

Consistency–Availability Trade-offs in Distributed NoSQL Systems: From the CAP Theorem to Tunable and Strongly Consistent Cloud Data Stores

Naser Mislimi^{1*}, Violeta Manevska²

¹Information Sciences, Tetovo University, Tetovo, North Macedonia

²Faculty of Information and Communication Technologies – Bitola,
University “St. Kliment Ohridski” - Bitola, 7000 Bitola, North Macedonia

E-mail: naser.mislimi@unite.edu.mk; violeta.manevska@uklo.edu.mk

*Corresponding author details: Naser Mislimi; naser.mislimi@unite.edu.mk

ABSTRACT

The move of data management from centralized relational servers to geo-distributed, partitioned platforms forced designers to face a trade-off that classical transaction theory set aside: when the network fails, a replicated system cannot stay both consistent and available. This paper examines how the consistency–availability trade-off in distributed NoSQL systems, first stated in the CAP theorem and later refined by PACELC, has shaped two decades of NoSQL and NewSQL design. Building on the distributed database and NoSQL treatment of Elmasri and Navathe, the paper organizes the consistency spectrum from linearizability through causal and session guarantees to eventual consistency and traces it in eleven representative systems: the Dynamo lineage, Cassandra’s tunable consistency, and strongly consistent designs such as Spanner, Calvin, CockroachDB, and FoundationDB. Two research tables summarize the model hierarchy and a PACELC classification of the surveyed systems. The analysis shows that the field has not converged on one point in the design space; it has industrialized the whole spectrum and exposed consistency as a per-operation, application-controlled parameter. The paper closes with four open problems: accurate time, the industrial absence of causal consistency, semantic verification, and elasticity versus coordination. The decisive question for designers is no longer consistency or availability, but how much latency, complexity, and money a given guarantee is worth.

Keywords: NoSQL databases; distributed databases; CAP theorem; PACELC; eventual consistency; quorum replication; NewSQL; cloud data stores

INTRODUCTION

Textbook treatments of database systems, including Elmasri and Navathe [1], present distributed databases through fragmentation, replication, and distribution transparency: a distributed database should behave, from the application’s view, as one logical database. Designers tried to reach this goal through two-phase commit, distributed serializability, and synchronous replica updates, and in one data center, it worked. It was broken by web platforms in the early 2000s. They were serving hundreds of millions of users from thousands of machines on different continents, making partial failure the new normal and adding cross-region latency into the perception and revenue of users. From here, the NoSQL movement arose. Four defining features are listed by Elmasri and Navathe [1]: horizontal scalability, relaxed consistency for replicated data, flexible data models, and simplified data query interfaces. The second is the one with the most theory: relaxed consistency was not a “shortcut”, but a reaction to an impossibility result.

Brewer’s conjecture [2], proved by Gilbert and Lynch [3], states that it is impossible to guarantee Linearizability, Availability of all requests, and Partition Tolerance simultaneously in a replicated service. In the event of a failure, wide-area partitions will happen at some point, which means that you have to make a choice between consistency and availability: the CP versus AP dichotomy. This has been complicated by twenty years of practice. They found that, in fact, the “two out of three” slogan was misleading, as the more common trade-off occurs during normal operation, on each request, between latency and consistency, and Abadi’s PACELC refinement showed that this is the case. There were two directions in which industry shifted. Eventual consistency was put to the test by Amazon’s Dynamo [6]. Google’s Spanner [7] demonstrated external consistency across continents without the problem being prohibitively expensive, something that many people had thought was impractical. In between, Cassandra [8] has a consistency-per-operation dial and Amazon DynamoDB has an availability-first

design, with opt-in strong reads and distributed transactions [10]. This paper provides a critical survey of the consistency–availability trade-off in NoSQL systems. It has three contributions to make. First, it arranges the consistency spectrum [11–14] in a hierarchy based on the anomalies each level of consistency permits (Table 1) instead of a list of vendor slogans. Second, it categorizes eleven representative systems using the PACELC framework, instead of marketing descriptions (Table 2, Figure 1). Third, it lists unsolved problems that are not addressed in the textbook, such as empirical verification [15], coordination-free convergence [16], and the economics of synchronized time. The paper is organized as follows. The scope and method of the review are provided in the next section. The next sections cover the distributed database concepts which are the basis of the problem, the formulation of the consistency hierarchy, a survey of representative systems, and a comparative analysis with practical implications. The paper ends with open challenges, limitations, and conclusions. The scope of the review and the method used are described. The review is not based on a convenience sample. Three criteria were used for inclusion. First, every system had to be described by a peer-reviewed primary paper; MongoDB is the only exception; it was included in the systems through surveys that report its consistency options [17, 18] due to its market weight. Second, the range of the set was to span from AP to CP, and there was to be at least one research system at each end. Third, each system had to have influenced later designs in some way, either by citing it or by being adopted into industry.

There were 11 such systems. Two properties were selected from each primary source to be used for classification. PA occurs when the system continues to accept writes on both sides of the partition, and PC when it refuses or blocks writes on the minority side. EL is given if it has the default read and write paths without cross-replica coordination, and EC is given otherwise. Class is defined as a tunable system with a default class plus a qualifier. The highest standard guarantee is that which a client can ask for in a documented and supported interface: experimental modes do not count. The method is known to have a weakness; it relies on documentation. A discussion of the verification follows later in this paper, explaining why documentation must be tested from distributed database theory to the CAP impossibility.

The Classical Framework

According to the classical theory, there are three design dimensions [1]. Fragmentation determines the mode of splitting (horizontal, vertical, or mixed), and the location of the fragments. Replication determines the number of copies of each fragment and how changes are spread to the copies. How much of this machinery the application sees is determined by transparency, while distribution transparency is made up of location, replication, and fragmentation transparency, and the classical model treats this as a good thing. The first two are maintained much as they are in NoSQL systems.

Horizontal fragmentation is also called consistent hashing, popularized by Dynamo [6]. They place them in a way that is not transparent. The partition key and consistency level are passed into the application; otherwise, they would be hidden, and the application would have to pay the costs. The framework is completed by distributed concurrency control and atomic commitment. Distributed serializability is an extension of the theory of serializability in centralized systems, and atomic commitment is given by two-phase commit (2PC). It is not an inadvertent blocking under coordinator failures [1]. It is a symptom of the deeper cause of CAP being exposed, namely that any protocol that requires unanimous agreement will be halted if the network cannot achieve unanimity. This is an observation that was made earlier by Lamport [19] in 1978: in a distributed system, there is no “now” of the system. All the coordination protocols since then are attempts to combine one.

CAP and PACELC

However, Brewer made a conjecture about CAP as a keynote speaker at PODC [2]. Two years later, Gilbert and Lynch [3] showed that it was possible to define consistency as linearizability, availability as a response from every non-failing node, and partition tolerance as correctness in the presence of arbitrary message loss. The proof is simple: a write accepted on one side cannot be read by a write on the other side during a partition, so a write must block and therefore give up availability, or else it returns the old data, which means it gives up consistency. It has had more effect on the design of a system than its length. That influence has not been a healthy one in every instance.

Brewer's retrospective [4] points out that partitions exist but are uncommon, and that the decision between C and A occurs at the operation level, not the system level; and that all three properties are continuous, not binary. Kleppmann [20] claims that formal definitions are too strict to be satisfied in the real world, and that the terms lead to sloganeering rather than specification. The missing clause is given by PACELC [5], which states that: When a Partition happens, a system sacrifices Availability for Consistency; When it doesn't, it sacrifices Latency for Consistency. The else clause is more important, as it is always executed. Dynamo is PA/EL; Spanner-class is PC/EC, which pays a latency premium on each and every write. The vocabulary, now standard [17], provides the structure for Table 2.

Economic Causes and Consistency Trade-offs (BASE vs. ACID)

What was the economic cause of weak consistency in BASE? BASE (Basically Available, Soft state, eventually consistent) is the NoSQL equivalent of ACID. Pritchett [21] presented it as a deliberate engineering compromise: reduced consistency for an order-of-magnitude increase in throughput and availability. The case study was provided by Vogels [14] from Amazon. Most commerce applications operate with consistency, as they can

reconcile inventory when it is oversold, so the database doesn't need to cover guarantees that the business isn't using. Stonebraker [22] dissented. Most of the gains in performance that he said a single object, the most realistic and strongest guarantee of single effect is linearizability [11], which states that each operation seems to happen instantaneously between its call and return, with respect to real time. It is what most engineers refer to as "consistency," and it is the C in the CAP proof [3]. Strict serializability is an extension of it to provide the real-time constraint along with serializable isolation for multi-object transactions, and Spanner's NoSQL technology brings, he said, other factors, like bypassing the logging and locking overheads, and weak consistency gives developers a permanent responsibility for reconciliation. Both roles have proven themselves to be successful in a variety of ways. There are systems that allow us to resolve the disagreement on an operation-by-operation basis: these are tunable-consistency systems.

THE SPECTRUM OF CONSISTENCY MODELS

Strong models: The case of external consistency [7] provides it on a global scale. The key word is coordination. The cost of some communication between the replicas, or with a clock oracle, is on the critical path of every write; protocols can move this cost away from the critical path but not eliminate it. **Causal and Session Guarantees.** Causal consistency assures all operations appear in causal order to each process, but concurrent operations can be in different orders. It has special theoretical status, as it is the strongest model that remains available during partitions [23]. Terry et al. [13] divide this middle ground into four guarantees for a single client: read-your-writes, monotonic reads, monotonic writes, and writes-follow-reads. They're affordable and eliminate the most noticeable irregularities. A user who posts a comment and then cannot see it suffers a read-your-write violation, not an abstract defect.

Viotti and Vukolić [12] catalog more than fifty named non-transactional models in a formal lattice. The catalog is a scholarly achievement and a symptom of the problem: practitioners struggle to tell what a given store promises. Bailis et al. [23] carried the analysis to transactions. Read committed isolation and atomic multi-item visibility are achievable with high availability; serializability and snapshot isolation are not. Their result draws the exact boundary of what partitions permit, and it shows that many systems settle for weaker guarantees than the impossibility results require.

Eventual and Bounded Models

Eventual consistency [14] promises only that replicas converge once updates stop. The definition is liveness-only; it constrains nothing about what reads observe in the meantime, and Bailis and Ghodsi [24] note that a system that always returns the value 42 satisfies it. Convergence machinery rescues the model in practice: version vectors to detect concurrent updates, anti-entropy with Merkle trees to repair divergent replicas [6] and, above all, conflict-free replicated data types (CRDTs) [16]. By restricting operations to commutative ones, or states to a join-semilattice, CRDTs converge without coordination and turn conflict resolution into a verified property of the data type. Counters, sets, and collaborative-editing sequences built as CRDTs now run in production. A further family bounds staleness quantitatively. Bounded staleness guarantees that reads observe a state no older than a fixed interval or version count. Snapshot isolation guarantees that each transaction reads a consistent, possibly stale, cut of the database. It is cheap over multi-version storage, so most NewSQL systems offer it by default [25], and it forbids dirty and non-repeatable reads, leaving only write skew. These models map onto service-level agreements. A promise of "staleness under five seconds" is contractible in a way that "eventual" is not, and cloud providers price consistency levels accordingly [9].

TABLE 1: Consistency Model Hierarchy and Permitted Anomalies.

Model	Core guarantee	Anomalies permitted	Available in partition
Strict serializability [7]	Serializable transactions in real-time order	None	No
Linearizability [11]	Atomic single-object operations in real-time order	None (single object)	No
Serializability	Equivalent to some serial order	Stale reads relative to real time	No
Snapshot isolation [25]	Reads from a consistent snapshot	Write skew	No
Bounded staleness	Reads lag by at most Δt or k versions	Stale reads within the bound	No
Causal [19], [23]	Happened-before order preserved	Concurrent writes seen in different orders	Yes (sticky)
Session guarantees [13]	Read-your-writes, monotonic reads and writes, writes-follow-reads	Staleness across sessions	Yes (sticky)
Read committed [23]	No dirty reads	Non-repeatable reads, lost updates	Yes
Eventual [14]	Convergence once updates stop	Arbitrary stale and reordered reads	Yes

Note: Sticky availability requires each client to stay with one replica set [23].

Reading the Hierarchy Correctly

Two cautions apply to Table 1. First, models compose badly across abstraction layers: a linearizable store beneath a cache with no guarantees yields a system with no guarantees. Second, the documented model can diverge from the delivered one under fault injection [15]. The hierarchy is a specification vocabulary, not a description of reality. Reality must be tested.

CONSISTENCY IN PRACTICE: REPRESENTATIVE SYSTEMS

The Dynamo Lineage: Availability First

The founding document of the AP tradition and one of the most cited systems papers of its time is Dynamo [6]. It was designed with honesty: the shopping cart needs to support writes during network failures, because a rejected write is a lost write. Consistent hashing partitioned the data, sloppy quorums provided write persistence in the event that the home replicas were unavailable, vector clocks were used to store multiple versions of data, and reconciliation was performed on read operations. If there are conflicts, the system returns all conflicts, and the cart combines ALL conflicts using a set union operation; that is a documented and accepted anomaly.

Cassandra [8] was developed at Facebook and has become an Apache project, which combined the replication from Dynamo with the wide-column model of Bigtable and added tunable consistency per operation. For every read and write, a client selects a level between 1 and ALL. The common case is achieved by quorum intersection with a read quorum R and a write quorum W where $R + W > N$ for N being the replication factor, and the client gives up on quorum when latency is a bigger priority. This design took the CAP tradeoff from an architectural commitment to a runtime parameter. It's arguably the most consequential API decision in the history of NoSQL; it acknowledges that operations in one application exist at various points across the spectrum.

The re-architected DynamoDB [9] is as far from its 2007 namesake as you can get. It isn't based on sloppy quorums and has predictable single-digit millisecond latency at scale, with eventually consistent and strongly consistent reads being first-class options; according to the paper, it delivered 89.2 million requests per second during peak retail events. Idziorek et al. [10] designed one-shot ACID transactions that are supported by a timestamp ordering protocol that ensures that transactional traffic does not negatively impact non-transactional latency. The flagship AP system has spent fifteen years re-acquiring, in opt-in form, the guarantees it first discarded.

The Bigtable Lineage: Single-Master Order

Bigtable [26] represents the contemporaneous CP-leaning tradition: a sparse, sorted map range-partitioned into tablets, each served by exactly one tablet server, with Chubby for coordination. Single

ownership gives strongly consistent single-row transactions without version vectors or read repair; the cost is brief unavailability while a failed server's tablets move. HBase copies this design in the Hadoop ecosystem, and the pattern of strong consistency within a partition and no transactions across partitions became the default for a generation of scale-out stores. MongoDB uses a replica set in which a primary serializes writes. Its move from weak default write concerns to majority-acknowledged writes and causally consistent sessions mirrors the industry drift toward stronger defaults [17], [18].

The Strong-Consistency Counter-Movement

Spanner [7] overturned the post-CAP assumption that strong global consistency was off the table. TrueTime exposes clock uncertainty as an interval, and the system waits out that interval before making writes visible, so timestamp order matches real-time order. Paxos per share and 2PC across shards then give externally consistent transactions across data centers. The insight is economic as much as technical: GPS receivers and atomic clocks keep uncertainty in single-digit milliseconds, so the wait is commercially tolerable. Spanner is openly CP. Its authors argue that, with enough network engineering, the availability lost to rare partitions is smaller than the availability lost to ordinary operational causes.

Calvin [27] reaches strong guarantees through determinism. It sequences all transactions into a replicated log before execution, so every replica reaches the same outcome without a distributed commit protocol. The price is interactivity: transactions must be known in advance. The design marks an important point in space, although its direct industrial adoption, mainly FaunaDB, has stayed modest.

The NewSQL generation industrialized these ideas without exotic hardware. CockroachDB [28] layers SQL over a Raft-replicated [29], range-partitioned key-value store and provides serializable isolation with hybrid logical clocks, explicitly for deployments that cannot assume TrueTime. TiDB [25] combines a Raft-replicated row store with a columnar replica to serve hybrid transactional-analytical workloads from the same consistent data. FoundationDB [30] deserves attention for two reasons. Its unbundled architecture separates transaction ordering, logging, and storage so each scales independently under strict serializability. Its deterministic simulation framework, which runs the whole distributed system with injected faults inside one single-threaded simulator, has become the reference standard for credible consistency claims.

Coordination Avoidance as a Research Frontier

Anna [31] marks the endpoint of the coordination-avoidance line. Its partitions share nothing, not even memory, and exchange lattice-based state through asynchronous messages. It scales near-linearly from one multicore machine to a geo-distributed cluster and reports throughput orders of magnitude beyond

conventional designs on cache-friendly workloads, but only for the consistency levels, up to causal, that lattices can express. Anna is a research system, yet it makes the cost of coordination vivid: everything above causal consistency sits on a steeper price curve.

COMPARATIVE ANALYSIS AND DISCUSSION

A PACELC Classification

Table 2 classifies the surveyed systems in PACELC terms [5], with the replication mechanism and the strongest standard guarantee of each, and Figure 1 places the systems with primary papers on a timeline. Three observations follow. First, the population drifts toward PC/EC over time. Apart from Bigtable, the early influential designs took the AP position; after 2012, nearly every new general-

purpose system took the CP position, and the largest AP incumbent has grown opt-in strong guarantees (Figure 1). The AP position was not wrong. Consensus protocols, clock infrastructure, and engineering practice improved enough to push the latency cost of consistency below the pain threshold of many applications, a quantitative shift that “two out of three” cannot express [4], [20]. Second, tunability has won in both camps. Cassandra, MongoDB and DynamoDB all expose consistency per operation, which supports the view that consistency is an application property rather than a system property [5], [24]. Third, no production system in the table leads with causal consistency despite its theoretical status [23]. The middle of the spectrum remains industrially underpopulated.

TABLE 2: PACELC Classification of Surveyed Systems.

System	PACELC	Replication	Strongest standard guarantee
Dynamo [6]	PA/EL	Sloppy quorum	Eventual, app merge
Cassandra [8]	PA/EL (tunable)	Quorum (R, W)	Per-operation quorum
DynamoDB [9]	PA/EL (opt-in C)	Multi-Paxos	Linearizable read; ACID transactions [10]
Bigtable [26]	PC/EC	Single master	Single-row transactions
MongoDB	PC/EC (tunable)	Replica set	Causal sessions
Spanner [7]	PC/EC	Paxos + TrueTime	External consistency
Calvin [27]	PC/EC	Ordered log	Serializability
CockroachDB [28]	PC/EC	Raft	Serializable isolation
TiDB [25]	PC/EC	Raft	Snapshot isolation
FoundationDB [30]	PC/EC	Unbundled log	Strict serializability
Anna [31]	PA/EL	Lattice gossip	Causal (coordination-free)

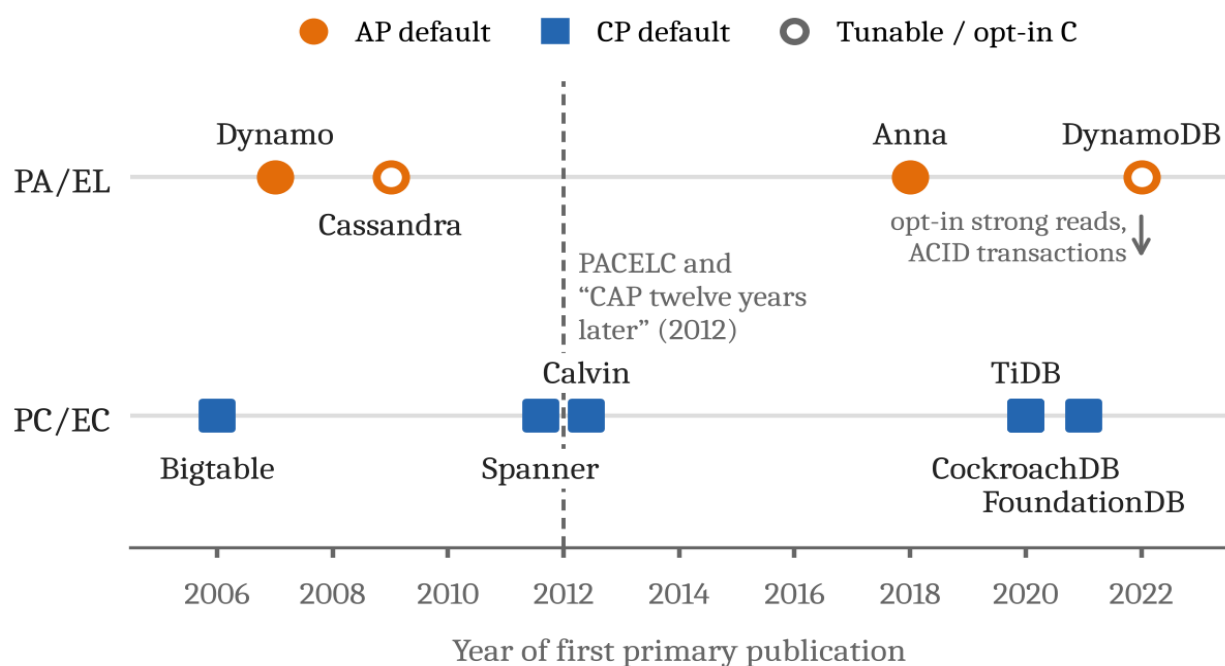


FIGURE 1: Timeline of Surveyed Systems by Default PACELC Class.

Note: Markers show the year of each system's first primary publication; Spanner and Calvin (both 2012) are offset for legibility. MongoDB is omitted because no primary system paper exists.

The drift can be seen in Figure 1. The PA/EL lane has the 2007 to 2009 designs that kick-started the NoSQL movement, plus another research system (Anna) that has been built from the ground up to be a coordination-free goal. After the year 2012, the PC/EC lane became full as Brewer's retrospective [4], and Abadi's PACELC paper [5] restated the debate, and Spanner [7] demonstrated global strong consistency in production. The timeline is no evidence for causation; engineering progress in consensus and clocks was simultaneous. It does indicate that the post-2012 generation believed that strong consistency was a given and was engineered, not abandoned. The Arithmetic of Quorums. The condition $R + W > N$ can be interpreted as a linearizability guarantee. It is not. In this case, $N = 3$, $R = W = 2$, and there is no failure; thus there is an intersection guarantee. In case of failure, it allows one replica failure for both reads and writes.

Quorum intersection failure in sloppy quorums [6] is designed to fail during partitions, and even strict quorums permit non-linearizable executions in the presence of concurrent writes while the writes are not read-repaired carefully [15],[24]. Experimental research with the consistency levels used in NoSQL stores has verified that there are staleness and latency costs that vary with load, topology, and configuration [32]. Quorum arithmetic only bounds staleness probabilistically, not remove it. Applications requiring a guarantee (not a tendency) should be required to implement explicitly linearizable primitives at a higher latency price point in these systems, like compare-and-set, lightweight transactions, and consensus-based reads. That premium is PACELC's else-clause collecting its fee. A Worked Example: Configuring Quorums. By simple calculation, quorum parameters are translated into availability and latency. With $N = 3$, the configuration $R = W = 1$ can only tolerate two failures on either the read or write path, and it allows for the lowest latency but doesn't provide any intersections, so if the latest write fails, the read might not reach it. Reads are fast and intersecting, with a single unreachable replica that prevents any writing. $R = W = 2$ is the balanced point: survives one failure on both paths and maintains intersection. When the number of replicas $R = W = 3$, $N = 5$ survives two failures on each path at the expense of an increased number of replicas contacted. The latency part of the same computation is modified by geography. Suppose there are 3 copies, one of which is local to the client. A write whose $W = 1$ finishes at local latency. If a write operation is given a write value of 2, it has to wait for the closest remote region, with a latency of one cross-region round-trip plus the latency for the inside of the continent, which is 10–20 ms, and between the United States and Europe, which is 70–100 ms. Write $W = 3$ for the last region.

The numbers available remain constant, but the latency numbers do depend on placement, and so operators of geo-distributed stores put more effort into quorum formula than into placement. The example also makes it clear that there is a limit of

tunability, as no setting provides intersection, partition survival, and local latency at the same time. On each operation, the client selects two of them, which is PACELC, which is based on arithmetic. The Geo-Replication's Latency Ledger: Rough but instructive numbers can be provided for the else clause. Within one data center, the Paxos or Raft round trip between replicas only takes a millisecond, making intra-region strong consistency now cheap compared to storage and network costs [28], [30]. Physics control takes over across regions: A quorum that is distributed across the United States pays about 70-100ms of round-trip time per coordinated write, and a globally distributed quorum pays twice that. Spanner's commit wait is added on top, and is only tolerable because TrueTime has it in single-digit milliseconds [7].

These constants are more effective in describing the architecture of each of the systems in Table 2 than any qualitative principle. The first-order design decision is where to place the leader: if the leader is remote, then writes will need to incur cross-region cost. Follower reads, and bounded-staleness reads are included to avoid read traffic getting stuck in the quorum path. Cassandra and DynamoDB go on sale because any user and any load test will see a difference of two orders of magnitude between the AP configurations, if a local write takes 1ms and the coordinated one takes 150ms. There's a consistency price list in round trips, and the systems that grow allow applications to read the price list before ordering.

Trust but Verify: Empirical Consistency Checking

The NoSQL decade revealed a discrepancy between promised and actual guarantees. But Jepsen's methodology evolved into Elle [15] – a polynomial-time checker for consistency violations from client-observed histories of production databases – and the former evolved into a series of production databases that are vulnerable to consistency violations. In a similar approach to FoundationDB's simulation-first practice [30], they approach the problem from the developer perspective by injecting faults prior to release instead of after. Both external checking and internal simulation have brought to the industry a standard of evidence that makes the consistency model more than a marketing claim. That is a standard any credible future system paper will have to live up to. Implications for Data-Intensive Architecture: These storage-level trade-offs are put into the context of the larger context of data-intensive systems by Margara et al. [17], and they note that consistency decisions take place at higher levels, in stream processors, caches, and above the database, in application-level sagas.

A strongly consistent store underneath an eventually consistent pipeline is an eventually consistent application, while CRDT-based application layers [16] are capable of providing convergence on top of a weak store. In the surveys of NoSQL systems [18, 33, 34], the primary reasons for choosing a NoSQL system are operational: it offers managed availability, ecosystem, and cost.

Consistency semantics is not the main reason for choosing a NoSQL system. The granularity of the research community's distinctions [12] is thus greater than the granularity of adoption decisions. Closing the gap in communication is a research problem in itself. Design Guidance for Practitioners: The analysis results in four practical rules. Keep in mind that it is operations, not applications, that are being classified. Each service has a different anomaly tolerance for its checkout write, balance read, and page-view counter, each of which is tunable. Second, begin with the name of the anomaly, NOT the model name. A team that can point to an anomaly that they will not accept can choose the lowest level in Table 1 that prevents them from accepting it; e.g., they can accept a lost update, but they cannot accept a stale balance, so they choose the lowest level that doesn't allow a stale balance. Thirdly, set up leaders prior to tuning quorums. Most protocol-level tuning is overcome by making a decision on who gets the leader. Fourth, demand evidence. A vendor claim of serializability or causal consistency only counts if published fault-injection results are offered [15] or a simulation approach similar to FoundationDB's is provided [30].

The rules are illustrated by two types of workloads. Read-heavy services that can tolerate bounded staleness benefit the most from follower reads and session guarantees, which hide the local latency level anomalies. Read-heavy services that can tolerate bounded staleness are more likely to benefit from the removal of the visible anomalies at the local latency level with follower reads and session guarantees. Invariants that extend over multiple objects (e.g., stocks that must not become negative) require coordinated transactions, or some kind of CRDT design that explicitly proves that the invariant is preserved by concurrent updates. There are no single global consistency settings for either pattern.

OPEN CHALLENGES AND FUTURE DIRECTIONS

There are four outstanding and pressing issues. The first is true time on each deployment. In Spanner, external consistency depends on the use of proprietary clock infrastructure [7], while CockroachDB's hybrid logical clocks do not rely on it, but at the penalty of poorer real-time guarantees of external consistency provided by clocks in the face of skew [28]. Finally, public clouds now offer high-accuracy time services, and a tight connection between bounded, non-TrueTime uncertainty and achievable guarantees and their tail latency cost is not currently available. The second is the lack of causal consistency in the industrial world. It's known by various names: the strongest always-available model [23], CRDTs make convergence free [16], and session guarantees are thirty years old [13], but no major cloud store leads with causal semantics. The gap may be due to metadata overhead in version vectors at hyperscale, complexity of interfaces, and the lack of a compelling managed product. It is an empirical question, and the answer has not been given by the community. The third is semantic-level verification. There are a few examples of checking the isolation of key-value

histories, including Elle [15] and FoundationDB's simulation of its own implementation [30]. Neither enforces end-to-end invariants like no negative balances, no double booking over stores, caches, and queues. An open problem of great value is to extend the notion of checkable consistency from storage interfaces to application invariants, possibly in combination with CRDT composition [16]. The fourth is Elasticity versus Coordination in Serverless. The fourth is Elasticity versus Coordination in Serverless. The compute and storage separation [9] and the ability to scale up and down automatically in a few seconds are enabled by a disaggregated architecture, whereas a consensus group and a transaction manager maintain state that is resistant to fast reconfiguration [30]. Anna [31] only avoids the conflict by abandoning the consistency of strong logic. If second-scale elasticity becomes controllable and serializable transactions or the trade-off itself becomes fundamental, then PACELC [5] has a third axis: "partition, latency and elasticity". All four issues are honed by emerging workloads. There is a difference between machine-learning feature stores, which allow for the presence of stale data, and unbounded staleness. In edge computing, the number of replicas increases massively, and the reliability per replica decreases. New quorum placement restrictions arise from privacy regulations that are not what the classical fragment allocation literature [1] predicted.

CONCLUSIONS

From the distributed database and NoSQL model of Elmasri and Navathe [1] that introduced the consistency – availability trade-off formally, this paper followed its development to PACELC and 20 years of production systems. The trajectory of events is obvious. The first generation of NoSQLs, Dynamo and Cassandra, read CAP as a license to throw away strong consistency, and they gained availability and scale that relational databases of the era couldn't match. The counter-movement, from Spanner, through CockroachDB and FoundationDB, was sufficient to make the latency cost of strong consistency an engineering variable, not a constant, and bring it to the point where serializable geo-distributed transactions became a commodity. It is the largest AP descendant service's interface, and the default philosophy for the NewSQL generation, and the synthesis- consistency as a tunable per-operation, priced parameter- has won. The theory lesson is more focused. CAP responded to a question regarding the limits of the possible; PACELC to the cost of the desirable. A causal middle ground and elastic coordination asks how these prices can be verified, reduced, and rebalanced, and the current frontier of verified guarantees is what these questions lead to. The consistency model is now an architect's choice, an operation-by-operation choice, and one that is dependent on anomaly tolerance and latency budgets, not the fidelity of a database family. The good spots for the researcher are not at either end of the spectrum, which industry has already developed, but in the apparently under-developed middle ground and the tools that render any point on it trustworthy.

REFERENCES

- [1] Elmasri, R., & Navathe, S. B. (2016). *Fundamentals of Database Systems* (7th ed.). Hoboken, NJ, USA: Pearson.
- [2] Brewer, E. A. (2000). Towards robust distributed systems. *Proceedings of the 19th ACM Symposium on Principles of Distributed Computing (PODC)*, p. 7. Portland, OR, USA.
- [3] Gilbert, S., & Lynch, N. (2002). Brewer's conjecture and the feasibility of consistent, available, partition-tolerant web services. *ACM SIGACT News*, 33(2), 51–59.
- [4] Brewer, E. (2012). CAP twelve years later: How the “rules” have changed. *Computer*, 45(2), 23–29.
- [5] Abadi, D. J. (2012). Consistency tradeoffs in modern distributed database system design: CAP is only part of the story. *Computer*, 45(2), 37–42.
- [6] DeCandia, G., et al. (2007). Dynamo: Amazon's highly available key-value store. *Proceedings of the 21st ACM Symposium on Operating Systems Principles (SOSP)*, 205–220.
- [7] Corbett, J. C., et al. (2013). Spanner: Google's globally distributed database. *ACM Transactions on Computer Systems*, 31(3), Article 8.
- [8] Lakshman, A., & Malik, P. (2010). Cassandra: A decentralized structured storage system. *ACM SIGOPS Operating Systems Review*, 44(2), 35–40.
- [9] Elhemali, M., et al. (2022). Amazon DynamoDB: A scalable, predictably performant, and fully managed NoSQL database service. *Proceedings of the USENIX Annual Technical Conference (ATC)*, 1037–1048.
- [10] Idziorek, J., et al. (2023). Distributed transactions at scale in Amazon DynamoDB. *Proceedings of the USENIX Annual Technical Conference (ATC)*, 705–717.
- [11] Herlihy, M. P., & Wing, J. M. (1990). Linearizability: A correctness condition for concurrent objects. *ACM Transactions on Programming Languages and Systems*, 12(3), 463–492.
- [12] Viotti, P., & Vukolić, M. (2016). Consistency in non-transactional distributed storage systems. *ACM Computing Surveys*, 49(1), Article 19.
- [13] Terry, D. B., Demers, A. J., Petersen, K., Spreitzer, M. J., Theimer, M. M., & Welch, B. B. (1994). Session guarantees for weakly consistent replicated data. *Proceedings of the 3rd International Conference on Parallel and Distributed Information Systems (PDIS)*, 140–149.
- [14] Vogels, W. (2009). Eventually consistent. *Communications of the ACM*, 52(1), 40–44.
- [15] Kingsbury, K., & Alvaro, P. (2020). Elle: Inferring isolation anomalies from experimental observations. *Proceedings of the VLDB Endowment*, 14(3), 268–280.
- [16] Shapiro, M., Preguiça, N., Baquero, C., & Zawirski, M. (2011). Conflict-free replicated data types. *Proceedings of the 13th International Symposium on Stabilization, Safety, and Security of Distributed Systems (SSS)*, 386–400.
- [17] Margara, A., Cugola, G., Felicioni, N., & Cilloni, S. (2024). A model and survey of distributed data-intensive systems. *ACM Computing Surveys*, 56(1), Article 2.
- [18] Davoudian, A., Chen, L., & Liu, M. (2018). A survey on NoSQL stores. *ACM Computing Surveys*, 51(2), Article 40.
- [19] Lamport, L. (1978). Time, clocks, and the ordering of events in a distributed system. *Communications of the ACM*, 21(7), 558–565.
- [20] Kleppmann, M. (2015). A critique of the CAP theorem. *arXiv:1509.05393 [cs.DC]*.
- [21] Pritchett, D. (2008). BASE: An ACID alternative. *ACM Queue*, 6(3), 48–55.
- [22] Stonebraker, M. (2010). SQL databases v. NoSQL databases. *Communications of the ACM*, 53(4), 10–11.
- [23] Bailis, P., Davidson, A., Fekete, A., Ghodsi, A., Hellerstein, J. M., & Stoica, I. (2013). Highly available transactions: Virtues and limitations. *Proceedings of the VLDB Endowment*, 7(3), 181–192.
- [24] Bailis, P., & Ghodsi, A. (2013). Eventual consistency today: Limitations, extensions, and beyond. *Communications of the ACM*, 56(5), 55–63.
- [25] Huang, D., et al. (2020). TiDB: A Raft-based HTAP database. *Proceedings of the VLDB Endowment*, 13(12), 3072–3084.
- [26] Chang, F., et al. (2008). Bigtable: A distributed storage system for structured data. *ACM Transactions on Computer Systems*, 26(2), Article 4.
- [27] Thomson, A., Diamond, T., Weng, S.-C., Ren, K., Shao, P., & Abadi, D. J. (2012). Calvin: Fast distributed transactions for partitioned database systems. *Proceedings of the ACM SIGMOD International Conference on Management of Data*, 1–12.

- [28] Taft, R., et al. (2020). CockroachDB: The resilient geo-distributed SQL database. Proceedings of the ACM SIGMOD International Conference on Management of Data, 1493–1509.
- [29] Ongaro, D., & Ousterhout, J. (2014). In search of an understandable consensus algorithm. Proceedings of the USENIX Annual Technical Conference (ATC), 305–319.
- [30] Zhou, J., et al. (2021). FoundationDB: A distributed unbundled transactional key-value store. Proceedings of the ACM SIGMOD International Conference on Management of Data, 2653–2666.
- [31] Wu, C., Faleiro, J. M., Lin, Y., & Hellerstein, J. M. (2021). Anna: A KVS for any scale. IEEE Transactions on Knowledge and Data Engineering, 33(2), 344–358.
- [32] Ferreira, S., Mendonça, J., & Andrade, E. (2025). Experimental performance analysis of data consistency levels in NoSQL databases. Software: Practice and Experience, 55(6), 1059–1070.
- [33] Gessert, F., Wingerath, W., Friedrich, S., & Ritter, N. (2017). NoSQL database systems: A survey and decision guidance. Computer Science – Research and Development, 32(3–4), 353–365.
- [34] Cattell, R. (2010). Scalable SQL and NoSQL data stores. ACM SIGMOD Record, 39(4), 12–27.