

WHITE PAPER · DATABASE CACHING

Multi-Level Caching for Geo-Distributed, Edge and IoT Data

Hierarchies from device to edge to region to core

White Paper 6 · Database Caching series · Version 1.0 · 26 October 2026
Bassirah Technology — Innovation & R&D

Multi-Level Caching for Geo-Distributed, Edge and IoT Data

White Paper 6 · Database Caching series · Bassirah Technology — Innovation & R&D · 26 October 2026

Document control

Field	Value
Series	Database Caching, paper 6 of 6
Author	Bassirah Technology — Innovation & R&D
Version	1.0 · 26 October 2026
Scope	A technology survey. It describes published research and publicly documented systems; it does not report Bassirah benchmarks or customer results.

Contents

1. Abstract	4
2. Introduction	4
2.1 Why caches multiply	4
2.2 How this paper builds on the series.....	4
2.3 Scope.....	4
2.4 Terms used in this paper	5
3. The hierarchy	5
4. Lessons from hierarchical and cooperative caching.....	5
5. CDN tiers for data APIs	6
6. Caches and replicas across regions	6
6.1 Regional caches over a single primary.....	6
6.2 Leader and follower cache tiers.....	6
6.3 Read replicas and follower reads as cache tiers	6
6.4 Multi-primary regions with conflict-free data types.....	7
6.5 The economics of regional caches	7
7. Edge data stores.....	7
8. Devices and offline-first caches.....	8
8.1 Local-first data	8
8.2 Conflict-free replicated data types.....	8
8.3 Sync systems.....	8
9. IoT data	8
9.1 Edge tiers for sensor data.....	8
9.2 Freshness as the measure.....	9
9.3 Last-value caches	9
10. Consistency across levels.....	9
11. Security and data residency at the edge.....	9

12. Comparative analysis.....	10
13. Open challenges and future opportunities.....	10
13.1 Consistency and availability at the edge.....	10
13.2 Preserving intent when merging.....	11
13.3 Placement and sizing across tiers	11
13.4 Freshness and energy for IoT.....	11
13.5 Invalidation across many locations	11
13.6 Security and residency	11
13.7 Observability across the hierarchy.....	11
13.8 Platform change at the edge	11
14. What this means for platform decisions	12
References	12

1. Abstract

Data increasingly lives far from the people and devices that use it. Users are spread across regions, applications run at edge locations close to them, and sensors produce readings on devices with intermittent connectivity. Serving that data quickly means caching it at several levels at once: on the device, at the edge, in each region and in front of the core database. This paper examines how such hierarchies are built and what each level can promise. It draws on the research on hierarchical and cooperative caching, then describes CDN tiers for data APIs, caches and replicas across regions, multi-primary replication with conflict-free data types, edge key-value stores and edge databases, and offline-first caches on devices. It covers IoT data, where freshness matters more than hit ratio and Age of Information provides the measure, and the consistency choices that every level of the hierarchy forces. It then discusses security and data residency at the edge, compares the approaches qualitatively, sets out open challenges and closes the Database Caching series with notes for platform decisions.

2. Introduction

2.1 Why caches multiply

Three forces push caches outward from the core database. The first is latency: a request that must cross regions or continents pays for the distance on every round trip, and the consistency trade-offs that geo-distributed systems make stem largely from avoiding such round trips even when the network is healthy [1]. The second is cost and bandwidth: when data lives in another region or cloud, a cache in the consuming region can pay for itself by avoiding cross-region transfer charges [2], and IoT data volumes make shipping every reading to a central cloud inefficient [3]. The third is availability: edge computing places compute and storage close to devices partly so that they keep working when the connection to the cloud is lost [4]. Each force adds a cache level, and each level adds a copy that can be stale.

2.2 How this paper builds on the series

Paper 1 described caching strategies, eviction, invalidation and the pitfalls of a single cache tier, including HTTP and CDN caching. Paper 4 described tiers within one machine, from DRAM to flash. Paper 5 described invalidation and consistency between a cache and its database. This paper extends those ideas across geography and device classes. Paper 6 of the Data Storage Engines series described distributed SQL databases, whose read replicas and follower reads reappear here as cache tiers (Section 6.3).

2.3 Scope

This paper covers multi-level caching of data across devices, edge locations, regions and the core: CDN tiers in front of data APIs, regional caches and replicas, edge data stores, offline-first device caches and IoT data caching. It concentrates on peer-reviewed work and official documentation, and cites foundational work where it introduced an idea still in use. Statements about products reflect documentation accessed on 26 October 2026; edge platforms change quickly, and features may be added, renamed or retired. It does not report latency, hit-ratio, cost or freshness figures; where a source reports results, we describe them qualitatively and attribute them to the authors.

2.4 Terms used in this paper

The **origin** or **core** is the authoritative database or service. A **region** is a cloud region or data center with its own copy or cache of data. An **edge location** is a point of presence close to users. A **device** is a phone, browser, vehicle, gateway or sensor that holds a local copy. A **tier** is one level of this hierarchy. A **replica** is a copy maintained by replication rather than by cache fills. **Age of Information** is the time elapsed since the newest value a receiver holds was generated [5]. A **partition** is a loss of connectivity between parts of the system.

3. The hierarchy

Table 1 summarizes the levels a multi-level cache can use and what each typically provides. The descriptions are qualitative; actual behavior depends on the products, configuration and network.

Level	Typical store	Typical consistency	Main strengths	Main constraints
Device	Embedded database (for example SQLite), local document store, browser storage	Local reads always available; sync is eventual, often with conflict-free merging	Works offline; no network cost per read	Limited capacity and energy; untrusted environment
Edge location	CDN cache, edge key-value store, edge database replica	Usually eventual; some stores offer per-key or per-session guarantees	Close to users; absorbs read load	Many locations to invalidate; regional regulation
Region	Regional cache cluster, read replica, multi-primary replica	Eventual or bounded staleness; read-your-writes with extra mechanisms	Serves a region without cross-region round trips	Replication lag; cross-region conflicts
Core	Primary database and its own caches	Strong	Authoritative	Distance from users; capacity and cost of serving all reads

4. Lessons from hierarchical and cooperative caching

The questions multi-level data caches face were first studied for web caching. Che, Tung and Wang modeled cache hierarchies analytically and showed that tiers should be designed so that lower levels do not simply duplicate the contents of upper ones [6]. Summary cache let cooperating caches share compact Bloom-filter summaries of their contents instead of querying every peer, an alternative to strict hierarchy [7]. Wolman et al. found that cooperation among proxy caches brings diminishing returns beyond a modest population, because the benefit is bounded by how much the workload is shared and cacheable, not by capacity alone [8]. Two later results refine where capacity should go. eMRC shows that sizing a multi-tier cache requires understanding how misses at one tier feed the next, and approximates miss-ratio curves cheaply so that capacity can be allocated across tiers [9]. RobinHood shows, for requests that fan out to many backends, that reallocating cache capacity toward the backends that dominate tail latency beats spreading it evenly [10]. The common lesson is that each tier sees the filtered miss stream of the tier in front of it and should be sized for that stream.

5. CDN tiers for data APIs

Content delivery networks place servers close to users across a globally distributed platform [11], and they now cache data API responses as well as static content. Without tiering, every edge location that misses fetches from the origin independently. Tiered caching adds upper tiers: Cloudflare's Tiered Cache answers lower-tier misses from an upper tier so that only upper tiers contact the origin [12], and Amazon CloudFront, which already has regional edge caches as a middle tier, can add an Origin Shield through which all regional caches funnel, collapsing concurrent misses for the same object into as few origin requests as possible [13]. AWS notes that this helps geographically spread audiences and suits cacheable content, not dynamic or rarely requested responses [13].

HTTP semantics govern every such tier. RFC 9111 defines freshness, validation with conditional requests, and the difference between shared caches such as CDNs and private caches in user agents [14], and RFC 5861 adds stale-while-revalidate and stale-if-error, which let outer tiers keep serving while refreshing or while the origin fails [15]. Invalidation is the hard part of a distributed tier. CloudFront supports invalidation by path and by cache tag, but AWS recommends versioned file names for frequently updated content, because invalidation does not reach caches downstream of the CDN, such as browsers [16]. Fastly lets the origin tag responses with surrogate keys, so one purge removes a related group of objects, and offers soft purges that mark content stale while a fresh copy is fetched [17]. Surrogate keys are the CDN counterpart of the tag-based invalidation described in paper 5.

6. Caches and replicas across regions

6.1 Regional caches over a single primary

Facebook's memcache deployment shows the classic structure: caches per cluster, regional pools shared by several clusters, and regions that hold replicas of a single writable master region [18]. Invalidations are driven from the database commit log, and when a user writes from a non-master region, a remote marker redirects that user's subsequent misses to the master region until replication catches up, limiting stale reads [18].

6.2 Leader and follower cache tiers

TAO organizes its cache into a leader tier, with one coordinator per shard in each region, and follower tiers that clients use; followers forward misses and writes to the leader, and by restricting the data model, replica-region caches can be updated at write time so that clients sharing a cache see their own writes without inter-region calls [19].

6.3 Read replicas and follower reads as cache tiers

A nearby replica serving reads is a cache whose freshness is bounded by replication lag rather than by a time to live. CockroachDB's follower reads let the replica nearest the request serve reads at a timestamp old enough to be safe, either a fixed past timestamp or the freshest one the replica can serve without blocking; safety rests on closed timestamps below which no new writes can land [20]. Cloudflare's D1 places read-only replicas in several regions and gives each session sequential consistency through bookmarks: a replica waits until it is at least as current as the version the session last saw [21]. Upstash's Global Database serves reads from

any region and processes writes at a primary, documenting itself as eventually consistent [22]. These designs differ mainly in what they promise a session that writes and then reads.

6.4 Multi-primary regions with conflict-free data types

Some systems accept writes in every region. Redis Active-Active databases are built on conflict-free replicated data types: each participating cluster accepts reads and writes locally, a syncer propagates changes, and concurrent updates are merged automatically by the data types themselves, for example by adding concurrent counter increments, rather than by discarding one write [23]. The documentation describes strong eventual consistency with an optional causal consistency setting, notes that deleted keys persist as tombstones until all instances observe the delete [23], and, for Redis Cloud, states that failover between regions is handled by the application or network layer and that Active-Active needs more memory than a standalone database [24]. Section 8 describes the theory behind such data types.

6.5 The economics of regional caches

In the cloud, cache capacity is a cost choice rather than a hardware limit. MACARON's authors argue that a cache in the consuming region pays for itself mainly by avoiding cross-region and cross-cloud transfer charges, and that it should size and configure itself, including a cheaper object-storage tier behind memory, to minimize total cost [2]. The five-minute reasoning of paper 4 applies here with transfer prices in place of device prices.

7. Edge data stores

Edge platforms offer data stores that run next to edge compute, with consistency models stated in their documentation. Table 2 summarizes four documented designs.

Store	Write path	Read path	Documented consistency	Intended use
Cloudflare Workers KV	Central stores	Per-location cache, then regional and central tiers	Eventual; read-your-writes not guaranteed, even at the writing location	Read-heavy, highly cacheable data
Cloudflare Durable Objects	One instance per object name	The same instance	Strongly consistent, transactional per object	Coordination and per-entity state
Cloudflare D1 with read replication	Primary database	Regional read replicas through sessions	Sequential consistency per session	Relational data read from many regions
Fastly KV Store	Central store	Lazily cached at each point of presence	Eventual	Runtime-updated configuration and lists

Workers KV is itself a multi-tier cache: a read at a location is served from that location's cache when possible and otherwise falls through regional and central tiers; popular values are refreshed in the background; negative lookups are cached; and the documentation recommends Durable Objects where stronger consistency or frequent writes are needed [25]. A Durable Object gives each name a single instance with private, strongly consistent storage, so it can serve as the authoritative home for an entity at the edge [26]. Fastly's KV Store is cached lazily at each point of presence on first request, so the first read at a location is slower

than later ones [27]. The pattern across platforms is consistent: eventually consistent caches for reads, plus a strongly consistent home for writes and coordination.

8. Devices and offline-first caches

8.1 Local-first data

At the device level, the cache is often also a writable replica. Local-first software keeps the primary copy of data on the device, works offline and syncs with peers and servers when it can, treating the cloud as a secondary replica [28]. SQLite's own guidance describes it as suited to devices that run without administration and to a local "cache for enterprise data" that reduces latency and server load and keeps applications working during outages [29].

8.2 Conflict-free replicated data types

Offline writes on many devices conflict, and conflict-free replicated data types (CRDTs) resolve them without coordination. Shapiro et al. defined state-based and operation-based CRDTs and the guarantee they provide, strong eventual consistency: replicas that have received the same updates converge, regardless of order, without rollback [30], [31]. Kleppmann and Beresford extended the idea to nested JSON documents [32], and Almeida surveys the design approaches developed since, including delta-state and pure operation-based designs [33]. Open-source libraries such as Automerge and Yjs make CRDTs available to applications, with local storage and automatic merging when connectivity returns [34], [35]. CRDTs guarantee convergence, not intent: the merged result follows each data type's rule, and invariants that need coordination, such as uniqueness, cannot be guaranteed without it. Standard CRDTs also assume honest replicas; Kleppmann showed how to restore convergence when some replicas may be malicious, which matters for devices and edge nodes outside an operator's control [36].

8.3 Sync systems

Sync systems connect device caches to servers. Couchbase Lite replicates with a server-side Sync Gateway in push, pull or both directions, and retries when the remote is unreachable [37]. CouchDB replication is driven by a changes feed, resumes from checkpoints, and can produce conflicts that the application resolves [38]; PouchDB implements the same protocol in the browser so web applications can work offline [39]. PowerSync keeps an in-app SQLite database synchronized with a backend database such as PostgreSQL [40]. The device tier also carries platform risk: MongoDB retired its Atlas Device Sync service, which reached end of life after a deprecation period, leaving teams to replace the sync layer while keeping the local store [41].

9. IoT data

9.1 Edge tiers for sensor data

IoT systems generate data continuously at the edge of the network. Shi et al. argue that edge nodes should cache, filter and pre-process IoT data rather than ship all of it to the cloud [3], and surveys of edge caching organize the field around where to cache, what to cache and how [42], including caches at base stations, edge servers and devices in mobile networks [43].

Sensor readings are often transient, losing value after a limited lifetime, which makes caching them a distinct problem from caching content [44].

9.2 Freshness as the measure

For sensor data, freshness matters more than hit ratio. Age of Information measures, at the receiver, how long ago the newest value it holds was generated; Kaul, Yates and Gruteser showed that updating as fast as possible is not optimal, because queueing can make data staler [5], and Yates et al. survey the theory and its applications, including caching [45]. Pappalardo, Virdis and Mingozi apply it to IoT edge caches, setting refresh parameters to meet freshness requirements while minimizing the energy constrained devices spend sensing and transmitting [46]. The lesson for multi-level caches is to set freshness targets per data stream and measure age, not only hits.

9.3 Last-value caches

Simple last-value caches are common in IoT. MQTT retained messages keep the most recent message per topic at the broker for new subscribers, and a message expiry interval discards it when it becomes too old [47]. Time-series databases offer similar features: InfluxDB 3 keeps the latest values per series in memory so that "latest reading" queries avoid scanning stored data [48].

10. Consistency across levels

Every level of the hierarchy makes a consistency choice. Brewer's refinement of the CAP theorem notes that the choice between consistency and availability matters only during partitions, so systems can detect partitions, operate in an explicit partition mode and reconcile afterwards, for example with CRDT merges [49]. Abadi's PACELC adds that even without partitions, systems trade latency against consistency, which explains why geo-distributed and edge caches relax consistency when the network is healthy [1]. Between strong and eventual consistency lie guarantees that users notice directly. Terry et al. defined four session guarantees, read your writes, monotonic reads, writes follow reads and monotonic writes, that a client can obtain on top of weakly consistent replicas [50]; D1's session bookmarks and memcache's remote markers are implementations of such guarantees [21], [18]. Bailis et al. showed which transactional guarantees remain achievable while staying available under partitions, and which, such as serializability, do not [51]. COPS showed that causal consistency, stronger than eventual, can be provided for geo-replicated storage with local reads and writes [52]. For a multi-level cache, the practical step is to state, per tier and per data set, which of these guarantees a client gets and what it sees when it moves between tiers.

11. Security and data residency at the edge

Edge caches widen the attack surface. In cache-poisoned denial of service, an attacker sends a request that the cache forwards but the origin rejects, and the shared cache stores the error under the normal cache key, so legitimate users receive it [53]. In web cache deception, an attacker tricks a CDN into caching a victim's personalized response as if it were public, then retrieves it [54]. Both arise when the cache and origin interpret requests differently, and both are mitigated by strict cache keys, caching only responses explicitly marked cacheable and testing the cache and origin together. Data residency adds a regulatory

dimension: a globally distributed cache may store or decrypt regulated data where it should not, and CDN providers offer controls over where traffic is processed and where metadata and keys are kept [55]. At the device and peer level, untrusted replicas can corrupt shared state unless synchronization tolerates malicious participants [36].

12. Comparative analysis

Table 3 compares the approaches in this paper. The ratings are qualitative and relative to each other; actual behavior depends on the products, configuration, network and workload.

Approach	Read latency for distant users	Write path	Consistency offered	Offline tolerance	Operational effort
Core database only	High	Single primary	Strong	None	Low
CDN tiers in front of data APIs	Very low for cacheable responses	Origin	Freshness and validation rules; purge is asynchronous	Stale-if-error during origin failures	Medium
Regional caches over one primary	Low	Primary, with invalidation from the log	Eventual, plus session mechanisms	Low	Medium
Read replicas and follower reads	Low	Primary or leaseholder	Bounded staleness or session consistency	Low	Low to Medium
Multi-primary with CRDTs	Low	Local in every region	Strong eventual, optionally causal	Medium (regions continue during partitions)	Medium
Edge key-value and edge databases	Very low	Central store or per-object home	Eventual, per-object strong or per-session	Low to Medium	Low (managed)
Offline-first device caches	Very low (local)	Local, synchronized later	Strong eventual via CRDTs or application merges	High	Medium to High
IoT edge and last-value caches	Very low	Device to edge	Freshness bounded by refresh and expiry	Medium	Medium

13. Open challenges and future opportunities

13.1 Consistency and availability at the edge

Challenge. Edge and device tiers must stay available during partitions, which rules out some guarantees [49], [51]. **Research direction.** Causal and session guarantees over weak replicas [50], [52], and conflict-free data types with richer semantics [33]. **Opportunity.** Clear, per-tier guarantees that applications can rely on would make edge caching safe for more data.

13.2 Preserving intent when merging

Challenge. CRDTs guarantee convergence, not the result a user intended, and cannot enforce invariants that need coordination [30], [33]. **Research direction.** New data types and hybrid designs that coordinate only for the operations that need it [33], [51]. **Opportunity.** Merges that respect application rules would make offline-first applications practical for business data.

13.3 Placement and sizing across tiers

Challenge. Each tier sees a filtered miss stream, and the best placement depends on workload, latency and transfer costs [9], [2]. **Research direction.** Miss-ratio modeling for multi-tier caches [9], cost-driven self-configuration [2], edge caching frameworks [42], and learned placement, which paper 2 introduced for single caches. **Opportunity.** Tiers that size and place themselves from live traffic and prices would replace hand-tuned hierarchies.

13.4 Freshness and energy for IoT

Challenge. Refreshing sensor caches more often costs device energy and bandwidth, and refreshing as fast as possible can make data staler [5], [46]. **Research direction.** Age-of-Information-aware refresh and caching [45], [46]. **Opportunity.** IoT caches that meet stated freshness targets at minimum energy would extend device life and reduce network load.

13.5 Invalidation across many locations

Challenge. Purging or invalidating data at many edge locations is asynchronous and does not reach caches beyond the CDN [16]. **Research direction.** Tag-based purging [17], versioned resources [16], and the dependency-driven invalidation described in paper 5. **Opportunity.** Invalidation that reaches every tier predictably would let teams cache more dynamic data at the edge.

13.6 Security and residency

Challenge. Disagreements between cache and origin create attacks [53], [54], and caches can place regulated data in the wrong jurisdiction [55]. **Research direction.** Systematic testing of cache and origin behavior and Byzantine-tolerant synchronization for untrusted replicas [36]. **Opportunity.** Edge caches that are secure and residency-aware by default would widen the set of data that can be cached close to users.

13.7 Observability across the hierarchy

Challenge. Staleness and misses at one tier show up as symptoms at another, and few systems measure freshness end to end. **Research direction.** Freshness metrics such as Age of Information [45], consistency measurement as described in paper 5, and miss-ratio modeling across tiers [9]. **Opportunity.** End-to-end views of hit ratio, staleness and cost per tier would make multi-level caches operable.

13.8 Platform change at the edge

Challenge. Edge and device platforms evolve quickly, and services can be retired, as MongoDB's device sync service was [41]. **Research direction.** Open protocols and libraries for sync, such as CouchDB replication and open-source CRDT libraries [38], [34], [35]. **Opportunity.** Designs that separate the local store from the sync service reduce the cost of changing providers.

14. What this means for platform decisions

Map each data set to the tiers that need it and state, per tier, the freshness and consistency a client gets. Put CDN tiers and origin shields in front of cacheable API responses, with explicit freshness rules and tag-based purging. For regional reads, prefer read replicas or follower reads with session guarantees where users must see their own writes, and choose multi-primary replication only where local writes in every region justify conflict-free data types and their memory cost. Use edge key-value stores for read-heavy data, with a strongly consistent home for writes. For devices, design local-first with a clear merge model and keep the local store independent of any one sync provider. For IoT data, set freshness targets and measure age as well as hits. Test cache and origin together for security, and place regulated data deliberately.

For in-memory caching across regions, Bassirah works with Redis as a Strategic Technology Partner; Redis Active-Active provides multi-primary geo-distribution with conflict-free data types, with the trade-offs its documentation states. Other products and projects named in this paper are ecosystem references. In each case, validate the design on representative workloads, network conditions, failure scenarios and versions before relying on it.

This paper closes the Database Caching series. Paper 1 set out the strategies, architectures, eviction policies and pitfalls of a single cache; papers 2 to 6 examined learned policies, semantic caching for AI applications, memory and storage tiers, precise invalidation and, here, hierarchies across geography and devices. One lesson recurs: a cache is a second copy of the truth, and the design work is in deciding how far that copy may drift, then measuring that it does not drift further.

References

1. Abadi, D. J. (2012). Consistency Tradeoffs in Modern Distributed Database System Design: CAP is Only Part of the Story. *Computer (IEEE)*, 45(2), pp. 37–42. (<https://doi.org/10.1109/MC.2012.33>)
2. Park, H., Qiu, Z., Ganger, G. R. and Amvrosiadis, G. (2024). Reducing Cross-Cloud/Region Costs with the Auto-Configuring MACARON Cache. *Proceedings of the ACM SIGOPS 30th Symposium on Operating Systems Principles (SOSP '24)*, pp. 347–368. (<https://doi.org/10.1145/3694715.3695972>)
3. Shi, W., Cao, J., Zhang, Q., Li, Y. and Xu, L. (2016). Edge Computing: Vision and Challenges. *IEEE Internet of Things Journal*, 3(5), pp. 637–646. (<https://doi.org/10.1109/JIOT.2016.2579198>)
4. Satyanarayanan, M. (2017). The Emergence of Edge Computing. *Computer (IEEE)*, 50(1), pp. 30–39. (<https://doi.org/10.1109/MC.2017.9>)
5. Kaul, S., Yates, R. and Gruteser, M. (2012). Real-time status: How often should one update?. In *Proceedings of IEEE INFOCOM 2012*, pp. 2731–2735. (<https://doi.org/10.1109/INFCOM.2012.6195689>)
6. Che, H., Tung, Y. and Wang, Z. (2002). Hierarchical Web caching systems: modeling, design and experimental results. *IEEE Journal on Selected Areas in Communications*, 20(7), pp. 1305–1314. (<https://doi.org/10.1109/JSAC.2002.801752>)
7. Fan, L., Cao, P., Almeida, J. and Broder, A. Z. (2000). Summary cache: a scalable wide-area Web cache sharing protocol. *IEEE/ACM Transactions on Networking*, 8(3), pp. 281–293. (<https://doi.org/10.1109/90.851975>)

8. Wolman, A., Voelker, G. M., Sharma, N., Cardwell, N., Karlin, A. and Levy, H. M. (1999). On the scale and performance of cooperative Web proxy caching. Proceedings of the 17th ACM Symposium on Operating Systems Principles (SOSP '99), pp. 16–31.
(<https://doi.org/10.1145/319151.319153>)
9. Liu, Z., Lee, H. W., Xiang, Y., Grunwald, D. and Ha, S. (2021). eMRC: Efficient Miss Ratio Approximation for Multi-Tier Caching. 19th USENIX Conference on File and Storage Technologies (FAST '21), pp. 293–306.
(<https://www.usenix.org/conference/fast21/presentation/liu>)
10. Berger, D. S., Berg, B., Zhu, T., Sen, S. and Harchol-Balter, M. (2018). RobinHood: Tail Latency Aware Caching -- Dynamic Reallocation from Cache-Rich to Cache-Poor. 13th USENIX Symposium on Operating Systems Design and Implementation (OSDI 18), pp. 195–212.
(<https://www.usenix.org/conference/osdi18/presentation/berger>)
11. Nygren, E., Sitaraman, R. K. and Sun, J. (2010). The Akamai Network: A Platform for High-Performance Internet Applications. ACM SIGOPS Operating Systems Review, 44(3), pp. 2–19. (<https://doi.org/10.1145/1842733.1842736>)
12. Cloudflare (n.d.). Tiered Cache. Cloudflare Cache documentation. Official documentation, accessed 26 October 2026.
(<https://developers.cloudflare.com/cache/how-to/tiered-cache/>)
13. Amazon Web Services (n.d.). Use Amazon CloudFront Origin Shield. Amazon CloudFront Developer Guide. Official documentation, accessed 26 October 2026.
(<https://docs.aws.amazon.com/AmazonCloudFront/latest/DeveloperGuide/origin-shield.html>)
14. Fielding, R., Nottingham, M. and Reschke, J. (Eds.) (2022). HTTP Caching. RFC 9111, STD 98, Internet Engineering Task Force. (<https://doi.org/10.17487/RFC9111>)
15. Nottingham, M. (2010). HTTP Cache-Control Extensions for Stale Content. RFC 5861, Informational, Internet Engineering Task Force. (<https://doi.org/10.17487/RFC5861>)
16. Amazon Web Services (n.d.). Invalidate files to remove content. Amazon CloudFront Developer Guide. Official documentation, accessed 26 October 2026.
(<https://docs.aws.amazon.com/AmazonCloudFront/latest/DeveloperGuide/Invalidation.html>)
17. Fastly (n.d.). Working with surrogate keys; and Soft purges. Fastly Documentation, Full-site delivery guides. Official documentation, accessed 26 October 2026.
(<https://www.fastly.com/documentation/guides/full-site-delivery/purging/working-with-surrogate-keys/> ; <https://www.fastly.com/documentation/guides/full-site-delivery/purging/soft-purges/>)
18. Nishtala, R., Fugal, H., Grimm, S., Kwiatkowski, M., Lee, H., Li, H. C., McElroy, R., Paleczny, M., Peek, D., Saab, P., et al. (2013). Scaling Memcache at Facebook. 10th USENIX Symposium on Networked Systems Design and Implementation (NSDI '13), pp. 385–398.
(<https://www.usenix.org/conference/nsdi13/technical-sessions/presentation/nishtala>)
19. Bronson, N., Amsden, Z., Cabrera, G., Chakka, P., Dimov, P., Ding, H., Ferris, J., Giardullo, A., Kulkarni, S., Li, H., et al. (2013). TAO: Facebook's Distributed Data Store for the Social Graph. 2013 USENIX Annual Technical Conference (USENIX ATC '13), pp. 49–60.
(<https://www.usenix.org/conference/atc13/technical-sessions/presentation/bronson>)
20. Cockroach Labs (n.d.). Follower Reads. CockroachDB documentation (stable). Official documentation, accessed 26 October 2026.
(<https://docs.cockroachlabs.com/docs/stable/follower-reads>)

21. Cloudflare (n.d.). Global read replication. Cloudflare D1 documentation, Best practices. Official documentation, accessed 26 October 2026. (<https://developers.cloudflare.com/d1/best-practices/read-replication/>)
22. Upstash (n.d.). Upstash Redis Global Database. Upstash Documentation. Official documentation, accessed 26 October 2026. (<https://upstash.com/docs/redis/features/globaldatabase>)
23. Redis (n.d.). Active-Active geo-distributed Redis; and Develop applications with Active-Active databases. Redis Software documentation. Official documentation, accessed 26 October 2026. (<https://redis.io/docs/latest/operate/rs/databases/active-active/> ; <https://redis.io/docs/latest/operate/rs/databases/active-active/develop/develop-for-aa/>)
24. Redis (n.d.). Active-Active Redis. Redis Cloud documentation. Official documentation, accessed 26 October 2026. (<https://redis.io/docs/latest/operate/rc/databases/active-active/>)
25. Cloudflare (n.d.). How KV works. Cloudflare Workers KV documentation. Official documentation, accessed 26 October 2026. (<https://developers.cloudflare.com/kv/concepts/how-kv-works/>)
26. Cloudflare (n.d.). What are Durable Objects? Cloudflare Durable Objects documentation. Official documentation, accessed 26 October 2026. (<https://developers.cloudflare.com/durable-objects/what-are-durable-objects/>)
27. Fastly (n.d.). Data stores (Edge state concepts). Fastly Documentation. Official documentation, accessed 26 October 2026. (<https://www.fastly.com/documentation/guides/concepts/edge-state/data-stores/>)
28. Kleppmann, M., Wiggins, A., van Hardenberg, P. and McGranaghan, M. (2019). Local-first software: you own your data, in spite of the cloud. In Proceedings of the 2019 ACM SIGPLAN International Symposium on New Ideas, New Paradigms, and Reflections on Programming and Software (Onward! 2019), pp. 154–178. (<https://doi.org/10.1145/3359591.3359737>)
29. SQLite project (n.d.). Appropriate Uses For SQLite. Official documentation, accessed 26 October 2026. (<https://www.sqlite.org/whentouse.html>)
30. Shapiro, M., Preguiça, N., Baquero, C. and Zawirski, M. (2011). Conflict-Free Replicated Data Types. In Stabilization, Safety, and Security of Distributed Systems (SSS 2011), Lecture Notes in Computer Science, vol. 6976, Springer, pp. 386–400. (https://doi.org/10.1007/978-3-642-24550-3_29)
31. Shapiro, M., Preguiça, N., Baquero, C. and Zawirski, M. (2011). A comprehensive study of Convergent and Commutative Replicated Data Types. Research Report RR-7506, Inria. (<https://inria.hal.science/inria-00555588>)
32. Kleppmann, M. and Beresford, A. R. (2017). A Conflict-Free Replicated JSON Datatype. IEEE Transactions on Parallel and Distributed Systems, 28(10), pp. 2733–2746. (<https://doi.org/10.1109/TPDS.2017.2697382>)
33. Almeida, P. S. (2024). Approaches to Conflict-free Replicated Data Types. ACM Computing Surveys, 57(2), pp. 1–36. (<https://doi.org/10.1145/3695249>)
34. Automerge project (n.d.). Welcome to Automerge. Official documentation, accessed 26 October 2026. (<https://automerge.org/docs/hello/>)
35. Yjs project (n.d.). Introduction (Yjs documentation). Official documentation, accessed 26 October 2026. (<https://docs.yjs.dev/>)

36. Kleppmann, M. (2022). Making CRDTs Byzantine fault tolerant. In Proceedings of the 9th Workshop on Principles and Practice of Consistency for Distributed Data (PaPoC '22, held with EuroSys '22), ACM, pp. 8–15. (<https://doi.org/10.1145/3517209.3524042>)
37. Couchbase (n.d.). Data Sync using Sync Gateway (Couchbase Lite for Swift). Official documentation, accessed 26 October 2026. (<https://docs.couchbase.com/couchbase-lite/current/swift/replication.html>)
38. Apache Software Foundation (n.d.). Introduction to Replication (Apache CouchDB documentation, section 2.1). Official documentation, accessed 26 October 2026. (<https://docs.couchdb.org/en/stable/replication/intro.html>)
39. PouchDB project (n.d.). Replication (PouchDB guides). Official documentation, accessed 26 October 2026. (<https://pouchdb.com/guides/replication.html>)
40. PowerSync (n.d.). PowerSync Docs (overview). Official documentation, accessed 26 October 2026. (<https://docs.powersync.com/intro/powersync-overview>)
41. MongoDB, Inc. (n.d.). Device Sync Deprecation (Atlas App Services) and Atlas Device SDKs Deprecation. Official documentation, accessed 26 October 2026. (<https://www.mongodb.com/docs/atlas/app-services/sync/device-sync-deprecation/> ; <https://www.mongodb.com/docs/atlas/device-sdks/deprecation/>)
42. Li, H., Sun, M., Xia, F., Xu, X. and Bilal, M. (2024). A Survey of Edge Caching: Key Issues and Challenges. *Tsinghua Science and Technology*, 29(3), pp. 818–842. (<https://doi.org/10.26599/TST.2023.9010051>)
43. Yao, J., Han, T. and Ansari, N. (2019). On Mobile Edge Caching. *IEEE Communications Surveys & Tutorials*, 21(3), pp. 2525–2553. (<https://doi.org/10.1109/COMST.2019.2908280>)
44. Al-Ward, H., Tan, C. K. and Lim, W. H. (2022). Caching transient data in Information-Centric Internet-of-Things (IC-IoT) networks: A survey. *Journal of Network and Computer Applications*, 206, article 103491. (<https://doi.org/10.1016/j.jnca.2022.103491>)
45. Yates, R. D., Sun, Y., Brown, D. R., Kaul, S. K., Modiano, E. and Ulukus, S. (2021). Age of Information: An Introduction and Survey. *IEEE Journal on Selected Areas in Communications*, 39(5), pp. 1183–1210. (<https://doi.org/10.1109/JSAC.2021.3065072>)
46. Pappalardo, M., Viridis, A. and Mingozzi, E. (2022). Energy-Optimized Content Refreshing of Age-of-Information-Aware Edge Caches in IoT Systems. *Future Internet*, 14(7), article 197. (<https://doi.org/10.3390/fi14070197>)
47. Banks, A., Briggs, E., Borgendale, K. and Gupta, R. (eds.) (2019). MQTT Version 5.0. OASIS Standard, 7 March 2019. OASIS. (<https://docs.oasis-open.org/mqtt/mqtt/v5.0/os/mqtt-v5.0-os.html>)
48. InfluxData (n.d.). Manage the Last Value Cache (InfluxDB 3 Core). Official documentation, accessed 26 October 2026. (<https://docs.influxdata.com/influxdb3/core/admin/last-value-cache/>)
49. Brewer, E. (2012). CAP twelve years later: How the "rules" have changed. *Computer (IEEE)*, 45(2), pp. 23–29. (<https://doi.org/10.1109/MC.2012.37>)
50. Terry, D. B., Demers, A. J., Petersen, K., Spreitzer, M. J., Theimer, M. M. and Welch, B. B. (1994). Session guarantees for weakly consistent replicated data. In Proceedings of the 3rd International Conference on Parallel and Distributed Information Systems (PDIS 1994), IEEE, pp. 140–149. (<https://doi.org/10.1109/PDIS.1994.331722>)
51. Bailis, P., Davidson, A., Fekete, A., Ghodsi, A., Hellerstein, J. M. and Stoica, I. (2013). Highly available transactions: virtues and limitations. *Proceedings of the VLDB Endowment*, 7(3), pp. 181–192. (<https://doi.org/10.14778/2732232.2732237>)

52. Lloyd, W., Freedman, M. J., Kaminsky, M. and Andersen, D. G. (2011). Don't settle for eventual: scalable causal consistency for wide-area storage with COPS. In Proceedings of the 23rd ACM Symposium on Operating Systems Principles (SOSP '11), pp. 401–416. (<https://doi.org/10.1145/2043556.2043593>)
53. Nguyen, H. V., Lo Iacono, L. and Federrath, H. (2019). Your Cache Has Fallen: Cache-Poisoned Denial-of-Service Attack. In Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security (CCS '19), pp. 1915–1936. (<https://doi.org/10.1145/3319535.3354215>)
54. Mirheidari, S. A., Arshad, S., Onarlioglu, K., Crispo, B., Kirda, E. and Robertson, W. (2020). Cached and Confused: Web Cache Deception in the Wild. In 29th USENIX Security Symposium (USENIX Security 20), pp. 665–682. (<https://www.usenix.org/conference/usenixsecurity20/presentation/mirheidari>)
55. Cloudflare, Inc. (n.d.). Data Localization Suite. Official documentation, accessed 26 October 2026. (<https://developers.cloudflare.com/data-localization/>)

Product and project names are trademarks of their respective owners. Their mention describes publicly documented technology and does not imply a business relationship, except where Bassirah names a Strategic Technology Partner.