated to all of our data centers, updated or invalidated in multiple layers of cache, and delivered through a scalable publisher-subscriber system to other services such as Search and News Feed. No single data placement strategy can efficiently serve all workloads in a heavily sharded system, so many of these services choose a specialized sharding function and maintain their own data store, caches, and indexes.

The biggest barrier to providing stronger consistency guarantees in an environment like Facebook's is that the consistency mechanism must **integrate consistency across many stateful services**. All of the scaling mechanisms described above lead to extra copies of data. Even if each of our caches and independent services were linearizable [24], inconsistencies would still be present in the aggregated result. Inter-service tracking is complicated by services that store data derived from a query to a lower layer, making object-level tracking insufficient. While these challenges are acute for Facebook at its present scale, we first encountered them when Facebook was much smaller. Solutions to these problems will benefit the growing class of applications whose implementation relies on sharding and separation into stateful services.

Another fundamental challenge is that a general consistency mechanism at scale must **tolerate high query amplification**. Fine-grained sharding and separation into optimized services means that the application layer acts like a heuristic query planner. Application logic implements a complex high-level query by issuing many subqueries. In practice, a single user request may result in thousands of subqueries, with a critical path that is dozens of subqueries long. The fork/join structure of subqueries causes latency outliers to have a disproportionate effect on total latency, and the large number of subqueries would cause slowdowns or unavailability to quickly propagate through the social graph when attempting to preserve stronger consistency guarantees.

In addition to the scaling challenges of Facebook's data size and query volume, a system for strengthening consistency at Facebook must **scale to handle linchpin objects**. Over time, Facebook's graph has evolved to contain many different types of objects including friends, posts, comments, photos, etc. While the cardinality of the incoming and outgoing edges from most objects is low, objects such as the Facebook Page of a celebrity of popular athlete have a problematic combination of high cardinality, high read rate, and high write rate. Linch-

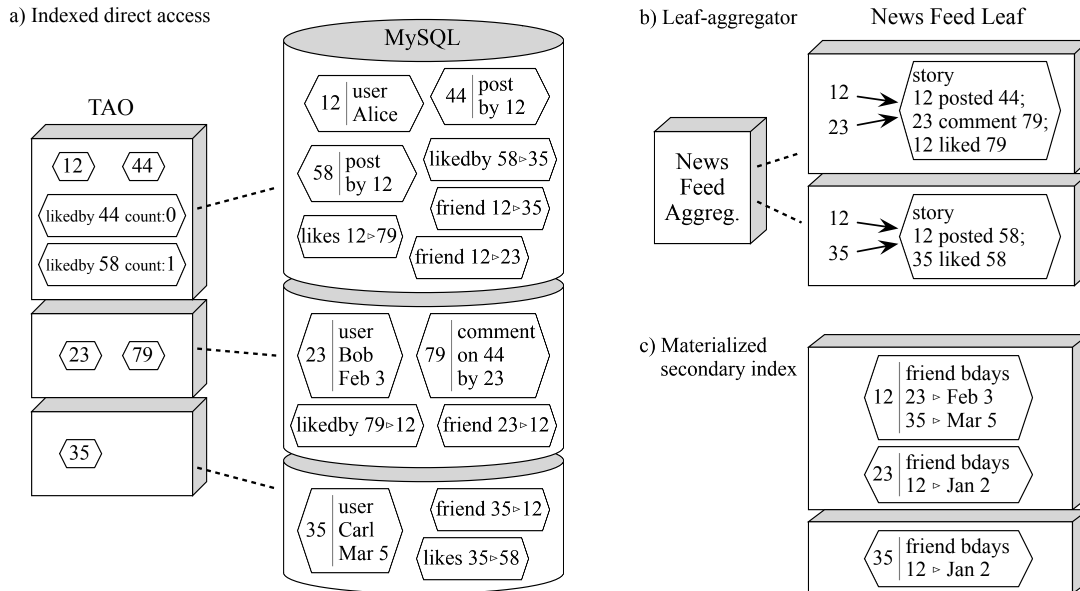


**Figure 1: A hypothetical data layout for two posts by Alice, with interactions by friends Bob and Carl. MySQL and TAO (a) shard nodes by key and edges by their source. In News Feed (b) information is aggregated and indexed by all actors. Materialized secondary indexes (c) answer queries that would touch many TAO shards.**

pin objects accelerate the propagation of dependencies throughout the system due to their highly connected nature, and they are a throughput challenge for systems in which readers can delay writers or vice versa.

To justify building and deploying a system with strong consistency guarantees we must **provide a net benefit to users**. It is easy to argue that user experience will be qualitatively better if consistency improves and all other system characteristics stay the same, but in practice the extra communication and coupling required for stronger distributed properties will increase latency or reduce availability. It is not obvious that a system that trades stronger consistency for increased latency or reduced availability would be a net benefit to people using Facebook, especially when compared against a weakly consistent system that resolves many inconsistencies with ad hoc mechanisms.

These fundamental challenges are joined by operational challenges: (1) fully characterizing worse-case throughput, (2) a polyglot environment, (3) varying deployment schedules, and (4) reduced incremental benefit from existing consistency workarounds.

We focus our discussion on the challenges of providing stronger forms of read consistency. This problem is a subset of the problem tackled by systems that provide strong read semantics and transactional writes.[1]

[1] An option we do not explore in this paper is a transaction-like facility that provides failure atomicity in an eventually consistent system.

# 2 Fundamental Challenges

This section describes fundamental challenges to adopting stronger consistency and transactions at Facebook.

## 2.1 Integrating Across Stateful Services

Facebook's back-end infrastructure is comprised of many different services, each using a data placement and communication structure optimized for its workload. Figure 1 shows how data may be duplicated with different sharding strategies in different services.

The TAO caching layer [10], for example, uses multi-tenancy and hierarchical caching to tolerate very high query rates for simple graph queries. Another system materializes secondary indexes to support specific queries that would have a high fanout or require multiple rounds in TAO. Unicorn [14] uses document sharding and a leaf-aggregator model to handle the dynamic queries from Facebook's Search. The News Feed backend denormalizes all of the information associated with a candidate story so that ranking can be performed locally. Facebook's deployment of memcache [21, 35] uses the flexible mcrouter proxy [30] to effect a wide variety of replication and locality strategies. Facebook's Messenger service uses a data store optimized for efficient access to recent messages [20], but often embeds links to TAO nodes. In total, there are hundreds of services running in production at Facebook, many of which maintain a copy of portions of the social graph.

Facebook's architecture of cooperating heterogeneous services is different from the monolithic service that most research designs assume. This difference alone is not fundamental—Facebook can be externally viewed as a single service even though internally it is comprised of many services. Instead, it is the disaggregated nature of these internal services that creates the challenges that have yet to be solved: (1) storing data consistently across services; (2) handling decentralized access to services; and, most importantly, (3) composing results from services that internally store data from subqueries.

**Storing Data Consistently Across Services** The same data lives in many different services. No single data layout is efficient for all workloads, especially at scale. In a centralized relational model this is reflected by the wide variety of storage engines and index types available, and by features such as materialized views. In a sharded and scaled environment like Facebook's, services may maintain their own optimized cache, index, or copy of a portion of the social graph. Wormhole [40] is our scalable pub-sub system that asynchronously delivers update notifications to all of the services that manage state.

Services can appear stateful without directly managing their own state when their results are memoized in memcache. Another class of hidden stateful service mirrors database triggers, where a service issues new writes when it is notified of an existing data changes.

To provide strong consistency across our services, we would need to propagate update notifications synchronously impacting availability and latency. A causal system could avoid atomic updates, but would still need to be integrated into every service and threaded through every communication channel greatly increasing the complexity of the dependency graph. These effects are especially challenging for services that perform aggregation, because the input dependencies of a result may change much more frequently than the result itself.

**Decentralized Access to Services** The Facebook web site and its mobile applications use parallelism and pipelining to reduce latency. The notification bar and the central content are fetched by separate requests, for example, and then merged at the client. Mobile applications also aggressively cache results using local storage, and then merge query results with previously seen values. The application layer also makes extensive use of asynchrony, overlapping as many service subqueries as possible. This complex execution structure is necessary to overlap communication delays, but it leaves the user's device as the only safe location to define session-based guarantees [43] or demarcate isolation boundaries.

Requests for a single user are usually routed to the same cluster in the same region. This routing stability assists in reducing the number of user-visible inconsistencies, because most of the variance in asynchronous update propagation is inter-region. Cluster stickiness is not a guarantee, however; users are moved whenever the total capacity of a cluster or data center drops suddenly.

**Composing Results** An important complication of inter-service composition is that services often store the results of range scans or other complex queries, such as the list of people that 'Like' something. An inter-service coordination mechanism must capture the dependencies of the queries, rather than of the data that results. In a monolithic data system these dependencies are tracked via predicate or range locks. The difference is particularly clear in negative caches, where a service keeps track locally of things that did not exist during an earlier query. Each service can be thought of as implementing its own limited query language, so it is not clear that the dependencies can be captured in a way that is both sufficiently generic and sufficiently precise.

## 2.2 Query Amplification

A single user request to Facebook amplifies to thousands of, possibly dependent, internal queries to many services. A user request to Facebook is handled by a stateless web server that aggregates the results of queries to the various Facebook services. These results in turn might lead to more queries to the same, or other, services. For example, a user's home page queries News Feed to get a list of stories, which depends on a separate service to ensure story order of previously seen content. These stories are then fetched from TAO to get the content, list of comments, likes, etc. In practice, this behavior produces fork/join query patterns that have fan-out degrees in the hundreds and critical paths dozens deep. Query amplification presents two hurdles to strengthening consistency.

**Slowdown Cascades** Different services use different sharding and indexing schemes for the same data. For instance, TAO shards data based on a 64-bit integer id [10], News Feed shards based on a user id, and the secondary index system shards based on values. The use of different sharding schemes creates all-to-all ordering dependencies between shards in different services. This connectedness accelerates slowdown propagation in strongly consistent systems. A failure in one shard of one service would propagate to all shards of a downstream service.

Consider a single slow node in a service. In a weakly consistent system, this node could return possibly inconsistent, stale data. With millions of user requests hitting the system it is likely the slow node will be queried. In a weakly consistent system, this slow node poses a consistency problem, but does not affect the latency of the

system. However, in a system with stronger consistency, this slow node would cause increase in latency for operations. Given the query amplification, this slowdown can quickly cascade to all nodes of other systems.

**Latency Outliers** Parallel subqueries amplify the impact of latency outliers. A user request does not return until it has joined the results of all of its queries, and thus must wait for the slowest of its internal requests. For example, a web server generating a response might query TAO for a list of all the people who like a post, and then follow-up with a set of queries for those people's names. TAO's sharding strategy makes it likely that each of the followup queries goes to a different server. If even one of those subqueries has a high latency, then the web server's response will be delayed. Thus, 99$^{th}$ percentile latency in a low-level service has a non-outlier effect on overall user request latency. Internally we monitor outlier latencies up to the 99.99$^{th}$ percentile for some services. Strongly consistent systems with larger outlier latencies further worsen user request latency.

### 2.3 Linchpin Objects

The read rate, write rate, and dependence structure of the social graph varies widely. The most frequently read objects are often also frequently written to and highly connected. Examples of these linchpin objects include popular users like celebrities, popular pages like major brands, and popular locations like tourist attractions. By providing stronger consistency at the level of a shard, object, or other granularity, systems can often achieve better performance and throughput. Linchpin objects pose a direct challenge for these systems because the throughput of individual objects or shards must be very high. A system that provides stronger consistency must provide good performance for these linchpin objects.

Linchpin objects and query amplification combine to make outliers even more of a challenge. A linchpin object is likely to be included in a large fraction of user requests. If that object is the slowest request, or is on the slowest request path, then increasing its latency will increase user latency. Thus even moderate increases in read latency for linchpin objects is likely to have an outsized effect on user latency.

### 2.4 Net Benefit to Users

The intuitive arguments for stronger forms of consistency are attractive. Systems with strong consistency are easier for programmers to reason about and build on top of [6, 12] and they provide a better experience for users because they avoid some anomalous application behavior. However, these benefits are hard to quantify precisely, and they do not appear in isolation. When all aspects of user experience are considered it might even be the case that the benefit does not outweigh the detriment.

Improvements to programmer productivity arise from strengthening the properties or guarantees provided by the system, because there is a whole class of rare behavior that can be assumed to never occur. On the other hand, user benefits may be assumed to be proportional to the actual rate at which inconsistencies can be observed. Even in a weakly consistent system this rate may be very low. If only one in a million queries to a weakly consistent system returns an anomalous result, the benefit from excluding this type of behavior is less than it would be if inconsistent results were frequent.

Stronger properties are provided through added communication and heavier-weight state management mechanisms, increasing latency. Although optimizations may minimize these overheads during failure-free execution, failures are frequent at scale [15]. Higher latency may lead to a worse user experience, potentially resulting in a net detriment.[2]

Note that we are concerned here with the intrinsic latency implications of a more elaborate protocol, rather than with the downsides of increased complexity or reduced per-machine throughput. We consider those operational rather than fundamental challenges.

## 3 Operational Challenges

Our experience building and scaling Facebook points to additional operational challenges that we would face in trying to deploy a generic consistency solution.

**Fully Characterizing Worst-Case Throughput** Facebook operates continuously; we do not get a break after a failure during which we can reduce load to catch up on a work backlog or perform intensive recovery operations. Our data and services infrastructure has been designed, tuned, and battle-hardened to allow us to recover from error conditions with minimal impact to our users. Some of our systems discard work to catch up. Some perform opportunistic batching to increase their throughput when there is a work backlog. Some systems are sized for worst-case conditions.

While there is no theoretical impediment to preserving worst-case throughput despite strengthening Facebook's consistency guarantees, our experience is that failure and recovery behavior of a complex system is very difficult to characterize fully. Emergent behavior in cross-service interactions is difficult to find ahead of time, and may even be difficult to identify when it is occurring [9].

[2] We expect the latency difference needs to be sufficiently large and far away from the minimum perceivable threshold to matter.

**Polyglot Environment** While C++ is the predominant language for standalone services at Facebook, Python, Java, Haskell, D, Ruby and Go are also supported by Thrift, our inter-service serialization format. The application code coordinating service invocations is generally written in Hack and PHP, and final composition of query results may be performed by JavaScript in the browser, Objective C/C++ on IOS, or Java on Android.

Ports or foreign function interfaces must be provided to give all of these codebases access to an end-to-end consistency platform. Perhaps the trickiest part of this multi-language support is the wide variety of threading models that are idiomatic for different languages and runtimes, because it is likely the underlying consistency system will need to initiate or delay communication.

**Varying Deployment Schedules** The iteration speed of software is limited by its deployment schedule. Facebook has an extremely aggressive deployment process for the stateless application logic [38], but we are necessarily more conservative with stateful and mature low-level services. An inter-service coordination mechanism that is used by or interposes on these low-level services will have a deployment velocity that is constrained by the release engineering requirements of the most conservative component. Reduced deployment velocity increases practical difficulty and software engineering risk.

**Reduced Incremental Benefit in a Mature System** Are the upsides of fewer anomalies worth the downside of increased latency? Perhaps, but we currently lack the data to evaluate this. Further complicating the decision are the consistency workarounds already in place in a mature system. Known, compelling examples of Facebook workloads that need strong consistency already have been architected with a narrowly scoped consistency solution. Some operations bypass caches and read directly from a single authoritative database. Some use cases have an application-level schema that lets them repair temporary inconsistencies in the graph. Some systems expose routing information to their callers so that inter-region inconsistencies are avoided by proxying. While these workarounds would not exist in a newly built system, their presence in Facebook reduces the incremental benefits of deploying a generic system for stronger consistency guarantees.

# 4 Related Work

**Prior Facebook Publications** Earlier publications from Facebook include descriptions of the memcache cache and its workload [2, 35], the TAO graph store [10], the Wormhole pub-sub system [40], a characterization of load imbalance in the caches [26], an analysis of the messages use case [23], and work on understanding and improving photo/BLOB storage and delivery [7, 25, 34, 42]. The memcache and TAO papers discuss details of our inter-region consistency mechanisms and guarantees. This paper addresses the big picture with challenges across Facebook's internal services.

**Systems with Stronger Semantics** There is a long history of work on systems that provides stronger semantics, from replicated state machines [28, 39], to causal and atomic broadcasts [8], to systems designed for high availability and stronger consistency [37]. Recent work builds on the classic to provide similar properties for datacenter-scale and/or geo-replicated services. This includes work on scalable causal consistency [1, 4, 16, 17, 31, 32], strong consistency [12, 18, 22, 29, 36, 44], and transactions [5, 6, 12, 13, 27, 32, 33, 41, 45, 46, 47]. This research inspired this paper and we hope to help inform the future efforts of researchers.

Interestingly, work in these directions sometimes improves one dimension at the cost of another. For instance, Orbe [16] and GentleRain [17] improve upon the throughput of COPS [31] by tracking less fine-grained information about the causal dependencies between update operations. But, these techniques would result in even more severe slowdown cascades (§2.2). We hope this paper leads to a more thorough understanding of such trade offs.

**Discussions of Challenges to Stronger Consistency** Our work is inspired by previous discussions of some similar challenges. Cheriton and Skeen [11] present a criticism of enforcing an order in a communication substrate instead of end to end. Bailis et al. [3] discuss slowdown cascades to motivate enforcing only an explicit subset of causal consistency. Dean and Barroso [15] detail the importance and challenges for tail latency in high-scale services at Google, similar to our discussion of the effect of latency outliers due to query amplification. Our work is primarily distinguished by the breadth of challenges it covers and its focus on describing both fundamental and operational challenges for the complete Facebook system as it is currently constructed.

# 5 Conclusion

There is an exciting boom in research on scalable systems that provide stronger semantics for geo-replicated stores. We have described the challenges we see to adopting these techniques at Facebook, or in any environment that scales using sharding and separation into stateful services. Future advances that tackle any of these challenges seem likely to each have an independent benefit. Our hope is that these barriers can be overcome.

# References


[1] S. Almeida, J. Leitao, and L. Rodrigues. Chainreaction: a causal+ consistent datastore based on chain replication. In *EuroSys*, 2013.

[2] B. Atikoglu, Y. Xu, E. Frachtenberg, S. Jiang, and M. Paleczny. Workload analysis of a large-scale key-value store. In *SIGMETRICS*, 2012.

[3] P. Bailis, A. Fekete, A. Ghodsi, J. M. Hellerstein, and I. Stoica. The potential dangers of causal consistency and an explicit solution. In *SOCC*, 2012.

[4] P. Bailis, A. Ghodsi, J. M. Hellerstein, and I. Stoica. Bolt-on causal consistency. In *SIGMOD*, 2013.

[5] P. Bailis, A. Fekete, A. Ghodsi, J. M. Hellerstein, and I. Stoica. Scalable atomic visibility with ramp transactions. In *SIGMOD*, 2014.

[6] J. Baker, C. Bond, J. C. Corbett, J. Furman, A. Khorlin, J. Larson, J.-M. Leon, Y. Li, A. Lloyd, and V. Yushprakh. Megastore: Providing scalable, highly available storage for interactive services. In *CIDR*, Jan. 2011.

[7] D. Beaver, S. Kumar, H. C. Li, J. Sobel, and P. Vajgel. Finding a needle in haystack: Facebook's photo storage. In *OSDI*, 2010.

[8] K. P. Birman and R. V. Renesse. *Reliable Distributed Computing with the ISIS Toolkit*. IEEE Comp. Soc. Press, 1994.

[9] N. Bronson. Solving the mystery of link imbalance: A metastable failure state at scale. `https://code.facebook.com/posts/1499322996995183/solving-the-mystery-of-link-imbalance-a-metastable-failure-state-at-scale/`, 2014.

[10] N. Bronson, Z. Amsden, G. Cabrera, P. Chakka, P. Dimov, H. Ding, J. Ferris, A. Giardullo, S. Kulkarni, H. Li, M. Marchukov, D. Petrov, L. Puzar, Y. J. Song, and V. Venkataramani. Tao: Facebook's distributed data store for the social graph. In *USENIX ATC*, 2013.

[11] D. R. Cheriton and D. Skeen. Understanding the limitations of causally and totally ordered communication. In *SOSP*, Dec. 1993.

[12] J. C. Corbett, J. Dean, M. Epstein, A. Fikes, C. Frost, J. Furman, S. Ghemawat, A. Gubarev, C. Heiser, P. Hochschild, W. Hsieh, S. Kanthak, E. Kogan, H. Li, A. Lloyd, S. Melnik, D. Mwaura, D. Nagle, S. Quinlan, R. Rao, L. Rolig, Y. Saito, M. Szymaniak, C. Taylor, R. Wang, and D. Woodford. Spanner: Google's globally-distributed database. In *OSDI*, Oct. 2012.

[13] J. Cowling and B. Liskov. Granola: low-overhead distributed transaction coordination. In *USENIX ATC*, June 2012.

[14] M. Curtiss, I. Becker, T. Bosman, S. Doroshenko, L. Grijincu, T. Jackson, S. Kunnatur, S. Lassen, P. Pronin, S. Sankar, G. Shen, G. Woss, C. Yang, and N. Zhang. Unicorn: A system for searching the social graph. *VLDB*, 2013.

[15] J. Dean and L. A. Barroso. The tail at scale. *Comm. ACM*, 56(2), 2013.

[16] J. Du, S. Elnikety, A. Roy, and W. Zwaenepoel. Orbe: Scalable causal consistency using dependency matrices and physical clocks. In *SOCC*, 2013.

[17] J. Du, C. Iorgulescu, A. Roy, and W. Zwaenepoel. Gentlerain: Cheap and scalable causal consistency with physical clocks. In *SOCC*, 2014.

[18] R. Escriva, B. Wong, and E. G. Sirer. HyperKV: A distributed, searchable key-value store for cloud computing. In *SIGCOMM*, 2012.

[19] Facebook's Company Info. `http://newsroom.fb.com/company-info/`, 2014.

[20] J. Fein and J. Jenks. Building mobile-first infrastructure for messenger. `https://code.facebook.com/posts/820258981365363/building-mobile-first-infrastructure-for-messenger/`, 2014.

[21] B. Fitzpatrick. Memcached: a distributed memory object caching system. `http://memcached.org/`, 2014.

[22] L. Glendenning, I. Beschastnikh, A. Krishnamurthy, and T. Anderson. Scalable consistency in Scatter. In *SOSP*, Oct. 2011.

[23] T. Harter, D. Borthakur, S. Dong, A. Aiyer, L. Tang, A. C. Arpaci-Dusseau, and R. H. Arpaci-Dusseau. Analysis of hdfs under hbase: A facebook messages case study. In *FAST*, 2014.

[24] M. P. Herlihy and J. M. Wing. Linearizability: A correctness condition for concurrent objects. *ACM TOPLAS*, 12 (3), 1990.

[25] Q. Huang, K. Birman, R. van Renesse, W. Lloyd, S. Kumar, and H. C. Li. An Analysis of Facebook Photo Caching. In *SOSP*. ACM, 2013.

[26] Q. Huang, H. Gudmundsdottir, Y. Vigfusson, D. A. Freedman, K. Birman, and R. van Renesse. Characterizing load imbalance in real-world networked caches. In *HotNets*, 2014.

[27] T. Kraska, G. Pang, M. J. Franklin, S. Madden, and A. Fekete. Mdcc: Multi-data center consistency. In *EuroSys*, 2013.

[28] L. Lamport. Time, clocks, and the ordering of events in a distributed system. *Comm. ACM*, 21(7), 1978.

[29] C. Li, D. Porto, A. Clement, J. Gehrke, N. Preguiça, and R. Rodrigues. Making geo-replicated systems fast as possible, consistent when necessary. In *OSDI*, Oct. 2012.

[30] A. Likhtarov, R. Nishtala, R. McElroy, H. Fugal, A. Grynenko, and V. Venkataramani. Introducing mcrouter: A memcached protocol router for scaling memcached deployments. `https://code.facebook.com/posts/296442737213493/introducing-mcrouter-a-memcached-protocol-router-for-scaling-memcached-deployments/`, 2014.

[31] W. Lloyd, M. J. Freedman, M. Kaminsky, and D. G. Andersen. Don't settle for eventual: Scalable causal consistency for wide-area storage with COPS. In *SOSP*, Oct. 2011.

[32] W. Lloyd, M. J. Freedman, M. Kaminsky, and D. G. Andersen. Stronger semantics for low-latency geo-replicated storage. In *NSDI*, Apr. 2013.

[33] S. Mu, Y. Cui, Y. Zhang, W. Lloyd, and J. Li. Extracting more concurrency from distributed transactions. In *OSDI*, 2014.

[34] S. Muralidhar, W. Lloyd, S. Roy, C. Hill, E. Lin, W. Liu, S. Pan, S. Shankar, V. Sivakumar, L. Tang, and S. Kumar. f4: Facebook's warm blob storage system. In *OSDI*, 2014.

[35] R. Nishtala, H. Fugal, S. Grimm, M. Kwiatkowski, H. Lee, H. C. Li, R. McElroy, M. Paleczny, D. Peek, P. Saab, D. Stafford, T. Tung, and V. Venkataramani. Scaling memcache at facebook. In *NSDI*, 2013.

[36] J. Ousterhout, P. Agrawal, D. Erickson, C. Kozyrakis, J. Leverich, D. Mazières, S. Mitra, A. Narayanan, M. Rosenblum, S. M. Rumble, E. Stratmann, and R. Stutsman. The case for ramcloud. *ACM SIGOPS Operating Systems Review*, 43(4), 2010.

[37] K. Petersen, M. Spreitzer, D. Terry, M. Theimer, and A. Demers. Flexible update propagation for weakly consistent replication. In *SOSP*, Oct. 1997.

[38] C. Rossi. Ship early and ship twice as often. `https://www.facebook.com/notes/facebook-engineering/ship-early-and-ship-twice-as-often/10150985860363920`, 2012.

[39] F. B. Schneider. Implementing fault-tolerant services using the state machine approach: a tutorial. *ACM Computer Surveys*, 22(4), Dec. 1990.

[40] Y. Sharma, P. Ajoux, P. Ang, D. Callies, A. Choudhary, L. Demailly, T. Fersch, L. A. Guz, A. Kotulski, S. Kulkarni, S. Kumar, H. Li, J. Li, E. Makeev, K. Prakasam, R. van Renesse, S. Roy, P. Seth, Y. J. Song, K. Veeraraghavan, B. Wester, and P. Xie. Wormhole: Reliable pub-sub to support geo-replicated internet services. In *NSDI*, May 2015.

[41] Y. Sovran, R. Power, M. K. Aguilera, and J. Li. Transactional storage for geo-replicated systems. In *SOSP*, Oct. 2011.

[42] L. Tang, Q. Huang, W. Lloyd, S. Kumar, and K. Li. RIPQ: Advanced Photo Caching on Flash For Facebook. In *FAST*, Feb. 2015.

[43] D. B. Terry, A. J. Demers, K. Petersen, M. Spreitzer, M. Theimer, and B. W. Welch. Session guarantees for weakly consistent replicated data. In *PDIS*, Sept. 1994.

[44] D. B. Terry, V. Prabhakaran, R. Kotla, M. Balakrishnan, M. K. Aguilera, and H. Abu-Libdeh. Consistency-based service level agreements for cloud storage. In *SOSP*, 2013.

[45] A. Thomson, T. Diamond, S.-C. Weng, K. Ren, P. Shao, and D. J. Abadi. Calvin: fast distributed transactions for partitioned database systems. In *SIGMOD*, May 2012.

[46] C. Xie, C. Su, M. Kapritsos, Y. Wang, N. Yaghmazadeh, L. Alvisi, and P. Mahajan. Salt: combining acid and base in a distributed database. In *OSDI*, 2014.

[47] Y. Zhang, R. Power, S. Zhou, Y. Sovran, M. K. Aguilera, and J. Li. Transaction chains: achieving serializability with low latency in geo-distributed storage systems. In *SOSP*, pages 276–291. ACM, 2013.