Quantifying Eventual Consistency For Aggregate Queries*

Neil Burke
Faculty of Computer Science, Dalhousie University
neil.burke@dal.ca

Frank Dehne
School of Computer Science, Carleton University
frank@dehne.net

Andrew Rau-Chaplin
Faculty of Computer Science, Dalhousie University
arc@cs.dal.ca

David Robillard
School of Computer Science, Carleton University
d@drobilla.net

CCS CONCEPTS

• **Information systems** → **Online analytical processing engines**; • **Theory of computation** → **Incomplete, inconsistent, and uncertain databases**;

ACM Reference format:
Neil Burke, Frank Dehne, Andrew Rau-Chaplin, and David Robillard. 2017. Quantifying Eventual Consistency For Aggregate Queries. In *Proceedings of IDEAS '17, Bristol, United Kingdom, July 12-14, 2017,* 9 pages.
https://doi.org/10.1145/3105831.3105836

1 INTRODUCTION

1.1 Background

With the advent of inexpensive cloud computing resources, scalable distributed data stores have surged in popularity [7, 10, 16, 17, 20]. Such systems focus on horizontal scalability and take advantage of cheap, pay by the hour, compute nodes provisioned through the cloud [6]. In doing so, these systems are able to distribute query and insert load across many "shared nothing" compute nodes, improving latency and throughput performance. Consequently, the use of multiple compute nodes increases the likelihood that a node may fail at a given time, making availability a critically important quality [10]. Key-value stores typically address this problem by maintaining N redundant replicas of its data set [10, 16]. In doing so, if a single node in the system fails, $N-1$ nodes replicating the same data remain accessible. Increasing N increases the availability of a system.

However, introducing redundant replication to a system introduces the problem of consistency. Since networks are unpredictable, each insert operation will arrive at the N different replicas at different times. This leads to the data stored on replicas being inconsistent from one another until each insert arrives on each of the N replicas. The direct consequence of inconsistent replicas is the possibility that any read can return an outdated result. One widely popular method of approaching this problem is to implement a quorum consensus algorithm [12, 13, 15]. With a quorum consensus, each write operation is considered complete only after W replica replies confirming a successful write have been received. Likewise, each read operation requires a result from R replicas before a final result is returned. W and R, or the *write quorum* and *read quorum* are used to ensure each write or read operation has been applied to a subset of replicas. Setting $W+R > N$ guarantees consistency because each read will contain at least one of the W most recent writes. In order to decrease latency, many distributed data stores [3, 10, 16] use partial quorums, i.e. $W+R < N$, thereby accepting the possibility of data staleness. For example, the default settings for Apache Cassandra [16] are $N=3$ and $R=W=1$. In their landmark paper "Quantifying eventual consistency with PBS" [4], Bailis et.al. studied this trade-off between operation latency and data consistency. They introduced a *Probabilistically Bounded Staleness* (PBS) consistency measure to calculate expected bounds on staleness for partial quorums and provided an explanation for the experience by practitioners that basic eventual consistency in replicated data stores with partial quorums is often "good enough" in practice.

1.2 Contributions

Modern decision support systems rely heavily on data aggregation, typically provided by an underlying online-analytical processing (OLAP) system [8]. Data aggregation is accomplished by issuing binary associative aggregation queries (for example, *sum* or *max*) over a specified subset of the data items stored in the OLAP system. The issue of data staleness is of particular importance for real-time distributed OLAP systems such as CR-OLAP [11] and Druid [20] that support streaming data ingestion and real-time aggregate queries.

Aggregate Probabilistically Bounded Staleness. In this paper, we present *Aggregate Probabilistically Bounded Staleness (A-PBS).* Inspired by the *Probabilistically Bounded Staleness* (PBS) measure [3] for key-value stores, A-PBS measures staleness for *aggregate* queries. While a key-value query only retrieves a single key, an aggregate query in a distributed OLAP system typically aggregates a large set of data items specified

*Research partially supported by the Natural Sciences and Engineering Research Council of Canada and the IBM Centre For Advanced Studies Canada.

IDEAS '17, July 12-14, 2017, Bristol, United Kingdom

ACM ISBN 978-1-4503-5220-8/17/07…$15.00
https://doi.org/10.1145/3105831.3105836

by a multi-dimensional bounding box in d-dimensional space. Instead of examining the write/read history of the different copies of a single data item as in the case of PBS, the A-PBS measure introduced here depends on the write/read history of the different copies of all data items within a multi-dimensional space, possibly the entire database. This greatly increases the complexity of both measuring and modelling staleness, and clearly distinguishes A-PBS for distributed OLAP systems from the PBS measure for key-value stores.

The A-PBS measure introduced here includes a formal model for describing an OLAP system's data stream and the state of consistency for individual aggregate queries. A-PBS uses both the number of missed inserts and the relative numerical error of the query result to quantify staleness. These different cases do not arise for PBS. We introduce (t, c)-*staleness* for queries that have missed more than c inserts and were issued t time units after the last write, and (t, ϵ)-*staleness* for queries that have a relative numerical error greater than ϵ and were issued t time units after the last write. These measures are then utilized to introduce the following system wide probabilistic staleness measures: *bounded* (t, c)-*staleness* and *bounded* (t, ϵ)-*staleness*.

Simulation. To complement A-PBS, we also present a generic model and corresponding Monte Carlo simulation of data aggregation in quorum-replicated distributed OLAP systems. Given a list of system parameters, our model and simulation can be used to estimate staleness for aggregate queries, thereby enabling the exploration of the trade-offs between consistency and latency in quorum-replicated distributed OLAP systems.

Case Study. We used the *CR-OLAP* [11] quorum-replicated distributed OLAP system for a case study to evaluate our A-PBS measure and Monte Carlo simulation. The *CR-OLAP* system was chosen here because we had access to all the required system parameters. We observed that the staleness of aggregate queries predicted through our A-PBS measure and Monte Carlo simulation was close to the actually observed staleness of aggregate queries in *CR-OLAP*.

Our A-PBS analysis also confirmed our previous observation for *CR-OLAP* that a partial quorum with `[N=3, W=0, R=1]` is "good enough" in practice. Even very large aggregate queries that cover the entire database and are issued only 10 milliseconds after the last insert have ≈80% probability to have zero staleness. If staleness occurs for such aggregate queries, the number of missed data items is expected to be low, as only 0.5 inserts are missed on average. This results in only a very small numerical error in the aggregate query result for the *mean* and *mean* aggregation functions, and very close to zero probability of any numerical error for the *max* aggregation function.

2 RELATED WORK

Several papers have been published on the topic of imprecise or ambiguous data in OLAP systems [5, 9, 18], where rather than examining uncertain but eventually consistent data caused by a lack of synchronization within a distributed system, uncertain dimensional data and measures which are uncertain by nature are explored. For example, all measures may have a certain amount of error with a known distribution. Another work [5] proposes a modified OLAP model which incorporates this concept of uncertain data and, much like this paper, presents different metrics of query correctness within this context. However, since the metrics are within the context of inherently uncertain data, the model and corresponding metrics are inapplicable to eventually consistent OLAP systems studied in this paper.

Another work [19] presents a middleware for distributed OLAP systems which manages replication, insert and query operations through the system to guarantee a certain freshness bound. A "freshness index" is described, which measures between $[0, 1]$, how consistent a replica of a set of data is at the current point in time. In the paper, a model is used where query-answering OLAP nodes receive batch updates from writeable OLTP nodes. Thus, "delay freshness" is used to compute the freshness index: the time of the last update of an node $\tau(c)$, divided by the commit time of the most recent transaction on an OLTP node $\tau(c_0)$.

Much of the work of this paper is rooted in Probabilistically Bounded Staleness (PBS) for key-value stores [2–4], wherein the authors examine eventual consistency from the perspective of quorum-based key-value stores and present a probabilistic metric of staleness for simple partial quorum ($W + R \leq N$) read operations. PBS (k, t)-staleness is defined as a property of a key-value quorum system which, with a specified probability, returns a result within k versions of the most recent write for reads sent t units of time after a write response has been received by at least W replicas for each write relevant to the read.

Another work by the same authors [4] presents a model and corresponding simulation for evaluating PBS (k, t)-staleness are also presented. In it, distributions describing the time taken to read and write keys, as well as send read and write response messages, are sampled to estimate a key-value system's probability of bounded staleness.

3 AGGREGATE PROBABILISTICALLY BOUNDED STALENESS (A-PBS)

In this section, we present *Aggregate Probabilistically Bounded Staleness (A-PBS)*, a means of analyzing consistency in distributed OLAP systems. Like PBS, A-PBS defines metrics examining the consistency of aggregate queries in terms of missed writes. Unlike PBS, since aggregate queries, especially in OLAP, are conventionally numerical by nature [14], new consistency metrics are introduced that view consistency from the perspective of numerical error.

3.1 Data Streams and Queries

While queries in key-value stores essentially pull a single value specified by a key from a node, aggregate queries may involve a much larger percentage of the data. For example, a single query in a typical aggregate OLAP system can operate

on anything between a single point in the system to all points across all nodes in the system. This is an important distinction which divides key-value staleness analysis from aggregate OLAP-style staleness analysis. Therefore, we begin with a set of definitions describing the stream of incoming insert operations to a system, henceforth described as the *data stream*, and the *coverage* and *aggregation function* of an aggregate query, both fundamental for further discussion of correctness in aggregate stores.

Definition 3.1 (Input data stream DATA(n, Λ, D)). DATA(n, Λ, D) is a stream of n insert operations, where each insert, with measure value sampled from the distribution D, is sent to the system according to a Poisson process with rate Λ.

Definition 3.2 (Aggregate query Q). An aggregate query Q is defined by an aggregate function A and a coverage C which describes the percentage of inserts in a data stream required in the computation of the aggregate function.

Figure 1 presents a graphical representation of a simple data stream. Since each insert in the stream is sent to the system according to a Poisson process, the amount of time between adjacent inserts in the stream follows an exponential distribution using the same Λ parameter.

3.2 (t, c)-staleness

We proceed to define our first metric of staleness in an aggregate setting.

Definition 3.3 ((t, c)-staleness). Given an insert data stream DATA(n, Λ, D), a query Q, initiated t units of time after each insert in the stream has been partially committed, has (t, c)-staleness if and only if more than c insert operations covered by the query's bounding box were not included in the computation of the aggregate function A.

We define the notion of a *partially committed* insert as an insert which has been sent to a system, but may or may not be readable. In a partial quorum system, a partially committed insert is an insert that has been written to at least W replicas, but not all N replicas.

Figures 2 and 3 demonstrate queries with (t, c)-staleness and without (t, c)-staleness, respectively, and provide a visual representation of (t, c)-staleness and the "slack" parameters t and c. As time proceeds from left to right, insert operations, represented by white circles, are sent to the system according to a Poisson process with parameter Λ. Consequently, the distance of time between each adjacent insert obeys an exponential distribution with the same Λ parameter. Once each insert has been partially committed (for example, W replica replies have been received in a partial quorum system), t units of time are waited until the query, represented by the black diamond, is initiated.

3.3 (t, ϵ)-staleness

In key-value PBS, read operations are deemed fresh or stale solely by the number of writes missed by the read operation. In A-PBS, the number of missed insert operations is not the only means of qualifying query staleness. The numerical error of the aggregations results from a query can also be used as an indicator of a query's staleness, which can be useful in understanding the practical impact staleness has on a query's result and how different the result would be under a perfectly consistent system.

We refer to the (correct) result of a given aggregate query on a perfectly consistent system as the *true aggregate value*, and the (possibly incorrect) result observed from issuing the same query on an eventually consistent system as the *observed aggregate value*. We define the error of a query as the relative error of the true aggregate value and the observed aggregate value, or, more formally:

Definition 3.4 (Aggregate relative error). The aggregate relative error of a query Q with an observed aggregate value of o and a true aggregate value of v is $\frac{|o-v|}{v}$.

With this in mind, we present a definition, much like (t, c)-staleness, to classify the result of a query of being acceptably consistent, with respect to a relative error:

Definition 3.5 ((t, ϵ)-staleness). Given an insert data stream DATA(n, Λ, D), a query Q with aggregation function A, initiated t units of time after each insert in the stream has been partially committed, has (t, ϵ)-staleness if and only if the query's relative error is greater than ϵ.

(t, ϵ)-*staleness* essentially places an upper bound ϵ on the relative error of a query. A query whose relative error is less than or equal to ϵ is said to have an acceptable amount of error, in which case the query is acceptably consistent (similar to c in (t, c)-staleness). Where (t, c)-staleness measures staleness depending on whether or not points are present during the time the aggregation takes place, (t, ϵ)-staleness measures staleness based on the result of the aggregation. Thus, (t, ϵ)-staleness is dependent on the aggregation function A used by the query, as well as the distribution D of measure values in the data stream.

Another important difference is, unlike (t, c)-staleness, points missed by a query only impact (t, ϵ)-staleness if the missed point has an impact on the final aggregation. For example, missing a single point will not likely have an impact on (t, ϵ)-staleness where the aggregation function used is *max*, as that point would have to be the largest point covered by the query in order to impact the result of the aggregation function. Conversely, missing a single point where the aggregation function is *count* is guaranteed to increase the relative error of the query.

3.4 Probabilistic Staleness

So far, we have examined how staleness impacts individual queries. Since we would like to be able to reason about a system's accuracy as a whole, we now introduce staleness metrics on a probabilistic level.

Definition 3.6 (Bounded (t, c)-staleness). A system for aggregate queries on an input data stream DATA(n, Λ, D) has bounded (t, c)-staleness if and only if, with probability

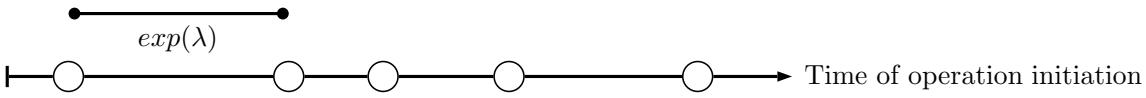

Figure 1: A DATA(5, Λ, D) insert stream. The white circles represent the points in time in which inserts in the stream are sent from the client. The amount of time between adjacent inserts is determined by sampling from an exponential distribution with parameter Λ.

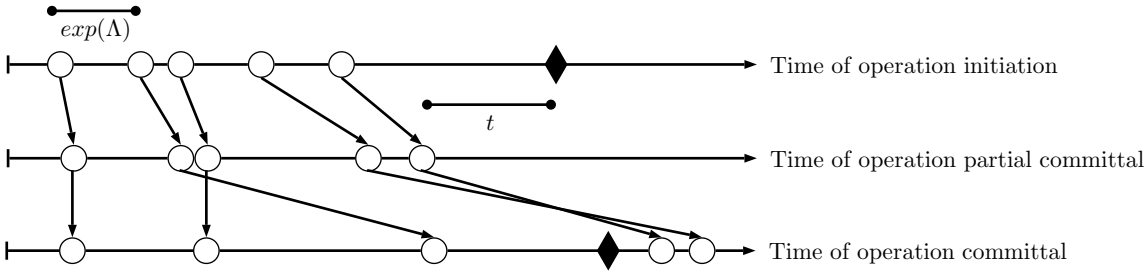

Figure 2: A query that has $(t, c{=}1)$-staleness. The upper bar represents the time of initiation of a query or insert, the middle bar represents the time at which each insert has been partially committed and the bottom bar represents the time at which the corresponding insert is readable, or the cutoff time at which the query begins to read committed inserts. White circles represent inserts, black diamonds represent queries. The last two inserts in the stream and the query are reordered, so more than $c = 1$ inserts are missed.

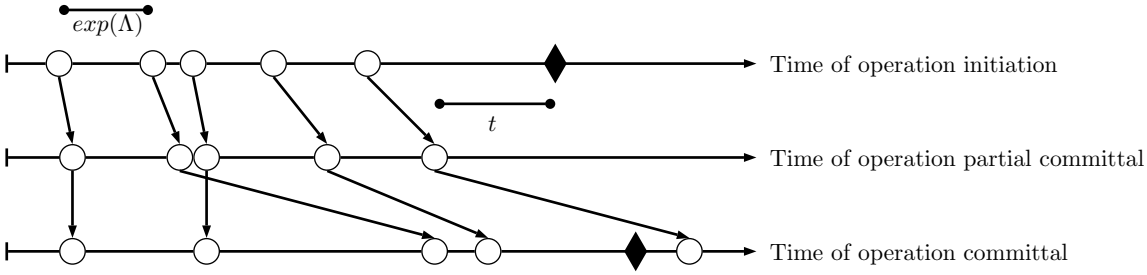

Figure 3: A query that does not have $(t, c{=}1)$-staleness. Since $c = 1$, the reordering of the last insert in the stream and the query does not impact (t, c)-staleness.

p, an aggregate query Q with coverage C does not have (t, c)-staleness.

Definition 3.7 (Bounded (t, ϵ)-staleness). A system for aggregate queries on an input data stream DATA(n, Λ, D) has bounded (t, ϵ)-staleness if and only if, with probability p, an aggregate query Q with coverage C and aggregation function A does not have (t, ϵ)-staleness.

Using *bounded (t, c)-staleness* and *bounded (t, ϵ)-staleness*, the probability of a system being unacceptably inconsistent (determined by c or ϵ), can be described. For example, given a system constantly ingesting a stream of new points at a rate of 10,000 a second, if we would like to approximate the probability of a query returning a result with a relative error of no more than 0.01, we need only to model a stream DATA(n, $\Lambda = \frac{1}{10000}$, D) and find the probability p of bounded $(t{=}0, \epsilon{=}0.01)$-staleness.

4 SIMULATION

In order to evaluate how the different parameters affect staleness within our model, a method for estimating the probability p of bounded (t, c)-staleness and bounded (t, ϵ)-staleness is needed. To accomplish this, we use a Monte Carlo simulation to evaluate repeated trials consisting of an insert stream followed by a query. The result of each trial is whether or not the query has (t, c)-staleness or (t, ϵ)-staleness.

4.1 Aggregate Model

The basis of our simulation is a simple model of a distributed quorum-replicated aggregate system with the following properties:

- The set of multi-dimensional point data and their associated measure values is partitioned into m *partitions*.

- Each partition of the data is redundantly stored in N *buckets*. The set of N buckets which replicate a partition is called a *bucket set*.
- Location data used to determine which buckets store which points are held in a structure called an *index*.
- Insertions and queries are produced by *clients*, and are sent to the index to route operations to the relevant buckets.

Insert operations in this model function similar to a typical quorum-replicated key-value store. Inserts are sent from a client to an index, and are then routed to all N buckets within the relevant bucket set. A response is sent to the client from the index once W buckets have reported a successful write.

Like inserts, queries are initiated from a client and sent to an index. An index that receives a query request must route the query to all N buckets for each bucket set relevant to the query. Once at least R bucket set aggregations are received from all relevant bucket sets, the index aggregates each response and sends the final aggregation to the client.

Figure 4 presents a graphical representation of the model.

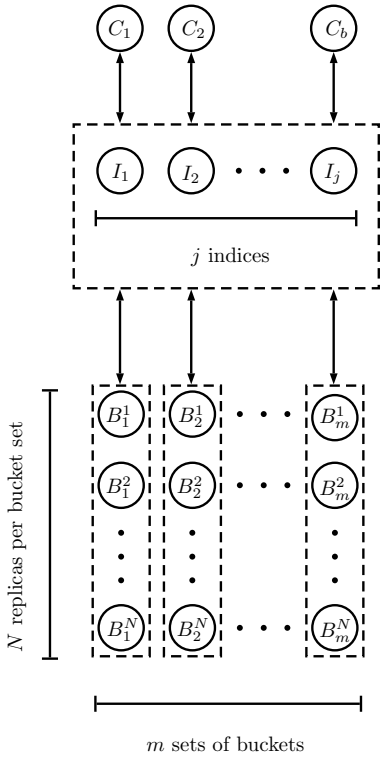

Figure 4: Diagram illustrating the node structure of the aggregate model.

4.2 Simulation Parameters

Each simulated insert represents an insert operation as described in our distributed aggregate model. Using a set of system distributions, which describe the latency timings of insert and query operations of a system, the time at which each simulated insert in the stream is consistent (or readable) at each of N replicas can be determined. The same set of distributions can be used to determine the time a query arrives at each bucket in the system, and the set of R buckets from each bucket set which are the first to arrive at an index. Comparing the insert committal times against the query arrival times can then be used to determine the number of stale inserts.

A summary of the key system parameters is given in Table 1. $T(M)$ is a distribution which describes the network latency of sending a message from an index to a bucket or a bucket to an index. For simplicity, we assume that all query and insertion requests and replies have the same impact on network latency, and thus network latencies for any type of message (insert or query) sent within the system can be drawn from $T(M)$. $T_w(I)$ and $T_w(B)$ are distributions which describe the time taken for an index (I) or bucket (B) to complete the local computation required for an insert. Likewise, $T_r(I)$ and $T_r(B)$ describe the time taken for an index or bucket to complete the local computation required for a query.

4.3 Algorithm

Each trial in our simulation begins by modelling the initiation time of each insert. We refer to the initiation time of an insert as q_i, where i is the number of inserts previous in the stream. We assign the earliest insert in the stream $q_0 = 0$, and all subsequent inserts $q_i = q_{i-1} + exp(\Lambda)$, where $exp(\Lambda)$ is a random sample from an exponential distribution according to the ingestion rate parameter Λ. The committal time of the insert for each of the N relevant buckets is computed by sampling $T_w(I)$, $T(M)$ and $T_w(B)$ and adding the insert's initiation time, q_i. The quorum reply time (the time at which the index has received the W write replies) is computed by adding a sample from $T(M)$ to the W^{th} fastest bucket replica committal time. The committal time of each insert, and the insert's quorum reply time are stored for later use.

Once each insert has been simulated, the query simulation begins by determining the time at which all inserts have been partially committed. This is accomplished by taking the max of each insert's quorum reply time from the previous step. The time is then offset by t to get the time of query initiation. This value is added to a random sample of $T_r(I)$ to get the time at which the index has done its local bucket location lookup. Then, for each bucket B in the system, the time of query arrival is recorded by adding a sample of $T(M)$ to the time the index has finished its local work. To determine which buckets are the first R responders, the time of query arrival from each bucket is offset by a random sampling from $T_r(B)$ and $T(M)$. From this, the time R responses have been received from each bucket set can be observed, and the maximum value is taken to get the time the query has met its read quorum rules for each bucket set.

Name	Description
$T(M)$	Distribution of time taken to send a message from one node to another
$T_w(I)$	Distribution of time taken for local insert work on an index
$T_w(B)$	Distribution of time taken for local insert work on a bucket
$T_r(I)$	Distribution of time taken for local query work on an index
$T_r(B)$	Distribution of time taken for local query work on a bucket

Table 1: The set of system parameters

The query simulation groups all buckets into two sets: those which have responded to the query before or during R responses have been received from each bucket set, and those which have not. Since those that have not responded in time are not included in the result of the query, they may be safely ignored. The buckets which have responded in time will simulate an aggregation operation on all inserts they contain (within the query's coverage), with the exception of points that were committed locally later than the time of query arrival on the current bucket. Afterwards, the partial aggregation of each bucket is combined by bucket set, and finally aggregated to get the observed aggregation value. This process of aggregation is repeated regardless of committal time, to compute the true aggregation value, the result of the aggregation we would expect to get under a perfectly consistent system. With the true value and the observed value, whether or not the query has (t, ϵ)-staleness can be determined by evaluating the error between the true and observed value. Likewise, (t, c)-staleness can be determined by comparing the observed and true counts in relation to the value of c.

5 CASE STUDY

In this section, we use the *CR-OLAP* [11] quorum-replicated distributed OLAP system for a case study to evaluate our A-PBS metrics and Monte Carlo simulation. By using recorded system parameters from *CR-OLAP* in our Monte Carlo simulation, we compare staleness metrics obtained by simulation against actual staleness metrics observed from an OLAP system.

We obtained the operational latency distributions $T_w(I)$, $T_w(B)$, $T_r(I)$, $T_r(B)$ and the network latency distribution $T(M)$ by sampling the amount of time taken to execute or transmit an insert or query operation in *CR-OLAP*. The latencies observed for *CR-OLAP* were from experiments on an Amazon EC2 cloud with 8 `c4.xlarge` nodes using the TPC-DS [1] data set with 8 hierarchical dimensions. Read and write latencies were recorded while processing a workload composed of an even mix of inserts and 100% coverage queries. The first 99 percentiles of the distributions used are shown as cumulative distribution functions in Figure 5. The ingestion rate was measured to be approximately 20,000 inserts per second with $N = 3$, or $\Lambda = \frac{1}{20000}$ seconds between inserts.

To determine the actual probability of bounded (t, c)-staleness on *CR-OLAP*, we first generate a pool of queries of which we know the approximate coverage and aggregation result. To do this, we submit to the system a stream of 100,000 insert operations. A quorum configuration of `[N=1, W=1, R=1]` is used to ensure consistency during the query generation step. After all inserts are complete, the random queries are issued to the system and their results are recorded. Once the pool of queries with known aggregation results and coverage has been generated, we approximate the probability of bounded (t, c)-staleness by repeating several trials of the following. We begin by clearing all previous inserts from the system and issuing the same stream of inserts (this time with a partial or eventually consistent quorum configuration) used during the query generation step. After waiting t units of time after W quorum responses from each insert have been received we issue a query from the pool. We compare the possibly incorrect result against the recorded correct result to determine whether the query in this trial has (t, c)-staleness or (t, ϵ)-staleness.

Figure 6(a) plots the observed *CR-OLAP* probabilities of bounded $(t, c{=}0)$-staleness across increasing t values in the x-axis with varying query coverages. We use the quorum configuration `[N=3, W=0, R=1]` instead of the typical `[N=3, W=1, R=1]`, as setting $W \geq 1$ in our test environment results in the relatively uninteresting case where nearly all queries return correct results. When $W = 0$, queries are initiated t units of time after the last item the data stream has been sent, without waiting for any partial committal responses. Figure 6(b) plots the estimated *CR-OLAP* probabilities of bounded $(t, c{=}0)$-staleness, obtained via simulation. In both (a) and (b), the probability of bounded staleness is proportional to the coverage of the query. Queries with higher coverage have a lower probability of bounded staleness, as they cover a greater number of possibly unreadable points. For 100% coverage queries, the simulated probabilities line up reasonably well with the observed probabilities of bounded staleness. For $<$ 100% coverage queries, the simulation is somewhat pessimistic compared to the observed probabilities, especially at the first 10 milliseconds. This is likely because the simulation's latency distributions are sampled from queries with 100% coverage, which *CR-OLAP* processes faster than lower coverage queries.

In Figure 7(a), we plot *CR-OLAP*'s observed probability of bounded $(t, \epsilon{=}0.00001)$-staleness using various aggregation functions and measure distributions. A small amount of relative error (0.00001) is allowed for demonstration purposes, as setting $\epsilon = 0$ is equivalent to bounded $(t, c{=}0)$-staleness for most aggregation functions. We include the *sum* aggregation function, whose rate of change is steady regardless of the

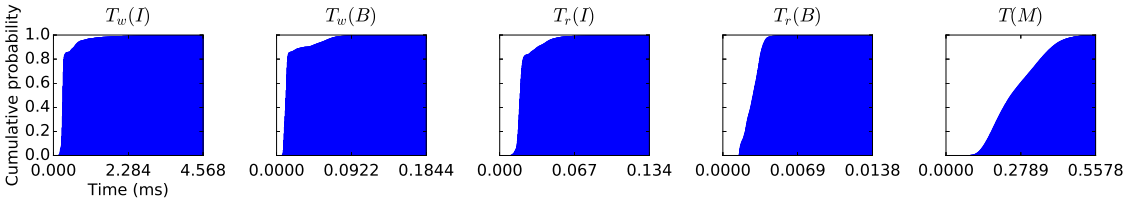

Figure 5: Cumulative distribution functions for distributions $T_w(I)$, $T_w(B)$, $T_r(I)$, $T_r(B)$ and $T(M)$ used in Section 5. For legibility, only the first 99 percentiles are plotted.

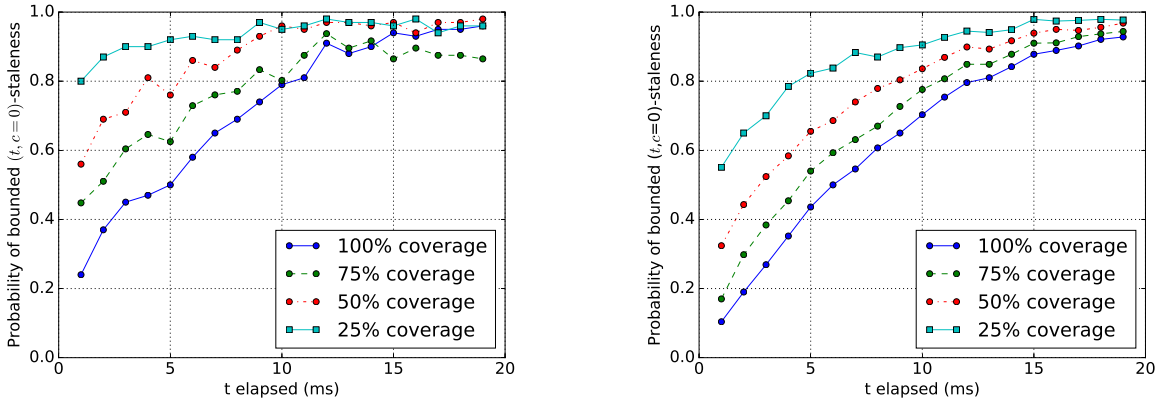

Figure 6: Observed (a) and simulated (b) bounded $(t, c{=}0)$-staleness with `[N=3, W=0, R=1]`.

number of points (similar to *count*), *max*, as its value is essentially determined by a single point and is therefore highly insensitive to randomly selected missing points, and *mean*, as it is a non-monotonic aggregation functions whose rate of change drops as the number of points in the aggregation increases. For the measure distributions, we use the folded normal distribution with mean $\mu = 0$ and standard deviation $\sigma = 1$ to represent a short tailed distribution of measure values, and an exponential distribution with $\lambda = 1$ to emulate the case where the distribution of measure values has a longer tail. Both distributions have been selected to yield only non-negative values in order to further contrast *sum*, which is monotonically increasing if D yields only positive values, with *mean*, whose aggregation value can increase or decrease on the inclusion of a single point. 100% coverage queries on a data stream with 100,000 inserts and quorum configuration `[N=3, W=0, R=1]` were used. At the top of the figure with $> 99\%$ probability of bounded error is the *max* aggregation function for both distributions, illustrating the function's insensitivity to missing points. The *sum* and *mean* aggregation functions have a much lower probability of bounded error, but still have a slightly higher probability than bounded $(t, c{=}0)$-staleness in Figure 6(a), due to the slight amount of slack in ϵ. We note that the *mean* aggregation function has a higher probability of bounded error than *sum*. This is because the amount of relative error incurred by missing a point under *mean* can be offset by missing a point to close to the side opposite of the true mean. Since we are using positive distributions, a missed point under *sum* cannot be offset by missing a negative point of similar magnitude. In Figure 7(b), which plots the simulated probability of bounded error, a small but noticeable gap in the probability of bounded error can be seen for the two measure distributions under *sum* and *mean*. The exponential distribution results in a slightly lower probability with *mean*, as the long, thin tail decreases the likelihood that a pair of missed inserts will offset each other, since the majority of the points lie to the left of the mean. With *sum*, the exponential distribution performs better, as a large part of the total sum is determined by a relatively small number of points with large measure values, who are therefore less likely to be missed compared to the much larger number of points with smaller measure values.

Figure 8(a) shows the expected number of missed inserts with varying coverage observed from *CR-OLAP*. Figure 8(b) shows the average relative error of a 100% coverage query on a data stream of 100,000 items with varying measure distribution and aggregation function, also observed from *CR-OLAP*. Both figures demonstrate that, in *CR-OLAP*, when queries are stale, their results are expected to be only one or two points off from the true result.

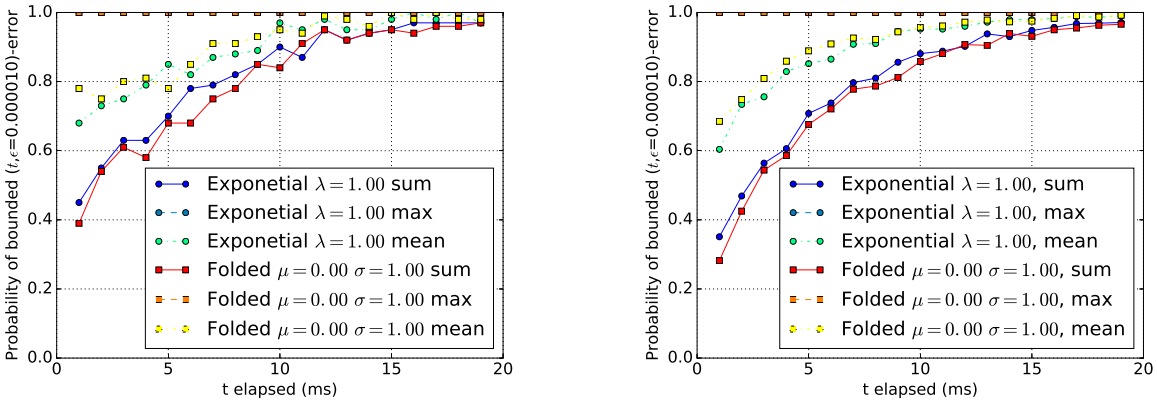

Figure 7: Observed (a) and simulated (b) bounded $(t, \epsilon{=}0.00001)$-staleness with `[N=3, W=0, R=1]`.

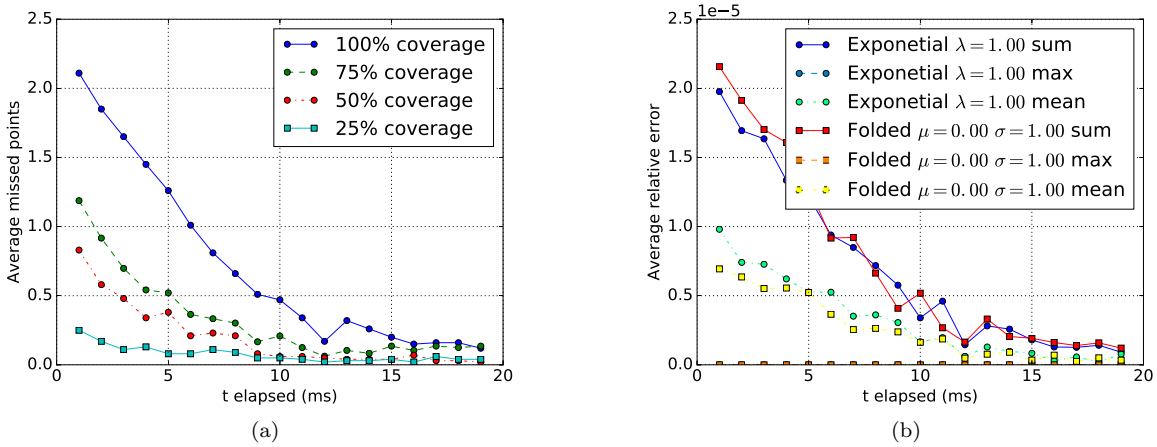

Figure 8: Observed average number of missed inserts (a) and relative error (b) with `[N=3, W=0, R=1]`.

Figure 9 demonstrates the impact a system's read and write speeds have on staleness by plotting the simulated probability of bounded $(t, c{=}0)$-staleness with varying query $(T_r(I), T_r(B))$ and insert $(T_w(I), T_w(B))$ distributions. Under the "fast reads" configuration, an exponential distribution with $\lambda = 1$ (mean of 1 millisecond) is used for the query distributions, and an exponential distribution with $\lambda = 0.5$ (mean of 2 milliseconds) is used for the insert distributions. "Fast writes" uses $\lambda = 1$ for its write distributions and $\lambda = 0.5$ for its read distributions, and "fast reads and writes" uses $\lambda = 1$ for all read and write distributions. In all configurations, the network transmission distribution $T(M)$ is set to $\lambda = 1$. The importance relative query and insert speeds have on bounded (t, c)-staleness is clearly illustrated. When inserts are as fast or faster than writes, more inserts are likely to become accessible by a query in the extended amount of time the query takes for processing at the index, leading to high probabilities of bounded (t, c)-staleness. With faster queries, the opposite is true; queries spend less time being processed at the index and thus queries arrive at buckets to aggregate insertions earlier.

6 CONCLUSION

In this paper, we have presented *Aggregate Probabilistically Bounded Staleness* (*A-PBS*), a measure of staleness for *aggregate queries*. Inspired by the *Probabilistically Bounded Staleness* (PBS) measure [3] for key-value stores, A-PBS measures staleness for *aggregate* queries in distributed OLAP systems that aggregate a large set of data items and depend on the write history of the different copies of all data items covered by the query.

Our A-PBS measure includes a formal model for describing an OLAP system's data stream and the state of consistency for individual aggregate queries. A-PBS uses both the number

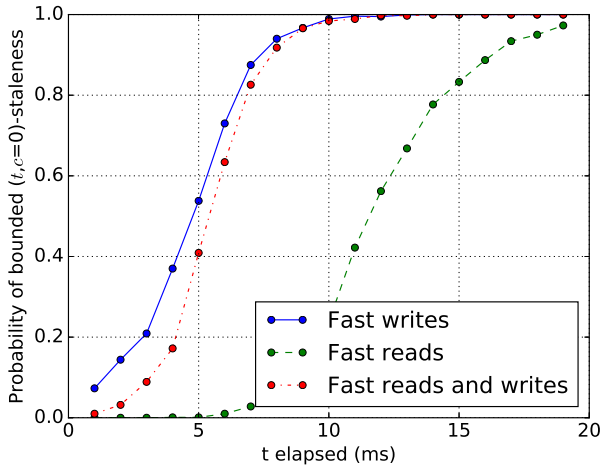

Figure 9: Probability of bounded $(t, c{=}0)$-staleness with varying read and write speeds.

of missed inserts and the relative numerical error of the query result to quantify staleness. To complement A-PBS, we have also presented a generic model and corresponding Monte Carlo simulation of data aggregation in quorum-replicated distributed OLAP systems. Given a list of system parameters, our model and simulation can be used to estimate staleness for aggregate queries, thereby enabling the exploration of the trade-offs between consistency and latency in quorum-replicated distributed OLAP systems.

In a case study evaluating our A-PBS measure and Monte Carlo simulation using the *CR-OLAP* [11] quorum-replicated distributed OLAP system, we observed that the staleness of aggregate queries predicted through our A-PBS measure and Monte Carlo simulation was close to the actually observed staleness of aggregate queries in *CR-OLAP*. For example, the difference between the simulated and observed probability of consistency for a query delayed 10 milliseconds was found to be between ≈1 and 10%, depending on coverage.

Our A-PBS analysis also confirmed our previous observation for *CR-OLAP* that a partial quorum with `[N=3, W=0, R=1]` is "good enough" in practice. Even very large aggregate queries that cover the entire database and are issued only 10 milliseconds after the last insert have ≈80% probability to have zero staleness. If staleness occurs for such aggregate queries, the number of missed data items is expected to be low, as at $t = 10$ milliseconds only only 0.5 inserts are missed on average, resulting in only a very small numerical error in the aggregate query result for the *sum* and *mean* aggregation functions, and very close to zero probability of any numerical error for the *max* aggregation function.

REFERENCES

[1] Transaction processing performance council, TPC-DS (decision support) benchmark. http://www.tpc.org. (????).

[2] Peter Bailis and Ali Ghodsi. 2013. Eventual consistency today: limitations, extensions, and beyond. *Commun. ACM* 56, 5 (2013), 55–63.

[3] Peter Bailis, Shivaram Venkataraman, Michael J Franklin, Joseph M Hellerstein, and Ion Stoica. 2012. Probabilistically bounded staleness for practical partial quorums. *Proc. VLDB* 5, 8 (2012), 776–787.

[4] Peter Bailis, Shivaram Venkataraman, Michael J Franklin, Joseph M Hellerstein, and Ion Stoica. 2014. Quantifying eventual consistency with PBS. *The VLDB Journal* 23, 2 (2014), 279–302.

[5] Doug Burdick, Prasad M Deshpande, TS Jayram, Raghu Ramakrishnan, and Shivakumar Vaithyanathan. 2007. OLAP over uncertain and imprecise data. *The VLDB Journal* 16, 1 (2007), 123–144.

[6] Rick Cattell. 2011. Scalable SQL and NoSQL data stores. *SIGMOD Record* 39, 4 (2011), 12–27.

[7] Fay Chang, Jeffrey Dean, Sanjay Ghemawat, Wilson C. Hsieh, Deborah A. Wallach, Mike Burrows, Tushar Chandra, Andrew Fikes, and Robert E. Gruber. 2008. BigTable: a distributed storage system for structured data. *ACM TOCS* 26, 2 (2008), 4:1–4:26.

[8] Surajit Chaudhuri and Umeshwar Dayal. 1997. An overview of data warehousing and OLAP technology. *SIGMOD Record* 26, 1 (1997), 65–74.

[9] Arbee LP Chen, Jui-Shang Chiu, and Frank SC Tseng. 1996. Evaluating aggregate operations over imprecise data. *Knowledge and Data Engineering, IEEE Transactions on* 8, 2 (1996), 273–284.

[10] Giuseppe DeCandia, Deniz Hastorun, Madan Jampani, Gunavardhan Kakulapati, Avinash Lakshman, Alex Pilchin, Swaminathan Sivasubramanian, Peter Vosshall, and Werner Vogels. 2007. Dynamo: Amazon's highly available key-value store. *ACM SIGOPS Review* 41, 6 (2007), 205–220.

[11] F Dehne, Q Kong, A Rau-Chaplin, H Zaboli, and R Zhou. 2015. Scalable real-time OLAP on cloud architectures. *J. Parallel and Distrib. Comput.* 79 (2015), 31–41.

[12] David K Gifford. 1979. Weighted voting for replicated data. In *Proceedings of the seventh ACM symposium on Operating systems principles*. ACM, 150–162.

[13] David Kenneth Gifford. 1981. *Information Storage in a Decentralized Computer System*. Ph.D. Dissertation. Stanford, CA, USA.

[14] Jiawei Han, Micheline Kamber, and Jian Pei. 2006. *Data mining: concepts and techniques*. Elsevier.

[15] Maurice Herlihy. 1986. A quorum-consensus replication method for abstract data types. *ACM TOCS* 4, 1 (1986), 32–53.

[16] Avinash Lakshman and Prashant Malik. 2010. Cassandra: a decentralized structured storage system. *ACM SIGOPS Review* 44, 2 (2010), 35–40.

[17] Neal Leavitt. 2010. Will NoSQL databases live up to their promise? *Computer* 43, 2 (2010), 12–14.

[18] Sally McClean, Bryan Scotney, and Mary Shapcott. 2001. Aggregation of imprecise and uncertain information in databases. *Knowledge and Data Engineering, IEEE Transactions on* 13, 6 (2001), 902–912.

[19] Uwe Röhm, Klemens Böhm, Hans-Jörg Schek, and Heiko Schuldt. 2002. FAS: a freshness-sensitive coordination middleware for a cluster of OLAP components. In *Proceedings of the 28th international conference on Very Large Data Bases*. VLDB Endowment, 754–765.

[20] F Yang, E Tschetter, X Leaute, N Ray, G Merlino, and D Ganguli. 2014. Druid: A real-time analytical data store. In *Proc. ACM SIGMOD*.