# PBS at Work: Advancing Data Management with Consistency Metrics

Peter Bailis, Shivaram Venkataraman, Michael J. Franklin, Joseph M. Hellerstein, Ion Stoica
*UC Berkeley*

## ABSTRACT

A large body of recent work has proposed analytical and empirical techniques for quantifying the data consistency properties of distributed data stores. In this demonstration, we begin to explore the wide range of new database functionality they enable, including dynamic query tuning, consistency SLAs, monitoring, and administration. Our demonstration will exhibit how both application programmers and database administrators can leverage these features. We describe three major application scenarios and present a system architecture for supporting them. We also describe our experience in integrating Probabilistically Bounded Staleness (PBS) predictions into Cassandra, a popular NoSQL store and sketch a demo platform that will allow SIGMOD attendees to experience the importance and applicability of real-time consistency metrics.

## Categories and Subject Descriptors

H.2.4 [**Database Management**]: Systems—*Distributed databases*

## Keywords

eventual consistency, monitoring, prediction; auto-tuning

## 1. INTRODUCTION

Modern distributed data stores offer a choice of consistency models. Weak consistency models are fast and guarantee "always-on" behavior but provide limited guarantees. Stronger consistency models are easier to reason about but are slower and potentially unavailable [4]. The choice of a consistency model has wide-ranging implications for application writers, operations management, and end-users. Yet, in light of its performance benefits [2], weak consistency is often considered acceptable.

Eventual consistency—perhaps the most commonly deployed weak consistency model—is particularly weak: in the absence of new writes to a data item, reads will eventually return the same value [8]. This eventual consistency provides no guarantees as to when new writes will become visible to readers and what versions of data items will be presented in the interim. For example, reading all `null` values from a database satisfies eventual consistency. Similarly, the data store can delay writes for weeks and not violate eventual consistency guarantees. Yet, despite these weak semantics, there is common sentiment among practitioners that eventual consistency is often "good enough" and "worthwhile" for many Internet applications.

In recent work, we provided quantitative metrics and accompanying techniques for analyzing the consistency provided by eventually consistent stores, called Probabilistically Bounded Staleness, or PBS [3]. Eventually consistent stores do not make promises about the length of time required to observe an update or the staleness of values, but this does not preclude us from making informed statements about a store's expected behavior. By using expert knowledge of the underlying data store and its replication protocols and performing lightweight in-situ profiling, we can inform data store users about what consistency they are likely to observe in the future. PBS provides a probabilistic answer to the question of "how eventual and how consistent is eventual consistency?"

PBS predictions are part of a larger trend towards providing quantitative measurements and analysis of weakly consistent stores. Recent work ranges from theoretical [5] to experimental studies [6] of properties such as linearizability, serializability, and regular register semantics. PBS in particular has experienced relative popularity in the NoSQL community and our implementation of PBS for quorum-based systems was integrated with Cassandra's mainline source and released as a feature of the 1.2.0 release in January 2013 [1]. Our discussions with other researchers indicate the possibility of additional technology transfer of monitoring and verification techniques in the future.

While monitoring and prediction are useful in themselves, perhaps more importantly, they enable a rich space of applications that were previously infeasible or impossible. Among these applications, we consider three to be the most promising: *i.)* dynamic request-based consistency parameters, or auto-tuning request routing based on latency- and consistency-based service level agreements (SLAs), *ii.)* database administration tasks with respect to the impact of slow nodes and networks, replication factors, and data store parameters like anti-entropy rates, and *iii.)* enhancement of traditional alerting and monitoring frameworks. We elaborate further in Section 2, but, in short, these quantitative metrics allow new ways to improve the performance, semantics, and maintenance of eventually consistent stores.

In this demo paper, we outline these advanced use cases in detail (Section 2), describe how quantitative metrics can be integrated into existing data stores based on our experiences with the Cassandra community (Section 3), and sketch how we plan to demonstrate this newly enabled functionality, allowing SIGMOD attendees to act as end-users, operations managers, and adversaries of an eventually consistent internet service (Section 4).


*SIGMOD'13,* June 22–27, 2013, New York, New York, USA.

## 1.1 PBS Metrics

To provide background on consistency metrics, we briefly summarize the metrics proposed in PBS [3]. Data staleness is expressed in terms of two metrics: wall-clock time and versions.

**t-visibility:** $t$-visibility captures the "window of inconsistency" in terms of wall clock time after a write happens. An eventually consistent system will eventually respond to all read requests with the last written value, and $t$-visibility provides an expected value for this "eventuality." More formally, $t$-visibility is the probability that a read operation starting $t$ seconds after the previous write completed will see the most recent value from the data store. For example, if a data store configuration has $t$-visibility of 95% at 100ms, it means that 100ms after the last write completes, 95% of read operations will see the most recent value.

**k-staleness:** $k$-staleness determines the probability of a read returning a value within a bounded number of versions. This estimate can be useful for estimating the probability of experiencing monotonic reads, under which users will never read older versions than they have already read (i.e., reads do not "go back in time") [8].

**$\langle$k, t$\rangle$-staleness:** We can combine $t$-visibility and $k$-staleness to consider both time- and version-based staleness together, called $\langle k, t\rangle$-staleness. For example, if a data store configuration has $\langle k, t\rangle$-staleness of 75% at 10ms and 3 versions, then 10ms after the last write completes, 75% of reads will return a value that is no more than 3 versions old.

# 2. USAGE SCENARIOS

Quantitative consistency metrics are useful for a variety of services that can tolerate eventual consistency. In this section, we outline three use cases via short vignettes in the context of a hypothetical microblogging service, DBSoc.

## 2.1 Dynamic Reconfiguration

Without quantitative guidance, choosing the correct values for replication parameters is a difficult task. We believe that, instead of reasoning about low-level replication settings, application writes should instead declaratively specify their objectives in the form of higher-level service agreements and allow the system to adjust its parameters to meet this goal. For example, in Dynamo-style systems like Cassandra, applications can choose the minimum number of replicas to read from ($R$) and write to ($W$). If $R$+$W$ is greater than the number of replicas ($N$), operations will achieve "strong" consistency (regular register semantics), and, if not, the system provides eventual consistency. With a replication factor of $N = 3$, we have several options for eventually consistent operation: for example, $\langle R{=}W{=}1\rangle$ is guaranteed to be fastest, but it is also the "least consistent," while $\langle R{=}2, W{=}1\rangle$ is "more consistent" but slower. Choosing the right configuration is non-trivial, especially without data regarding the effect of a given choice. Instead of reasoning about $R$ and $W$, service operators should express their requirements in terms of tolerable staleness:

*DBSoc's data scientists have learned that their users respond negatively to slow update propagation, and the company sets a t-visibility target of 500ms at the 99.9th percentile for their back-end data store requests (while minimizing latency).*

How should we configure the replication parameters for a given SLA? One approach is to manually tune the system—this is straightforward but is not always robust to changing conditions:

*Based on feedback from their infrastructure team, the developers set $R$=$W$=1 for their Dynamo-based data store. This meets the t-visibility consistency SLA, but, during peak traffic, the consistency SLA is sporadically violated.*

Instead of making a static assignment, metrics-enabled systems can auto-tune the replication parameters for each request. Consistency monitoring allows us to monitor SLA violations, while consistency prediction allows us the system to test replication parameter changes before making them:

*A consistency auto-tuner chooses R=W=1 for most traffic but switches to R=2, W=1 during peak workload times. While R=1, W=2 is also a viable solution, the auto-tuner determines that the 99.9th percentile operation latency would suffer since most reads are served from the data store's buffer cache.*

While the literature suggests several dynamic replication schemes [9], we are pleased that these techniques can now become a reality for production-ready data stores and real-world services.

## 2.2 DBA Tooling and Diagnoses

Consistency metrics can also be used in diagnostic tasks and to understand *why* a system is misbehaving. There are a number of system parameters that affect the performance and observed consistency of a distributed data store. However, system administrators currently face two major feature challenges: there is limited information available in terms of real-time consistency properties and a lack of mechanisms to understand how the system will behave as parameters are varied.

*The database administrators at DBSoc have received reports that a high-profile user is seeing very stale data.*

There are a host of consistency configuration options available to data store administrators. Taking Cassandra as an example, the administrators can configure read repair rates, perform active anti-entropy value exchange, and enable or disable replicas. Moreover, there are many causes for inconsistent reads: there may be slow nodes in the cluster or certain keys may form "hot spots."

*The administrators inspect the consistency metrics for each data store shard and identify a misbehaving set of nodes corresponding to a bad top-of-rack switch. They temporarily increase the rate of background version exchange for the shard and begin to spin up a new replica set on a different rack before rebooting the switch.*

We believe consistency metrics should allow standard analytics such as fine-grained drill-downs and roll-ups across both logical data items and physical-layer details like placement and hardware details. If, as in our Cassandra implementation (Section 3), monitoring is performed as a white-box, in-database service, low-level details like network topologies and per-operation latencies will be available to the administrator. Of course, it may be sufficient for most operations to simply experiment with common configuration parameters via prediction, but exposing such advanced analytics functionality is likely useful for power users.

## 2.3 Monitoring and Alerts

Consistency metrics allow new approaches to monitoring:

*DBSoc has a large number of DevOps staff who are responsible for keeping the service online. Currently, their monitoring and pager service is triggered when operation latency is high or if servers fail. As user experience is negatively impacted by inconsistent reads, the CTO configures custom alerts for the DevOps.*

As we discussed in Section 2.1, some parameters like per-request quorum settings, are amenable to SLA-based automatic control. However, there are a number of scenarios where traditional, man-

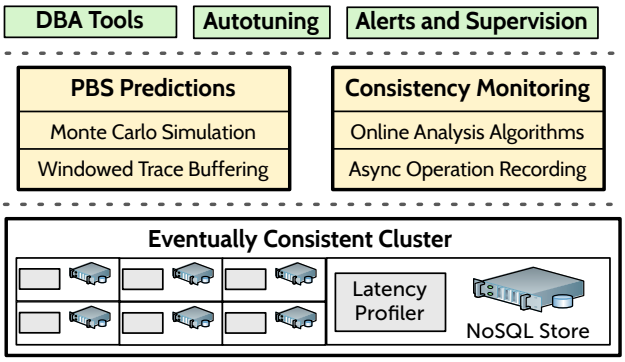



**Figure 1: Architecture for integrating PBS metrics with an existing data store. The PBS prediction and consistency monitoring modules provide metrics used by applications like dynamic query tuning, monitoring, and diagnostic tools.**

ual monitor-and-respond is an acceptable approach. If SLAs cannot be met under any circumstances (e.g., operation latency and $t$-visibility bounds are too restrictive), the correct response is to revise the SLAs or perform more invasive operations (e.g., adding more replicas) that may require active human oversight.

# 3. SYSTEM ARCHITECTURE

In this section, we briefly outline an architecture for providing consistency prediction, monitoring, and verification given our experiences implementing them in two production NoSQL stores—Cassandra and Voldemort. The techniques used to implement PBS are applicable to any system for which we can model the replication protocol and only require lightweight profiling.

## 3.1 Data Store Architecture

The lower two layers of Figure 1 show the changes that are required to integrate PBS-style metrics into an existing distributed data store. In each replica, or storage node, we perform lightweight latency profiling by piggybacking timestamps on messages exchanged between servers. Two separate modules providing consistency prediction and monitoring subsequently process this data. The modules are cleanly separable from the main data store implementation and can be adapted to different systems.

**PBS Prediction:** The PBS prediction module is responsible for calculating the $t$-visibility and $k$-staleness of the data store. This module records the latencies for relevant operations in the data store (in Dynamo-style systems, the respective latencies for writes, acknowledgments, reads, and responses—the PBS *WARS* model) that are sent during replication. The prediction module tracks a moving window of recent latencies across servers, and, when an end-user (or higher-level component) requests a consistency prediction, the module uses the latencies to provide a prediction using Monte Carlo analysis. The module simulates the interactions between thousands of read and write requests that behave stochastically according to the observed distributions. This allows us to estimate $t$-visibility, $k$-staleness, and latency for the operations under a range of parameter choices: replication factors, anti-entropy rates, and node failures. In our Cassandra implementation, we found that thousands of prediction trials typically complete within a second.

**Consistency Monitoring:** While predictions are useful, it is also beneficial to know what consistency a data store is actually providing. While PBS gives a probability distribution function (PDF) of reads that are consistent after a fixed amount of time, consistency monitoring amounts to integrating over the PDF: what percent of reads are actually consistent? Black box monitoring is somewhat more difficult than predictions because determining the latest write effectively requires consensus about the value of the latest write. This requires substantially more coordination than prediction but is the subject of considerable ongoing research [6].

On the other hand, our verifier for Cassandra uses *white box* techniques similar to the PBS predictor. Even though an operation might complete with a reply from a single replica (say $R$=1), the monitoring module asynchronously collects data from all replicas. The monitoring modules periodically exchange timing metadata for writes, which is passed to an online analysis algorithm that detects consistency violations.

## 3.2 Userspace Tools

As we described in Section 2.1, end-users can leverage consistency prediction to build tools to inform and enforce consistency SLAs. A simple, standalone tool can track the consistency and latency profile of queries over time by periodically invoking the PBS predictor. Database administrators can specify desired SLAs in terms of acceptable values for a combination of $t$-visibility, $k$-staleness, and operation latency. Based on the predictions, the enforcement tool can adjust replication parameters to ensure that the SLA is met while minimizing observed latency. For typical replication factors, the number of possible configuration is limited, meaning the tool can likely perform a complete state space search.

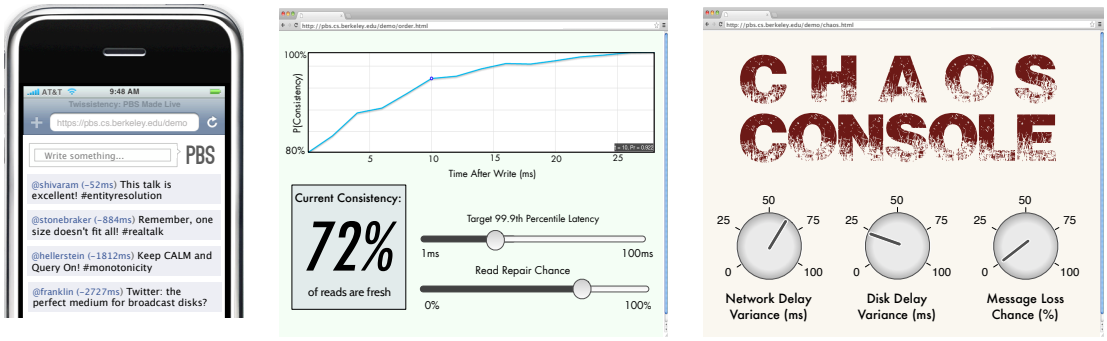
Database administrators can also leverage PBS predictions for greater insight into the latency-consistency trade-offs for their deployment. In our Cassandra implementation, we have added a new command (`predictconsistency`) to `nodetool`, a widely used administration interface. The `predictconsistency` command provides a flexible interface for performing PBS prediction. Data store administrators specify a hypothetical replication configuration ($N$, $R$, $W$) along with a write visibility time and subsequently receive a staleness prediction. This allows administrators to perform "what-if" analysis and project the impact on latency and consistency as their workload changes. We also export the consistency metrics over a Java MBean interface, meaning any tool that can interface with MBeans can receive prediction data.

Finally, administrators can use PBS to monitor the consistency provided by their data store. In contrast to offline consistency verification schemes [5], PBS provides a lightweight mechanism that can be used for (prediction-based) monitoring. Existing monitoring solutions like Ganglia or DataStax's OpsCenter periodically issue PBS prediction requests and plot the values of $t$-visibility and $k$-staleness as a time series. This can then be integrated with rule-based alerting systems to page an operator if the value of $t$-visibility passes beyond a particular threshold.

# 4. DEMO DETAILS

At SIGMOD 2013, we will present an end-to-end demonstration that will show how metrics defined by PBS can be used to improve consistency and user experience. We will also highlight how database administrators can monitor and modify system parameters to trade-off consistency for latency. The demonstration will consist of a user-facing application and several other components.

As a driving use case, we will implement **DBSoc**, a Twitter-like microblogging application. DBSoc will expose a web interface that will be available to all SIGMOD attendees and provide a mobile application that can be used to post messages about the conference. In addition to the messages posted by SIGMOD attendees, we plan to




(a) Mobile Web App (b) DBA and Query Tuning Dashboard (c) Chaos Console

**Figure 2: Mock-ups for the mobile social web application—which attendees will interact with as end-users—and back-end diagnostic views. In the Dashboard, attendees can monitor system consistency and set latency and consistency SLAs. In the Chaos Console, attendees will wreak havoc on the application's Cassandra cluster via several (real-time) configurable failure modes.**

**replay a corpus** of 4,937,001 Tweets from conversations obtained from the Twitter firehose between February and July 2011 [7]. This will help us explore heavier workloads, particularly in the absence of a deluge of activity from SIGMOD attendees. DBSoc will store messages in a **Cassandra cluster** running on Amazon EC2. PBS is already integrated in Cassandra, but we will also augment Cassandra to provide consistency monitoring as described in Section 3.1.

To illustrate the utility of consistency metrics, we will provide attendees with the opportunity to experience consistency from three perspectives: user, administrator, and powerful adversary.

The **user interface** screen will host the DBSoc front-end and will allow attendees to post and read messages (Figure 2(a)). We will also provide a setting that allows users to see how old messages are and manually inspect the end-to-end delay for each message.

To provide the experience of a database administrator or operations technician, we will have a **monitoring and control console** that measures and plots the consistency and latency over 10-second time intervals (Figure 2(b)). Additionally the interface will allow users to modify system parameters like the read repair chance and set consistency SLAs for read and write operations. We will change the system parameters in real-time, and the consistency SLAs will affect all users' actions on the site

Consistency is particularly interesting under changing environmental conditions, so we will allow participants to control parameters like system latency and the performance of several replicas. We will provide a **chaos console**, a separate interface that can be used to inject message delays, model message loss, and artificially slow Cassandra instances to demonstrate their effect on consistency (Figure 2(c)). We will physically implement the actual failures via both JVM manipulation and host-based kernel operations. In addition to improving engagement, this will allow attendees to both induce consistency anomalies and understand the impact of several common failure modes.

## 5. HIGH-LEVEL TAKEAWAY

Our demo will showcase new data store performance and administrative functionality enabled by emerging consistency prediction and monitoring. Given the amount of academic interest in this topic and recent industrial adoption, we believe there is merit in further study of applications that leverage these metrics. As a proof-of-concept of several of the applications we envision, this demonstration can serve as the impetus for a new wave of dynamic, intelligent, and more easily operable distributed data stores. Consistency metrics are coming; what else can we do with them?

**Acknowledgments** We would like to thank Jonathan Ellis for his help in reviewing our Cassandra patch and Aaron Davidson, Aviad Rubenstein, and Anirudh Todi for early experimentation with dynamic quorum policies. This research is supported in part by National Science Foundation grants CCF-1139158, CNS-0722077, IIS-0713661, IIS-0803690, and IIS-0917349, DARPA awards FA8650-11-C-7136 and FA8750-12-2-0331, Air Force OSR Grant FA9550-08-1-0352, the NSF GRFP under Grant DGE-1106400. and gifts from Amazon Web Services, Google, SAP, Blue Goji, Cisco, Clearstory Data, Cloudera, EMC, Ericsson, Facebook, General Electric, Hortonworks, Huawei, Intel, Microsoft, NetApp, NTT Multimedia Communications Laboratories, Oracle, Quanta, Samsung, Splunk, VMware and Yahoo!.

## 6. REFERENCES

[1] Apache Cassandra Jira: "Support consistency-latency prediction in nodetool". `https://issues.apache.org/jira/browse/CASSANDRA-4261`. September 2012. (See also `http://www.bailis.org/blog/using-pbs-in-cassandra-1.2.0/` and `http://pbs.cs.berkeley.edu/#demo`).

[2] D. J. Abadi. Consistency tradeoffs in modern distributed database system design: CAP is only part of the story. *IEEE Computer*, 45(2):37–42, 2012.

[3] P. Bailis, S. Venkataraman, M. J. Franklin, J. M. Hellerstein, and I. Stoica. Probabilistically bounded staleness for practical partial quorums. *PVLDB*, 5(8):776–787, 2012.

[4] S. Davidson, H. Garcia-Molina, and D. Skeen. Consistency in partitioned networks. *ACM Computing Surveys*, 17(3):341–370, 1985.

[5] W. Golab, X. Li, and M. A. Shah. Analyzing consistency properties for fun and profit. In *PODC 2011*, pages 197–206.

[6] M. Rahman, W. Golab, A. AuYoung, K. Keeton, and J. Wylie. Toward a principled framework for benchmarking consistency. In *HotDep 2012*.

[7] A. Ritter, C. Cherry, and B. Dolan. Unsupervised modeling of Twitter conversations. In *HLT-NAACL*, pages 172–180, 2010.

[8] W. Vogels. Eventually consistent. *CACM*, 52:40–44, 2009.

[9] H. Yu and A. Vahdat. Design and evaluation of a conit-based continuous consistency model for replicated services. *ACM TOCS*, 20(3):239–282, 2002.