

# Quantifying Eventual Consistency with PBS

By Peter Bailis, Shivaram Venkataraman, Michael J. Franklin, Joseph M. Hellerstein, and Ion Stoica

## Abstract

Data replication results in a fundamental trade-off between operation latency and consistency. At the weak end of the spectrum of possible consistency models is eventual consistency, which provides no limit to the staleness of data returned. However, anecdotally, eventual consistency is often “good enough” for practitioners given its latency and availability benefits. In this work, we explain this phenomenon and demonstrate that, despite their weak guarantees, eventually consistent systems regularly return consistent data while providing lower latency than their strongly consistent counterparts. To quantify the behavior of eventually consistent stores, we introduce Probabilistically Bounded Staleness (PBS), a consistency model that provides expected bounds on data staleness with respect to both versions and wall clock time. We derive a closed-form solution for version-based staleness and model real-time staleness for a large class of quorum replicated, Dynamo-style stores. Using PBS, we measure the trade-off between latency and consistency for partial, non-overlapping quorum systems under Internet production workloads. We quantitatively demonstrate how and why eventually consistent systems frequently return consistent data within tens of milliseconds while offering large latency benefits.

## 1. INTRODUCTION

Modern distributed data stores need to be scalable, highly available, and fast. These systems typically replicate data across different machines and increasingly across datacenters for at least two reasons: first, to provide availability when components fail and, second, to provide improved performance by serving requests from multiple replicas. Configuring and maintaining replicated data has significant consequences for application and data store design.<sup>1</sup> Performance at scale is critical for a large class of applications and, in practice, increased latencies may correspond to large amounts of lost revenue.<sup>22</sup> For example, at Amazon, 100 ms of additional latency resulted in a 1% drop in sales,<sup>15</sup> while 500ms of additional latency in Google’s search resulted in a corresponding 20% decrease in traffic.<sup>16</sup> However, lowering latency in distributed data stores has a cost: contacting fewer replicas for each operation can adversely impact achievable semantic guarantees.

To provide predictably low latency, modern systems often eschew protocols guaranteeing “strong” consistency of reads (e.g., the illusion of a single copy of replicated data) and instead opt for “weaker” semantics, frequently in the form of *eventual consistency*.<sup>1, 5, 7, 10</sup> This eventual consistency is one of the weakest properties

provided by modern stores: it provides no guarantees on data staleness except that, in the absence of new writes, reads will “eventually” return the effect of the most recent write(s).<sup>26</sup> Under this definition, a store that returns data that is weeks old is eventually consistent, as is a store that returns arbitrary data (e.g., always return value 42) as long as, at *some point* in the future, the store returns the last written data.<sup>4</sup> Due to this near-absence of useful semantics for end users, the decision to employ eventual consistency is often controversial.<sup>12, 23, 24</sup> In the many production stores providing eventual consistency today,<sup>10, 14</sup> users have little to no insight into the behavior of their stores or the consistency of their data, especially under varying replication configurations. However, the proliferation of eventually consistent deployments suggests that applications can often tolerate occasional staleness and that data tends to be “fresh enough” in many cases.

In this work, we bridge this gap between theoretical guarantees and current practice by quantifying the degree to which eventual consistency is both eventual and (in) consistent and explain why. Indeed, under worst-case conditions, eventual consistency results in an unbounded degree of data staleness. However, as we will show, the common case is often different. Core to our thesis is the observation that eventual consistency can be modeled as providing a probabilistic *expectation* of consistency given a particular workload and deployment environment. Accordingly, for varying degrees of certainty, eventually consistent stores can offer bounds on how far they may deviate from strongly consistent behavior. We present probabilistic models for such bounds called Probabilistically Bounded Staleness, or PBS.

To predict consistency with PBS, we need to know when and why eventually consistent systems return stale data and how to quantify the staleness of the data they return. In this paper, we present algorithms and models for two common staleness metrics in the literature: wall clock time<sup>21</sup> and versions.<sup>27</sup> PBS describes both measures, providing the probability of reading a write  $\Delta$  seconds after

The original version of this paper is entitled “Probabilistically Bounded Staleness for Practical Partial Quorums” and was published in *VLDB 2012*.<sup>5</sup> An invited extended version of this paper is entitled “Quantifying Eventual Consistency with PBS” and will appear in the *VLDB Journal’s* “Best of VLDB 2012” issue in 2014.<sup>7</sup> Portions of this work also appear in a *SIGMOD 2013* demo entitled “PBS at Work: Advancing Data Management with Consistency Metrics.”<sup>6</sup>

the write returns  $((\Delta, p)$ -semantics, or “how eventual is eventual consistency?”), of reading one of the last  $K$  versions of a data item  $((K, p)$ -semantics, or “how consistent is eventual consistency?”), and of experiencing a combination of the two  $((K, \Delta, p)$ -semantics). PBS does not propose new mechanisms to enforce deterministic staleness bounds;<sup>27</sup> instead, our goal is to provide a lens for analyzing, improving, and predicting the behavior of *existing*, widely deployed systems.

In this work, we apply PBS to quorum-replicated data stores such as Dynamo<sup>10</sup> and its several open-source descendants. Quorum systems ensure strong consistency across reads and writes to replicas by ensuring that read and write replica sets overlap. However, employing *partial* (or non-strict) quorums can lower latency by requiring fewer replicas to respond. With partial quorums, sets of replicas written to and read from need not overlap: given  $N$  replicas and read and write quorum sizes  $R$  and  $W$ , partial quorums imply  $R + W \leq N$ . For partial quorums, we derive closed-form solutions for PBS  $(K, p)$ -regular semantics and use Monte Carlo methods to explore the trade-off between latency and  $(\Delta, p)$ -regular semantics.

Finally, we use PBS to study the staleness observed in production deployments of Dynamo-style data stores under normal operation (i.e., failure-free scenarios). We show how high variance in write latency can lead to an increased window of inconsistency. For example, in one production environment, switching from spinning disks to solid-state drives dramatically improved expected consistency (e.g., 1.85 ms versus 45.5 ms wait time for a 99.9% probability of consistent reads) due to decreased write latency mean and variance. We also make quantitative observations of the latency-consistency trade-offs offered by partial quorums. For example, in another production environment, PBS calculations show an 81.1% combined read and write latency improvement at the 99.9th percentile (230 to 43.3 ms) for a 202-ms window of inconsistency (99.9% probability consistent reads). This analysis helps demonstrate the performance benefits that lead operators to choose eventual consistency and motivates additional end-user applications ranging from consistency monitoring to consistency-based service-level agreements (SLAs) and query planning.

## 2. PROBABILISTICALLY BOUNDED STALENESS

By popular definitions, eventually consistent data stores do not make any guarantees as to the recency of data that they return: “if no new updates are made to the object, eventually all accesses will return the last updated value.”<sup>26</sup> This is a useful *liveness* property, guaranteeing that something good eventually happens, but it provides no *safety* properties: the data store can return any data in the interim.<sup>4</sup> Many real-world eventually consistent stores do not make any guarantees beyond this definition, yet they are widely deployed and have seen increased popularity over the past decade (Section 3.2). As we will see, the protocols used by most eventually consistent stores indeed do not *enforce* additional guarantees, but they may *supply* them during operation.

To quantify semantics that are not guaranteed but are often provided, we develop a probabilistic framework for reasoning about consistency, called Probabilistically Bounded Staleness, or PBS. There are a wide range of possible consistency models that a data store can provide, from linearizability to causal consistency to eventual consistency; what is the likelihood that an end user will observe a given consistency model if it is not guaranteed? Here, we study variants of two classic kinds of (in)consistency in the form of staleness: versions and time. We develop variants of PBS metrics that provide quantitative expectations that a store will return a version that was written within the last  $K$  writes of the latest (where  $K = 1$  is latest) and that a store will return the latest version as of  $\Delta$  seconds ago. Ultimately, this does *not* provide a guarantee, but it is still useful for reasoning about and introspecting the behavior of a given system, similar to the usage of modern SLAs for performance (Section 6).

To begin, we first need to define a baseline for “strongly consistent” semantics. The Dynamo-style stores we study provide a choice between “strong” *regular* semantics and eventual consistency; we subsequently observe when the eventually consistent choices behave like their “strong” counterparts. According to the distributed systems literature:<sup>2</sup>

**DEFINITION 1.** *A read from a given data item obeys regular semantics if, in the case that the read does not overlap (in real time) with any writes to the same item, it returns the result the last completed write, or, if the read overlaps (in real time) with at least one write to the same data item, it returns either the result of the last completed write or the eventual result of one of the overlapping writes.*

Accordingly, regular semantics provide the illusion of a single copy of each replicated data item, except when there are concurrent reads and writes, during which writes’ effects may become visible and subsequently “disappear.” While this special, overlapping case is somewhat awkward to reason about (cf. definitions of linearizability<sup>13</sup>), this is a widely deployed data replication configuration.

Before we consider PBS applied to regular semantics, we first present a generalization of the semantics to account for multi-version staleness:<sup>2</sup>

**DEFINITION 2.** *A read from a given data item obeys  $K$ -regular semantics if, in the case that the read does not overlap (in real time) with a write to the same data item, it returns the result of one of the latest  $K$  completed writes, or, if the read overlaps (in real time) with a write to the same item, it returns either the result of one of the latest  $K$  completed writes or the eventual result of one of the overlapping writes.*

$K$ -regular semantics are useful for reasoning about *how stale* a given read can be and can also be used to enforce additional consistency properties such as monotonic reads, where reads do not appear to “go back in time.”<sup>5,25</sup>

We now present our first application of PBS principles. Given a system that does not guarantee  $K$ -regular

semantics, we can reason about its semantics by taking a probabilistic approach:

**DEFINITION 3.** A system provides  $(K, p)$ -regular semantics if each read provides  $K$ -regular semantics with probability  $p$ .

This modification is simple and straightforward: given an existing semantic guarantee, we can consider a probabilistic version of it, whereby reads may or may not obey the property. Of course, the above is simply a definition, and we must still determine how to actually provide PBS predictions, which will be the focus of the rest of the paper.

In addition to considering version-based staleness, we can also consider staleness with respect to real time. As we will see, message propagation and processing delays can influence consistency across time, so we extend regular semantics to consider time as well:

**DEFINITION 4.** A read obeys  $\Delta$ -regular semantics if it returns either the result of the latest write as of up to  $\Delta$  time units ago, the result of a write that was started but not completed as of up to  $\Delta$  time units ago.

We can similarly extend this definition to a probabilistic context:

**DEFINITION 5.** A system provides  $(\Delta, p)$ -regular semantics if each read obeys  $\Delta$ -regular semantics with probability  $p$ .

Although we do not consider them in this paper, it is possible to consider both time and version staleness together, which we term  $(K, \Delta, p)$ -semantics.<sup>5,7</sup>

### 3. QUORUM SYSTEM BACKGROUND

With PBS metrics in hand, we can proceed to apply them to real systems, where they are most useful. In this study, we will consider PBS metrics in the context of quorum-replicated data stores, which represent a large class of widely deployed real-world distributed data stores. However, with some work, we believe that our methodology is also applicable to other styles of replication. Here, we provide background on quorum systems, with a focus on current practice.

#### 3.1. Quorum foundations: Theory

Quorum systems have a long tradition as a replication strategy for distributed data.<sup>20</sup> Under quorum replication, a data store writes a data item by sending it to a set of servers responsible for the *replicas*, called a write quorum. To serve reads, the data store fetches the data from a possibly different set of replicas, called a read quorum. For reads, the data store compares the set of values returned by the replicas, and, given a total ordering of versions, can return the most recent value (or all values received, if desired). For each operation, the data store chooses (read or write) quorums from a set of sets of replicas, called a *quorum system*, with one system per data item. There are many kinds of quorum systems, but one simple configuration is to use read and write

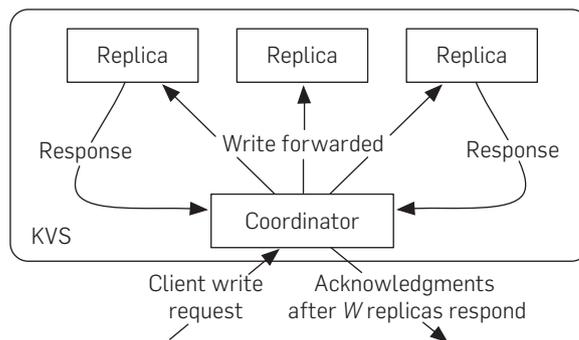
quorums of fixed sizes, which we will denote  $R$  and  $W$ , for a set of replicas of size  $N$ . A *strict quorum system* has the property that any two quorums in the quorum system overlap (have non-empty intersection), providing regular semantics. A simple example of a strict quorum system is the majority quorum system, in which each quorum is of size  $\lceil \frac{N+1}{2} \rceil$ . *Partial quorum systems*, which we will study, are a natural relaxation of strict quorum systems: at least two quorums in a partial quorum system do not overlap.<sup>19</sup>

#### 3.2. Quorum foundations: Practice

In practice, many distributed data management systems use quorums as a replication mechanism. Amazon's Dynamo<sup>10</sup> is the progenitor of a class of eventually consistent data stores that include Apache Cassandra,<sup>a</sup> Basho Riak,<sup>b</sup> and Project Voldemort.<sup>c</sup> All of these systems use the same variant of quorum-style replication and we are not aware of any widely adopted data store that uses a substantially different quorum-based replication protocol.

Dynamo-style quorum systems employ one quorum system per data item, typically maintaining the mapping of items to quorum systems using a consistent-hashing scheme or a centralized membership protocol. Each server in the system cluster stores multiple items. As shown in Figure 1, clients send read and write requests to a server in the system cluster, which subsequently forwards the request to *all* replicas for the operation's item. This coordinating server considers an operation complete when it has received responses from a predetermined number of replicas (typically configured per-operation). Accordingly, without message loss, all replicas eventually receive all writes. Dynamo denotes the replication factor of an item as  $N$ , the number of replica responses required for a successful read as  $R$ , and the number of replica acknowledgments required for a successful write as  $W$ . Like other strict quorum systems, Dynamo provides regular semantics when  $R + W > N$  during failure-free operation. However, unlike traditional

**Figure 1. Diagram of control flow for client write to Dynamo-style quorum ( $N = 3, W = 2$ ). A coordinator server handles the client write and sends it to all  $N$  replicas. The write call returns after the coordinator receives  $W$  acknowledgments.**



<sup>a</sup> <http://cassandra.apache.org/>

<sup>b</sup> <http://www.basho.com/riak/>

<sup>c</sup> <http://www.project-voldemort.com/>

quorum systems, Dynamo’s write quorum size increases even after the operation returns, growing via *anti-entropy*.<sup>4,10</sup> Coordinators send all requests to all replicas but consider only the first  $R$  ( $W$ ) responses. As a matter of nomenclature (and to disambiguate against “dynamic” quorum membership protocols), we will refer to these systems as *expanding partial quorum systems*.

As we discuss in extended versions of this paper,<sup>5,7</sup> system operators often report using partial quorum configurations in Dynamo-style stores, citing “maximum performance” in the “general case,” particularly for “low value” data or queries that need “very low latency and high availability.”

#### 4. PBS AND PARTIAL QUORUMS

Given our PBS metrics and an understanding of quorum behavior, we can develop models for the probability of consistency under partial quorums. Here, we briefly discuss version-based staleness for traditional probabilistic quorums and develop a more complex “white box” model of time-based staleness for Dynamo-style systems.

##### 4.1. PBS ( $K, p$ )-regular semantics

To understand static, non-expanding quorum behavior, we first revisit probabilistic quorum systems,<sup>19</sup> which provide probabilistic guarantees of quorum intersection in partial quorum systems. As an example, consider  $N$  replicas with read and write quorums of sizes  $R$  and  $W$  chosen uniformly at random. We can calculate the probability that the read quorum does not contain the last written version. This probability is the number of quorums of size  $R$  composed of replicas that were not written to in the write quorum divided by the number of possible read quorums:

$$p_s = \frac{\binom{N-W}{R}}{\binom{N}{R}} \quad (1)$$

The probability of inconsistency is high for small values of  $N$ . However, by scaling the number of replicas and quorum sizes, one can achieve an arbitrarily high probability of consistency.<sup>19</sup> For example, with  $N = 3, R = W = 1, p_s = 0.6$ , but with  $N = 100, R = W = 30, p_s = 1.88 \times 10^{-6}$ .<sup>2</sup> This is reminiscent of the Birthday Paradox: as the number of replicas increases, the probability of non-intersection between any two quorums decreases. Hence, the asymptotics of these systems are excellent—but only at asymptotic scales.

While probabilistic quorums allow us to determine the probability of returning the most recent value written to the database, they do not describe what happens when the most recent value is not returned. Here, we determine the probability of returning a value within a bounded number of versions ( $(K, p)$ -regular semantics). In the following formulation, we consider traditional, non-expanding write quorums (no anti-entropy).

Similar to the previous example, given independent, identically distributed (IID) choice of read and write quorums, returning one of the last  $k$  written versions is equivalent to

intersecting one of  $k$  independent write quorums. Given the probability of a single quorum non-intersection  $p$ , the probability of non-intersection with one of the last  $k$  independent quorums is  $p^k$ . Thus, the probability of non-intersection with uniform random choice in our example quorum system is Equation 1 exponentiated by  $k$ :

$$p = \left( \frac{\binom{N-W}{R}}{\binom{N}{R}} \right)^k \quad (2)$$

When  $N = 3, R = W = 1$ , this means that the probability of returning a version within 2 versions is  $0.5$ ; within 3 versions,  $0.703$ ; 5 versions,  $>0.868$ ; and 10 versions,  $>0.98$ . When  $N = 3, R = 1, W = 2$  (or, equivalently,  $R = 2, W = 1$ ), these probabilities increase:  $k = 1 \rightarrow 0.6, k = 2 \rightarrow 0.8, \text{ and } k = 5 \rightarrow >0.995$ .

This closed-form solution holds for quorums that do not change size over time. For expanding partial quorum systems, this solution is a lower bound on  $p$  for  $(K, p)$ -regular semantics. In the full version of this paper, we consider more advanced formulations of this analysis, including an analysis of monotonic reads consistency, quorum load, and closed-form solutions for mixed time and versions.<sup>5,7</sup>

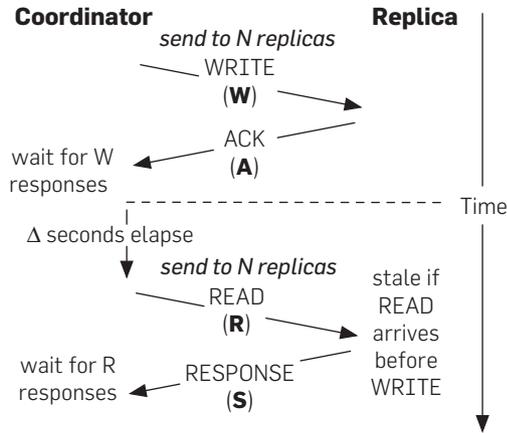
##### 4.2. Consistency in dynamo

We have a simple, closed-form model for  $(K, p)$ -regular semantics, but time-based ( $(\Delta, p)$ -regular) semantics are dependent on the quorum replication algorithm, workload, and any anti-entropy processes employed by a given system. In this section, we develop techniques for analyzing PBS  $(\Delta, p)$ -regular semantics in the context of Dynamo-style data stores.

Dynamo-style quorum systems are inconsistent as a result of read and write message reordering, in turn a product of message delays. In this work, we take a white box approach to inconsistency and directly examine the protocols behind consistency. Accordingly, we develop a model of message latency in Dynamo operation that captures the effects of message delays for write requests (W), write acknowledgments (A), read requests (R), and read responses (S), and which, for convenience, we call *WARS*. In Figure 2, we illustrate *WARS* using a space-time diagram for messages between a coordinator and a single replica for a write followed by a read  $\Delta$  seconds after the write completes. Accordingly,  $\Delta$  here corresponds to the  $\Delta$  in PBS  $(\Delta, p)$ -regular semantics. In brief, reads are stale when all of the first  $R$  responses to the read request arrived at their replicas before the last (completed) write request arrived.

For a write, the coordinator sends  $N$  messages, one to each replica. The message from the coordinator to replicas containing the write is delayed by a value drawn from distribution  $W$ . The coordinator waits for  $W$  responses from the replicas before it can consider the write completed. Each response acknowledging the write is delayed by a value drawn from the distribution  $A$ .

**Figure 2. The WARS model describes staleness in Dynamo by modeling message latencies between a coordinator and replicas for a write operation followed by a read operation  $t$  seconds later. In an  $N$  replica system, the depicted messages are exchanged with  $N$  replicas.**



For a read, the coordinator (possibly different than the write's coordinator, and possibly representing a different client than the client that issued the write) sends  $N$  messages, one to each replica. The message from coordinator to replica containing the read request is delayed by a value drawn from distribution  $R$ . The coordinator waits for  $R$  responses from the replicas before returning the most recent value it receives. The read response from each replica is delayed by a value drawn from the distribution  $S$ .

The read coordinator will return stale data if the first  $R$  responses received reached their replicas before the replicas received the latest version (delayed by  $w$ ). When  $R + W > N$ , this is impossible. However, under partial quorums, the frequency of this occurrence depends on the latency distributions. If we denote the write completion time (when the coordinator has received  $W$  acknowledgments) as  $w_t$ , a single replica's response is stale if  $r' + w_t + \Delta < w'$  for  $r'$  drawn from  $R$  and  $w'$  drawn from  $W$ . Writes have time to propagate to additional replicas both while the coordinator waits for all required acknowledgments (A) and as replicas wait for read requests (R). Read responses are further delayed in transit (S) back to the read coordinator, inducing further possibility of reordering. Qualitatively, long-tailed write distributions ( $W$ ) and relatively faster reads ( $R, S$ ) increase the chance of staleness due to reordering.

WARS considers the effect of message sending, delays, and reception, but this represents a difficult analytical formulation with several non-independent order statistics. As we discuss in Section 5.1, we instead explore WARS using Monte Carlo methods, which are straightforward to understand and implement. We have found that the WARS distributions are easy to parameterize given traces of real-world system behavior (Section 5.3).

## 5. PBS IN ACTION

Given our PBS models for Dynamo-style stores, we now apply them to real-world systems. As discussed in Section 4.2, PBS  $(\Delta, p)$ -regular behavior in a given system depends on the propagation of reads and writes across

replicas. We introduced the WARS model as a means of reasoning about inconsistency in Dynamo-style quorum systems, but quantitative metrics such as staleness observed in practice depend on each of WARS's latency distributions. In this section, we perform an analysis of Dynamo-style  $(\Delta, p)$ -regular semantics for both synthetic and real-world distributions to better understand how frequently "eventually consistent" means "consistent" and, more importantly, why Dynamo-style stores are indeed frequently consistent.

PBS  $(K, p)$ -regular analysis for partial quorums is easily captured in closed form (Section 4.1). It does not depend on write latency or any environmental variables. Indeed, in practice, without expanding quorums or anti-entropy, we observe that our derived equations hold true experimentally.

In contrast,  $(\Delta, p)$ -regular semantics depends on anti-entropy, which is more complicated. In this section, we focus on deriving experimental expectations for PBS  $(\Delta, p)$ -regular semantics. We first validate our Monte Carlo analysis based on the WARS model and using message-level traces gathered from Cassandra clusters in Berkeley. We next explore synthetic latency distributions and, for the remainder of our analysis, explore distributions from two Internet companies: LinkedIn and Yammer.

### 5.1. Monte Carlo simulation

We implemented WARS analysis in a Monte Carlo-based simulation. Calculating  $(\Delta, p)$ -regular semantics for a given value of  $\Delta$  is straightforward (see pseudocode in Bailis et al.<sup>7</sup>). To simulate the message delays between coordinator and each of the  $N$  replicas, denote the  $i$ th sample drawn from distribution  $D$  as  $D[i]$  and draw  $N$  samples from  $W, A, R,$  and  $S$ . Compute the time that the write request completes ( $w_t$ , or the time that the coordinator gets its  $W$ th acknowledgment; the  $W$ th smallest value of  $\{W[i] + A[i], i \in [0, N]\}$ ). Next, determine whether any of the first  $R$  replicas contained an up-to-date response: check whether any the first  $R$  samples of  $R$ , ordered by  $R[i] + S[i]$  obey  $w_t + R[i] + \Delta \leq W[i]$ . Repeating this process multiple times provides an approximation of the behavior specified by the trace. Extending this formulation to analyze  $(K, \Delta, p)$ -regular semantics given a distribution of write arrival times will require accounting for multiple writes across time. As described in extended versions of this paper,<sup>5,7</sup> we validated this analysis on traces that we collected from a real-world Cassandra cluster. We observed an average RMSE of 0.28% for  $(\Delta, p)$ -regular semantics prediction and an average N-RMSE of 0.48% for latency predictions.

### 5.2. Write latency distribution effects

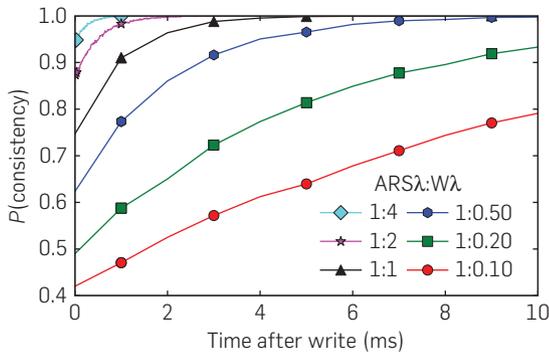
The WARS model of Dynamo-style systems dictates that high variance in latency increases staleness. Before studying real-world workloads (Section 5.3), we quantified this behavior in isolation via synthetic distributions: we swept a range of exponentially distributed write distributions (changing parameter  $\lambda$ , which dictates the mean and tail of the distribution) while fixing  $A = R = S$ .

Our results, shown in Figure 3, demonstrate this relationship. When the variance and mean of  $W$  are 0.0625 ms

and 0.25 ms ( $\lambda = 4$ , one-fourth the mean of  $A = R = S = 1$  ms), we observe a 94% chance of consistency immediately after the write and 99.9% chance after 1 ms. However, when the variance and mean of  $W$  are 100 ms and 10 ms ( $\lambda = 0.1$ , ten times the mean of  $A = R = S = 1$  ms), we observe a 41% chance of consistency immediately after write and a 99.9% chance of consistency only after 65 ms. As the variance and mean increase, so does the probability of inconsistency. Under distributions with fixed means and variable variances (uniform, normal), we observe that the mean of  $W$  is less important than its variance if  $W$  is strictly greater than  $A = R = S$ .

Decreasing the mean and variance of  $W$  improves the probability of consistent reads. This means that, as we will see, techniques that lower write latency variance result in more consistent reads. Instead of increasing read and write quorum sizes, operators could chose to lower (relative)  $W$  latencies through hardware configuration or by delaying reads, although this latter option is detrimental to performance for read-dominated workloads and may introduce undesirable queuing effects. Nonetheless, this PBS analysis illustrates the fact that stale reads can be avoided in a variety of ways beyond simple adjustment of quorum sizes.

**Figure 3.**  $(\Delta, p)$ -regular semantics with exponential latency distributions for  $W$  and  $A = R = S$ . Mean latency is  $1/\lambda$ .  $N = 3, R = W = 1$ .



### 5.3. Production latency distributions

To study real-world behavior, we obtained production latency statistics from two Internet-scale companies. While message-level  $WARS$  timing traces would deliver more accurate predictions, we opted for a pragmatic compromise: as described in extended versions of this work, we fit  $WARS$  distributions to each of the provided statistics (Figure 4).<sup>5,7</sup>

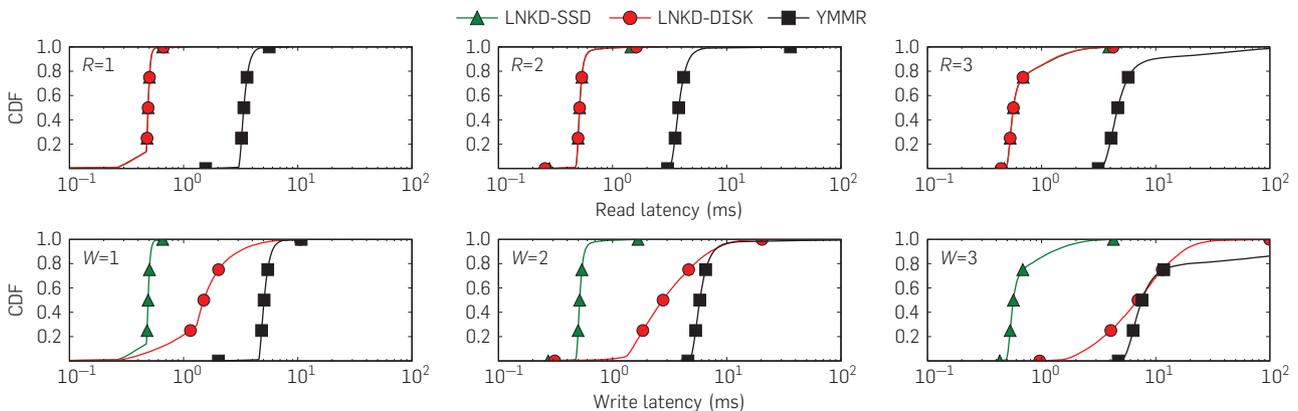
LinkedIn<sup>d</sup> is an online professional social network with over 225 million members as of July 2013. To provide highly available, low latency data storage, engineers at LinkedIn built Voldemort. Alex Feinberg, a lead engineer on Voldemort, graciously provided us with latency distributions for a single server under peak traffic for a user-facing service at LinkedIn, representing 60% read and 40% read-modify-write traffic.<sup>5,7</sup> Feinberg reports that, using spinning disks, Voldemort is “largely IO bound and latency is largely determined by the kind of disks we’re using, [the] data to memory ratio and request distribution.” With solid-state drives (SSDs), Voldemort is often “CPU and/or network bound” and “maximum latency is generally determined by [garbage collection] activity (rare, but happens occasionally) and is within hundreds of milliseconds.” We denote the LinkedIn spinning disk distribution as LNKD-DISK and SSD trace as LNKD-SSD.

Yammer<sup>e</sup> provides private social networking to over 200,000 companies as of July 2013 and uses Basho’s Riak for some client data. Coda Hale, an infrastructure architect, provided performance statistic for their production Riak deployment.<sup>5,7</sup> Hale mentioned that “reads and writes have radically different expected latencies, especially for Riak.” Riak delays writes “until the fsync returns, so while reads are often <1 ms, writes rarely are.” Also, although we do not model this explicitly, Hale also noted that the size of values is important, claiming “a big performance improvement by adding LZFS compression to values.” We denote the Yammer latency distribution as YMMR.

<sup>d</sup> <http://www.linkedin.com/>

<sup>e</sup> <http://www.yammer.com/>

**Figure 4.** Read and write operation latency for production fits for  $N = 3$  and varying  $R$  and  $W$ . For reads, LNKD-SSD is equivalent to LNKD-DISK. Depicted points mark significant percentiles for comparison. Higher values of  $R$  and  $W$  result in increased latency.



### 5.4. Staleness in production

The production latency distributions confirm that staleness is frequently limited in eventually consistent stores. We measured the  $(\Delta, p)$ -regular semantics for each distribution (Figure 5). LNKD-SSD and LNKD-DISK demonstrate the importance of write latency in practice. Immediately after write completion, LNKD-SSD had a 97.4% probability of consistent reads, reaching over a 99.999% probability of consistent reads after 5 ms. LNKD-SSD's reads briefly raced with its writes immediately after write completion. However, within a few milliseconds after the write, the chance of a read arriving before the last write was nearly eliminated. The distribution's read and write operation latencies were small (median 0.489 ms), and writes completed quickly across all replicas due to the distribution's short tail (99.9th percentile 0.657 ms). In contrast, under LNKD-DISK, writes take much longer (median 1.50 ms) and have a longer tail (99.9th percentile 10.47 ms). LNKD-DISK's  $(\Delta, p)$ -regular semantics reflects this difference: immediately after write completion, LNKD-DISK had only a 43.9% probability of consistent reads and, 10 ms later, only a 92.5% probability. This suggests that SSDs may greatly improve consistency due to reduced write variance. Similarly, one should expect that consistency would be improved by using explicit memory management rather than unscheduled garbage collection.

We experienced similar behavior with the other distributions. Immediately after write completion ( $\Delta = 0$ ), YMMR had a  $p = 89.3\%$  probability of consistency. However, the YMMR distribution only reached a  $p = 99.9\%$  probability of consistency at  $\Delta = 1364$  ms due to high variance and long-tail behavior in its write distribution. This hints that, given multiple replicas for a data item, the durability benefits of synchronously flushing writes to disk may have adverse effects on consistency. An alternative approach that could improve consistency and avoid this high variance would rely on multi-replica (in-memory or buffer cache) replication for durability and only flush writes asynchronously.

### 5.5. Quorum sizing

In addition to  $N = 3$ —the most common quorum size we encountered in practice—we consider how varying the number of replicas ( $N$ ) affects  $(\Delta, p)$ -regular semantics while maintaining  $R = W = 1$ . The results, depicted in Figure 6, show that the probability of consistency immediately after write completion decreases as  $N$  increases. With two replicas, LNKD-DISK has a 57.5% probability of consistent reads immediately after write completion but only a 21.1% probability with 10 replicas. However, at high probabilities ( $p$ ), the window of inconsistency ( $\Delta$ ) for increased replica sizes is close. For LNKD-DISK,  $\Delta$  at  $p = 99.9\%$  ranges from 45.3 ms for 2 replicas to 53.7 ms for 10 replicas.

Figure 5.  $(\Delta, p)$ -regular semantics for production operation latencies.

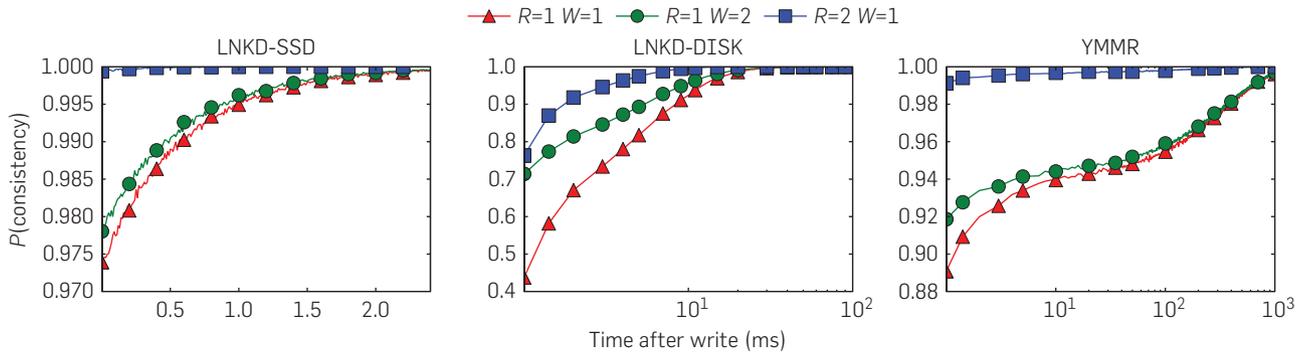
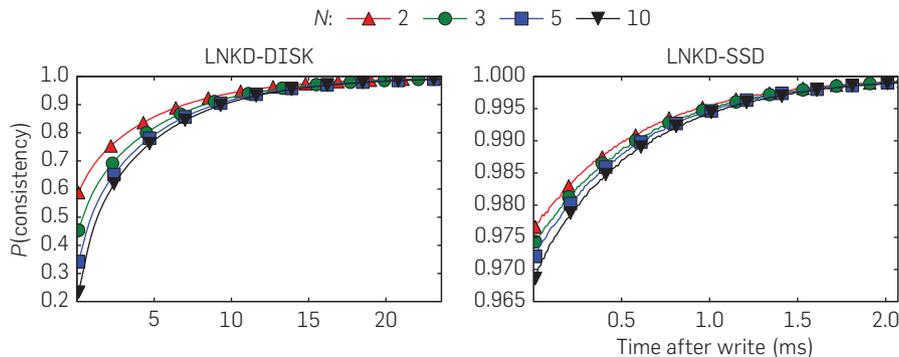


Figure 6.  $(\Delta, p)$ -regular semantics for production operating latencies varying the number of replicas  $N$  when  $R = W = 1$ . Inconsistencies  $p$  are likely to resolve quickly (low  $\Delta$ ) even for large replication factors.



These results imply that maintaining a large number of replicas for availability or better performance results in a potentially large impact on consistency immediately after writing. However, the  $(\Delta, p)$ -regular semantics probability ( $p$ ) will still rise quickly (small  $\Delta$ ).

### 5.6. Latency vs. staleness

Choosing values for  $R$  and  $W$  is a trade-off between operation latency and consistency. To measure this trade-off, we compared 99.9th percentile operation latencies with the corresponding  $\Delta$  at  $p = 99.9\%$  for quorum configurations where  $N = 3$ , typical of deployments in the field.

Partial quorums often exhibit favorable latency-consistency trade-offs (Table 1). For YMMR,  $R = W = 1$  results in low latency reads and writes (16.4ms) but high  $\Delta$  (1364ms). However, setting  $R = 2$  and  $W = 1$  reduces  $\Delta$  to 202ms and the combined read and write latencies are 81.1% (186.7ms) lower than the fastest strict quorum ( $W = 1, R = 3$ ). Allowing  $p = 99.9\%$ ,  $\Delta = 13.6$ ms reduces LNKD-DISK read and write latencies by 16.5% (2.48ms). For LNKD-SSD, across 10  $M$  writes (“seven nines”), we did not observe staleness with  $R = 2, W = 1$ .  $R = W = 1$  reduced latency by 59.5% (1.94ms) with a corresponding  $\Delta = 1.85$ ms. In summary, lowering values of  $R$  and  $W$  can greatly improve operation latency but, even in the tail, the duration of inconsistency ( $\Delta$ ) is relatively small.

We omit full results here, but we have also experimented with heterogeneous replica behavior and with multi-item guarantees such as causal consistency and transactional atomicity.<sup>7</sup>

## 6. DISCUSSION AND FUTURE WORK

In this section, we discuss PBS design decisions, describe our experiences integrating PBS with real-world stores and end-user applications, and suggest areas for future work.

### 6.1. Prediction and verification

In this work, we have developed techniques for consistency prediction, which provide an expectation of system behavior given a set of input data about the system and the current operating environment. Given a trace, one can predict staleness after an arbitrary amount of time or number of versions without having to actually run any additional queries.

Additionally, predictions allow users to easily perform “what-if” analysis across arbitrary replication configurations, request distributions ( $\Delta$  and  $K$ ), and hardware configurations (e.g., switching from SSDs to disks). We have found that prediction is computationally inexpensive and can be performed on a decentralized, per-replica basis. While prediction is flexible, it is only as good as the input traces. With representative input data, predictions will be accurate. However, with unrepresentative data (or bad models), prediction accuracy will suffer.

In contrast, verification<sup>21</sup> informs users how their data stores are performing with guaranteed certainty. If a user makes a change to their replication settings using a predictor, she may want to ensure that the change behaves as expected. While this verification is not well-suited to “what-if” analysis, it is an important complement to prediction. Verification effectively provides a metric that results from integrating the  $(K, \Delta, p)$ -regular semantics density function weighted by the given read request rate (measured with respect to time since the last write). Additionally, verification is algorithmically complex,<sup>11</sup> but in our experience is not terribly difficult to implement.

We believe that both techniques will be increasingly useful as systems begin to treat consistency as a continuous, quantitative metric. Taken together, consistency prediction and verification techniques form a powerful toolkit.

### 6.2. White and black box approaches

In this work, we use a white box approach to consistency and exploit expert knowledge of replication protocols to provide quantitative insight. This requires translating from user-centric, declarative specifications of consistency anomalies into back-end protocol events (e.g., in the WARS model, the reordering between read and write responses). We could have alternatively attempted to reverse engineer system internals or provide an implementation-independent predictor, but this black box analysis would be substantially more complex. We believe that verification techniques are more amenable to black box techniques and that the portability benefits of black box techniques must be weighed against their potential inaccuracy. Given our experiences integrating prediction into existing data

**Table 1.**  $\Delta$  for  $(\Delta, p = 99.9\%)$ -regular semantics and 99.9th percentile read ( $L_r$ ) and write latencies ( $L_w$ ), varying  $R$  and  $W$  with  $N = 3$ .

	LNKD-SSD			LNKD-DISK			YMMR		
	$L_r$	$L_w$	$t$	$L_r$	$L_w$	$t$	$L_r$	$L_w$	$t$
$R = 1, W = 1$	<b>0.66</b>	<b>0.66</b>	<b>1.85</b>	0.66	10.99	45.5	5.58	10.83	1364.0
$R = 1, W = 2$	0.66	1.63	1.79	0.65	20.97	43.3	5.61	427.12	1352.0
$R = 2, W = 1$	<b>1.63</b>	<b>0.65</b>	<b>0</b>	<b>1.63</b>	<b>10.9</b>	<b>13.6</b>	<b>32.6</b>	<b>10.73</b>	<b>202.0</b>
$R = 2, W = 2$	<b>1.62</b>	<b>1.64</b>	<b>0</b>	1.64	20.96	0	33.18	428.11	0
$R = 3, W = 1$	4.14	0.65	0	<b>4.12</b>	<b>10.89</b>	<b>0</b>	<b>219.27</b>	<b>10.79</b>	<b>0</b>
$R = 1, W = 3$	0.65	4.09	0	0.65	112.65	0	5.63	1870.86	0

Significant latency-staleness trade-offs are in bold.

stores and the large-scale adoption of open-source data stores, we believe that white box techniques are feasible, even if they require modifications to existing stores.

### 6.3. Real-world store integration

With the help of several open-source developers, we have developed patches for PBS functionality within two NoSQL stores: Cassandra and Voldemort. For Cassandra, we have taken two approaches: an invasive but more accurate implementation and an external but less accurate prediction module. For the former approach, we modified the Cassandra messaging layer to add a message creation timestamp in order to measure each of the  $W$ ,  $A$ ,  $R$ ,  $S$  distributions. When tracing is enabled on a given server, the messaging layer logs per-operation timestamps in a separate PBS prediction module. The timestamps are stored in an in-memory circular buffer for each of the required message latencies. Subsequently, users can call the PBS predictor module via an externally accessible interface, which they can use to provide advanced functionality like dynamic replication configuration and monitoring (see below). This provides relatively accurate predictions at the expense of having to instrument the messaging layer. However, as data stores like Cassandra have expanded their user-accessible monitoring data (e.g., per-query latency tracing), we have more recently been exploring predictions outside of the database—the latter approach—which we have implemented for Voldemort. We have open-sourced implementations of both approaches.

### 6.4. PBS applications

PBS enables functionality not possible without quantitative consistency metrics. With PBS, we can automatically configure replication parameters by optimizing operation latency given constraints on staleness and minimum durability. Data store operators can subsequently provide service level agreements to applications and quantitatively describe latency-staleness trade-offs to users. Operators can dynamically configure replication using online latency measurements. This optimization also allows disentanglement of replication for durability from replication for reasons of low latency and higher capacity. For example, operators can specify a minimum replication factor for durability and availability but can also automatically increase  $N$ , decreasing tail latency for fixed  $R$  and  $W$ . We expanded upon these possibilities in a *SIGMOD 2013* demo that featured real-time predictions for a live Cassandra cluster and mock web service.<sup>6</sup>

### 6.5. WARS limitations and extensions

There are several limitations and potential extensions of the *WARS* model, which we sketch here and discuss in greater detail in extended versions of this work.<sup>5,7</sup> *WARS* only models a single write and read and is therefore a conservative estimate for multi-write scenarios. Moreover, in our current treatment, *WARS* treats each distribution as IID. This is not fundamental to the model but is a limitation of our latency traces from industry. *WARS* also does

not capture effects of additional, commonly employed anti-entropy processes (e.g., read-repair, Merkle-tree exchange) and may be a conservative estimate of staleness. It does not address system behavior under failures, which varies from store to store, and it assumes that clients contact a coordinator server instead of issuing requests themselves (e.g., Voldemort). We believe that these limitations are not fundamental and can be accounted for in a white box model but nonetheless remain future work. Finally, clients requiring staleness detection may do so asynchronously, enabling speculative reads and compensatory actions.<sup>5,7</sup>

## 7. RELATED WORK

This research builds upon several related areas: quorum replication, consistency models, and approaches to quantifying consistency.

Managing replicated data is a long-studied problem in distributed systems and concurrent programming. There is a plethora of consistency models offering different trade-offs between semantics, performance, and availability. Traditional models such as serializability and linearizability as well as more recently proposed models such as timeline consistency<sup>8</sup> and parallel snapshot isolation<sup>23</sup> all provide “strong” semantics at the cost of high availability, or the ability to provide “always-on” response behavior at all replicas. In contrast, faced with a requirement for high availability and low latency, many production data stores have turned to weaker semantics to provide availability in the face of network partitions.<sup>9,26</sup>

Our focus in this paper is on the semantics provided by existing, widely deployed systems. Due to the prevalence of “strong” consistency and eventual consistency models in practice (and the explicit choice between these two models in Dynamo-style systems), we largely focus on this dichotomy. However, there are a range of alternative but still “weak” models. As an example, the Bayou system provided a range of “session guarantees,” including read-your-writes and monotonic reads consistency.<sup>25</sup> Similarly, a recent Technical Report from UT Austin claims that a variant of causal consistency is the strongest consistency model achievable in an available, one-way convergent (eventually consistent) system,<sup>18</sup> a model that has recently attracted systems implementations.<sup>17</sup> As we have hinted, probabilistic approaches are applicable to the consistency models beyond those we have considered here. Specific to staleness, prior research such as TACT<sup>27</sup> has examined how to provide deterministic staleness bounds. These deterministically bounded staleness systems represent the deterministic dual of PBS.

Our techniques for analyzing modern partial quorum systems draw on existing, largely theoretical literature. We briefly surveyed quorum replication<sup>20</sup> in Section 3. In this work, we specifically draw inspiration from probabilistic quorums<sup>19</sup> in analyzing expanding quorum systems and their consistency. We believe that revisiting probabilistic quorum systems—including non-majority quorum systems such as tree quorums—in the context of write propagation, anti-entropy, and Dynamo is a promising area for

theoretical work. While we study probabilistic guarantees for staleness, prior work on  $k$ -quorums<sup>2, 3</sup> have looked at *deterministic* guarantees that a partial quorum system will return values that are within  $k$  versions of the most recent write.<sup>2</sup>

Finally, recent research has focused on measuring and verifying the consistency of eventually consistent systems both theoretically and experimentally (Rahman et al. provide a brief survey<sup>21</sup>). This is useful for validating consistency predictions and understanding staleness violations.

## 8. CONCLUSION

In this paper, we introduced PBS, which models the expected staleness of data returned by eventually consistent data stores. PBS offers an alternative to the all-or-nothing consistency guarantees of many of today's systems by offering SLA-style consistency predictions. By extending prior theory on probabilistic quorum systems, we derived an analytical solution for the  $(K, p)$ -regular semantics of a partial quorum system, representing the expected staleness of a read operation in terms of versions. We also analyzed  $(\Delta, p)$ -regular semantics, or expected staleness of a read in terms of real time, under Dynamo-style quorum replication. To do so, we developed the *WARS* latency model to explain how message reordering leads to staleness under Dynamo. To examine the effect of latency on  $(\Delta, p)$ -regular semantics in practice, we used real-world traces from Internet companies to drive a Monte Carlo analysis. We find that eventually consistent Dynamo-style quorum configurations are often consistent after tens of milliseconds due in large part to their resilience to per-server latency variance. We conclude that eventually consistent partial quorum replication schemes frequently deliver consistent data during failure-free operation while offering significant latency benefits. We believe that continued study and deployment of quantitative consistency metrics will both enable useful end-user functionality and shed light on previously opaque and frequently controversial replication configurations.

## Interactive demonstration

An interactive demonstration of Dynamo-style PBS is available at <http://pbs.cs.berkeley.edu/#demo>.

## Acknowledgments

This work was greatly improved by feedback from many previously noted individuals.<sup>5, 7</sup> It was supported by gifts from Google, SAP, Amazon Web Services, Blue Goji, Cloudera, Ericsson, General Electric, Hewlett Packard, Huawei, IBM, Intel, MarkLogic, Microsoft, NEC Labs, NetApp, NTT Multimedia Communications Laboratories, Oracle, Quanta, Splunk, and VMware. This material is based upon work supported by the National Science Foundation Graduate Research Fellowship under Grant DGE 1106400, National Science Foundation Grants IIS-0713661, CNS-0722077, and IIS-0803690, NSF CISE Expeditions award CCF-1139158, the Air Force Office of Scientific Research Grant FA95500810352, and by DARPA contract FA865011C7136. 

## References

- Abadi, D.J. Consistency tradeoffs in modern distributed database system design: CAP is only part of the story. *IEEE Comput.* 45, 2 (2012), 37–42.
- Aiyer, A., Alvisi, L., Bazzi, R.A. On the availability of non-strict quorum systems. In *DISC 2005*.
- Aiyer, A.S., Alvisi, L., Bazzi, R.A. Byzantine and multi-writer  $k$ -quorums. In *DISC* (2006), 443–458.
- Bailis, P., Ghodsi, A. Eventual consistency today: Limitations, extensions, and beyond. *ACM Queue* 11, 3 (Mar. 2013), 20:20–20:32.
- Bailis, P., Venkataraman, S., Franklin, M.J., Hellerstein, J.M., Stoica, I. Probabilistically bounded staleness for practical partial quorums. *PVLDB* 5, 8 (2012), 776–787.
- Bailis, P., Venkataraman, S., Franklin, M.J., Hellerstein, J.M., Stoica, I. PBS at work: Advancing data management with consistency metrics. In *SIGMOD 2013 Demo*.
- Bailis, P., Venkataraman, S., Franklin, M.J., Hellerstein, J.M., Stoica, I. Quantifying eventual consistency with PBS. *VLDB J.* (2014). (see <http://link.springer.com/article/10.1007/s00778-013-0330-1>).
- Cooper, B.F., Ramakrishnan, R., Srivastava, U., Silberstein, A., Bohannon, P., Jacobsen, H.A., Puz, N., Weaver, D., Yerneni, R. Phnuts: Yahoo!'s hosted data serving platform. In *VLDB 2008*.
- Davidson, S., Garcia-Molina, H., Skeen, D. Consistency in partitioned networks. *ACM Comput. Surv.* 17, 3 (1985), 314–370.
- DeCandia, G., Hastorun, D., Jampani, M., Kakulapati, G., Lakshman, A., Pilchin, A., Sivasubramanian, S., Vosshall, P., Vogels, W. Dynamo: Amazon's highly available key-value store. In *SOSP 2007*, 205–220.
- Golab, W., Li, X., Shah, M.A. Analyzing consistency properties for fun and profit. In *PODC* (2011), 197–206.
- Hamilton, J. Perspectives: I love eventual consistency but... <http://perspectives.mdirona.com/2010/02/24/IloveEventualConsistencyBut.aspx> (24 Feb. 2010).
- Herlihy, M., Wing, J.M. Linearizability: A correctness condition for concurrent objects. *ACM Trans. Program. Lang. Syst.* 12, 3 (1990), 463–492.
- Kirkell, J. Consistency or bust: Breaking a Riak cluster. <http://www.oscon.com/oscon2011/public/schedule/detail/19762>. Talk at O'Reilly OSCON 2011 (27 Jul. 2011).
- Linden, G. Make data useful. <https://sites.google.com/site/glinden/Home/StanfordDataMining.2006-11-29.ppt> (29 Nov. 2006).
- Linden, G. Marissa Mayer at Web 2.0. <http://glinden.blogspot.com/2006/11/marissa-mayer-at-web-20.html> (9 Nov. 2006).
- Lloyd, W., Freedman, M.J., Kaminsky, M., Andersen, D.G. Stronger semantics for low-latency geo-replicated storage. In *NSDI 2013*.
- Mahajan, P., Alvisi, L., Dahlin, M. *Consistency, Availability, Convergence*. Technical Report TR-11-22, Computer Science Department, University of Texas at Austin, 2011.
- Malkhi, D., Reiter, M., Wool, A., Wright, R. Probabilistic quorum systems. *Inform. Commun.* 170 (2001), 184–206.
- Merideth, M., Reiter, M. Selected results from the latest decade of quorum systems research. In *Replication*, B. Charron-Bost, F. Pedone, and A. Schiper, eds. Volume 5959 of *LNCS* (2010). Springer, 185–206.
- Rahman, M., Golab, W., AuYoung, A., Keeton, K., Wylie, J. Toward a principled framework for benchmarking consistency. In *Proceedings of the 8th Workshop on Hot Topics in System Dependability* (Hollywood, CA, 2012), USENIX.
- Schurman, E., Brutlag, J. Performance related changes and their user impact. Presented at *Velocity Web Performance and Operations Conference* (San Jose, CA, Jun. 2009).
- Sovran, Y., Power, R., Aguilera, M.K., Li, J. Transactional storage for geo-replicated systems. In *SOSP 2011*.
- Stonebraker, M. Urban myths about SQL. [http://voltdb.com/\\_pdf/VoltDB-MikeStonebraker-SQLMythsWebinar-060310.pdf](http://voltdb.com/_pdf/VoltDB-MikeStonebraker-SQLMythsWebinar-060310.pdf). VoltDB Webinar (Jun. 2010).
- Terry, D.B., Demers, A.J., Petersen, K., Spreitzer, M.J., Theimer, M.M., Welch, B.B. Session guarantees for weakly consistent replicated data. In *PDIS 1994*.
- Vogels, W. Eventually consistent. *CACM* 52 (2009), 40–44.
- Yu, H., Vahdat, A. Design and evaluation of a conit-based continuous consistency model for replicated services. *ACM Trans. Comput. Syst.* 20, 3 (2002), 239–282.

Peter Bailis, Shivaram Venkataraman, Michael J. Franklin, Joseph M. Hellerstein, and Ion Stoica ([pbailis](mailto:pbailis),

[shivaram](mailto:shivaram), [franklin](mailto:franklin), [hellerstein](mailto:hellerstein), [istoica](mailto:istoica))@cs.berkeley.edu), University of California, Berkeley.