Self-Tuning Eventually-Consistent Data Stores

Shankha Chatterjee[a], Wojciech Golab[b,1,*]

[a]*Department of Electrical and Computer Engineering, University of Waterloo 200 University Avenue West, Waterloo, Ontario, N2L 3G1, Canada.*
[b]*Department of Electrical and Computer Engineering, University of Waterloo 200 University Avenue West, Waterloo, Ontario, N2L 3G1, Canada.*

Abstract

Replication protocols in distributed storage systems are fundamentally constrained by the finite propagation speed of information, which necessitates trade-offs among performance metrics even in the absence of failures. We focus on the consistency-latency trade-off, which dictates that a distributed storage system can either guarantee that clients always see the latest data, or it can guarantee that operation latencies are small (relative to the inter-data-center latencies) but not both. We propose a technique called *spectral shifting* for tuning this trade-off adaptively to meet an application-specific performance target in a dynamically changing environment. Experiments conducted in a real world cloud computing environment demonstrate that our tuning framework provides superior convergence compared to a state-of-the-art solution.

Keywords: self-tuning, consistency-latency trade-off, eventual consistency, distributed storage

---

1. Introduction

Distributed storage systems form the backbone of essential online services including web search, e-mail, social networking, and shopping. The replication protocols that protect such systems from permanent data loss are fundamentally constrained by the finite propagation speed of information, which necessitates trade-offs among performance metrics even in the absence of failures. In particular, any storage system that is replicated across data centers in different geographies may either guarantee that clients always see fresh data, or guarantee that operation latencies are small relative to the inter-data-center latencies, but not both. This leads to a difficult choice for application developers – bite

---

*Corresponding author.
Email addresses: `sschatte@uwaterloo.ca` (Shankha Chatterjee), `wgolab@uwaterloo.ca` (Wojciech Golab)

[1]Author supported in part by the Natural Sciences and Engineering Research Council (NSERC) of Canada, Discovery Grants Program, by a Google Faculty Research Award, the AWS Cloud Credits for Research program, and the Microsoft Azure for Research program.

the bullet and pay the high latency cost of strong consistency, optimize the system for low latency at the risk of exposing inconsistent data to applications and their users, or strike a compromise.

The search for a meaningful compromise between consistency and latency is challenging. Systems that enable application control over this trade-off mostly do so by implementing a quorum-based replication protocol, and by allowing the programmer to choose the size of the quorum for reading and writing, as in Amazon's Dynamo [1]. The different behaviors achievable using this approach represent a collection of discrete points in the trade-off space, which tends to be quite sparse in geo-replicated systems where latencies for strongly and weakly consistent operations can differ by orders of magnitude. Thus, applications whose requirements lie squarely in-between these discrete points are not always served well by such systems. Recent research prototypes (e.g., [2]) have evaded this problem by allowing applications to declare their consistency and latency targets precisely through service level agreements (SLAs), but these systems are not yet in mainstream use, and moreover they tend to support only restricted forms of consistency, such as deterministically bounded staleness.

Responding to a real world need for flexible performance tuning in distributed storage systems, we propose a technique for automated control over a probabilistic consistency-latency trade-off. Our framework can be layered on top of any key-value storage system that provides read and write operations, and supports eventual consistency – the property that in the absence of updates and failures, all replicas of a given key eventually converge to the same value. Given a target consistency threshold expressed as the proportion of the workload that participates in consistency anomalies, and a system that is unable to meet this threshold, the framework boosts consistency by injecting delays artificially into read and write operations. We introduce a novel technique called *spectral shifting* for calculating the duration of the optimal delay (i.e., one that meets the consistency threshold while minimizing latency), which allows the framework to adapt nimbly to changing network conditions and workload characteristics. Microbenchmark experiments using a practical cloud storage system show that our framework achieves superior convergence as compared to a state-of-the-art solution [3].

2. Background and Definitions

We model a distributed storage system abstractly as a collection of processes $p_1, p_2, ..., p_n$ that communicate by exchanging messages over point-to-point communication channels. The processes simulate a collection of shared read/write register objects, each identified by a unique *key*, using a distributed protocol. The processes and the network are asynchronous, and may suffer benign failures: processes may fail by crashing, and communication channels may drop messages but cannot corrupt or reorder them. The possibility of failure necessitates data redundancy (e.g., replication) to prevent loss of data, but we focus in this paper on the behavior of the system in failure-free executions where processing and network delays are bounded.

A *history* of operations executed by a distributed storage system is a sequence of *steps*, representing the invocations and responses of the procedures `Read` and `Write` (as in [4]). Steps record the time when an operation was invoked or produced a response, as well as the corresponding arguments (if any) and return value. The steps in a history appear in increasing order of time. Invocation and response steps corresponding to the same operation are called *matching*, and we assume that steps are tagged with sufficient information so that all matching pairs can be identified. We assume that every history H is *well-formed* meaning that it satisfies two properties: (i) if H contains a `Read` response step for key k and value v then H also contains a `Write` invocation step for k and v that precedes the response of the `Read`; and (ii) every invocation has a unique matching response and vice-versa. An *operation* is a matching invocation-response pair. A `Write` of value v to key k is denoted abstractly by `WriteOp`(k, v), and a `Read` of value v from key k is denoted by `ReadOp`(k, v). The projection of a history H onto operations that access some key k by reading or writing some distinct values v, v' is denoted $H|k, v, v'$. The invocation and response times of an operation are denoted by the functions `start` and `fin`. Given two operations op_1 and op_2, we say that op_1 *happens before* op_2 in a history H if `fin`$(op_1) <$ `start`(op_2), otherwise we say that op_1 and op_2 are *concurrent*. A history H is *linearizable* if there exists a total order T on the operations in H that extends the happens before relation, and where each `ReadOp` returns the value assigned by the most recent `WriteOp` (preceding the `ReadOp`) to the same key, or the initial value of the key if there is no such `WriteOp` [4]. A history H is *regular* if it satisfies the requirements of linearizability with one exception: a `ReadOp` may (but is not required to) return the value assigned by any `WriteOp` that accesses the same key and with which the `ReadOp` is concurrent in H [5].

The storage system can be implemented in a variety of ways, for example using quorum-based replication, and its internal design determines what correctness property its behaviors satisfy. We are interested in quantifying how far this behavior deviates from a standard correctness properties for read/write registers, such as linearizability and regularity. We choose regularity in particular because it is the strongest property supported (in some configurations) by popular quorum-replicated storage systems, such as Dynamo [1] and its derivatives. Specifically, we use the methodology of Golab, Li and Shah [6] to calculate the proportion of values (read or written) that participate in consistency anomalies with respect to regularity. This technique applied to a history H entails shifting the invocation and response steps of operations conceptually (i.e., in the course of mathematical analysis after H is recorded) in such a way that the time intervals of the operations expand outward, which causes pairs of operations related by "happens before" in H to become concurrent in the transformed history H'. One way to formalize such a transformation is the following:

Definition 1. *The t-relaxation of a history H is a history H_t obtained by decreasing the time of every* `ReadOp` *invocation event and increasing the time of every* `WriteOp` *response event by t time units.*

A t-relaxation of H tends to increase the number of possible total orders T

referred to by the definitions of linearizability and regularity, thus lessening the constraints imposed by these properties. Since we assume that every history is well-formed, it follows easily that for every history H there exists a $t \geq 0$ such that H_t is regular. In particular, such a t occurs when the operation intervals expand to the point where every `ReadOp` is concurrent with a `WriteOp` of the same value to the same key. This optimal value of t is our measure of inconsistency.

Definition 2. *The regular t-value of a history H is the smallest real number $t \geq 0$ such that the regular t-relaxation of H, denoted H_t, is regular.*

As suggested in [7], the inconsistency metric can be interpreted in a more fine-grained manner by considering smaller subhistories of a given history H where all the operations are applied to the same key k and access two distinct values v, v'.

Definition 3. *For any history H, key k, and distinct values v, v', the magnitude of the consistency anomaly due to the interaction of operations on key k that access v or v', denoted by the* scoring function *$\chi(H, k, v, v')$, is defined as the regular t-value of $H|k, v, v'$. Furthermore, $\chi(H, k, v)$ is defined as $\max_{v' \neq v} \chi(H, k, v, v')$.*

As explained in Section 3, under certain assumptions there exists an efficient technique for computing the regular t-value of any history H, as well as a precise relationship between this value and the scoring function $\chi(H, k, v, v')$.

3. Efficient Computation of the Inconsistency Metric

Following [7], we note that the regular t-value for a history can be computed in polynomial time under the following assumption, which we make henceforth:

Assumption 4. *For any history H and any distinct operations op_1, op_2 in H, if op_1 writes v_1 to key k and op_2 writes v_2 to the same key k then $v_1 \neq v_2$.*

The above assumption combined with our definition of a well-formed history means that each history has an implicit "reads from" mapping:

Definition 5. *For any history H that satisfies Assumption 4 and any read operation* `ReadOp`(k, v) *in H, the unique operation* `WriteOp`(k, v) *in H is called the* dictating write *of the read.*

Efficient computation of the regular t-value for a history H exploits the observation that consistency anomalies can be attributed to the interaction of operations accessing only two distinct values with respect to the same key [8].

Theorem 6. *For any history H, the regular t-value of H is equal to the following expression*

$$\max_{\text{key } k,\text{ value } v,\text{ value } v' \neq v} \chi(H, k, v, v')$$

where the maximum is interpreted as zero unless some key k is accessed with respect to at least two values in H.

Proof. If no key k in H is accessed with respect to at least two values in H then H is regular under Assumption 4 and the well-formedness criterion defined in Section 2. In that case the theorem holds since the regular t-value of H is 0.

Now suppose that at least one key is accessed with respect to at least two values in H. Let t denote the regular t-value of H (see Definition 2). Let H_t denote the t-relaxation of H (see Definition 1), which is regular. Since H_t is regular, it follows that $H_t|k, v, v'$ is regular for any k, v, v', which implies that $\chi(H, k, v, v') \leq t$. Thus, it follows that the regular t-value of H is an upper bound on the value of the scoring function:

$$t \geq \max_{\text{key } k, \text{ value } v, \text{ value } v' \neq v} \chi(H, k, v, v')$$

To complete the proof, suppose for contradiction that the above upper bound is not tight, meaning that

$$t > z > \max_{\text{key } k, \text{ value } v, \text{ value } v' \neq v} \chi(H, k, v, v')$$

for some real number z. This implies that H_z is not regular. Transform H to H', and hence H_z to H'_z, by removing any read that is concurrent with its dictating write. Then H'_z is not linearizable, it follows from Gibbons and Korach's characterization of linearizability [8] that there exists some key k and there exist some distinct values v, v' such that $H'_z|k, v, v'$ is not linearizable. Since the transformation from H_z to H'_z ensures that H'_z is regular if and only if it is linearizable (i.e., makes regularity equivalent to linearizability), it follows that $H'_z|k, v, v'$ is not regular either. This, in turn, implies that $\chi(H'_z, k, v, v') > 0$, and hence $\chi(H', k, v, v') > z$, which contradicts the earlier supposition that z is an upper bound on the value of the scoring function. □ □

Theorem 6 justifies formally the use of the scoring function $\chi(H', k, v, v')$ in the analysis of consistency by relating the value of this function in a precise way to the t-regular value of a history H. Thus, all consistency anomalies can be quantified with reference to one key and a pair of distinct values, similarly to the approach taken in [9, 7]. In the remainder of this section, we describe the mathematical formula for the scoring function. To that end, we first introduce some definitions modeled after Gibbons and Korach's [8].

Definition 7. *For any history H, key k, and value v written to k in H, the* cluster *$C(H, k, v)$ is the subset of operations in H of the form* `WriteOp`(k, v) *or* `ReadOp`(k, v).

Definition 8. *For any history H, key k, and value v accessed with respect to k in H, the* regularized cluster *$C'(H, k, v)$ is the subset of the cluster $C(H, k, v)$ excluding any reads that are concurrent with their dictating write.*

Definition 9. *For any history H, key k, and value v accessed with respect to k in H, the corresponding* zone *is the time interval denoted by $Z(H, k, v)$ and defined as follows. If $C'(H, k, v)$ contains at least one read, then*

$$Z(H, k, v) = [\min_{op \in C'(H,k,v)} \texttt{fin}(op), \max_{op \in C'(H,k,v)} \texttt{start}(op)]$$

where the left endpoint always corresponds to the finish time of $\texttt{WriteOp}(k,v)$ *and the right endpoint always corresponds to the start time of some* $\texttt{ReadOp}(k,v)$ *in* $C'(H,k,v)$*. Otherwise* $C'(H,k,v)$ *comprises a single write, and*

$$Z(H,k,v) = [\texttt{start}(\texttt{WriteOp}(k,v)), \texttt{fin}(\texttt{WriteOp}(k,v))]$$

A zone is called forward *if* $C'(H,k,v)$ *contains a* $\texttt{ReadOp}(k,v)$ *(first case above), otherwise it is called* backward *(second case). The functions* min *and* max *applied to a zone denote the leftmost and rightmost points in the zone's time interval, respectively.*

With Definitions 7–9 in mind, the closed-form mathematical formula for the scoring function is stated in Theorem 10.

Theorem 10. *For any history* H*, key* k *accessed in* H*, and distinct values* v, v' *accessed using key* k *in* H*, the scoring function* $\chi(H,k,v,v')$ *is equal to the following formula:*

1. *if* $Z(H,k,v)$ *and* $Z(H,k,v')$ *are both forward zones that overlap, and* $\texttt{WriteOp}(k,v)$ *happens before* $\texttt{WriteOp}(k,v')$*, then* $\chi(H,k,v,v') =$

$$\min\left\{\min\left(\frac{1}{2}\left(\max(Z(H,k,v)) - \min(Z(H,k,v'))\right), \frac{1}{2}\left(\max(Z(H,k,v')) - \min(Z(H,k,v))\right)\right),\right.$$
$$\left.\max\left(\texttt{start}(\texttt{WriteOp}(k,v')) - \texttt{fin}(\texttt{WriteOp}(k,v)), \frac{1}{2}\left(\max(Z(H,k,v')) - \min(Z(H,k,v'))\right)\right)\right\}$$

2. *if* $Z(H,k,v)$ *and* $Z(H,k,v')$ *are both forward zones that overlap, and* $\texttt{WriteOp}(k,v')$ *happens before* $\texttt{WriteOp}(k,v)$*, then the formula is as in the previous case, with* v *and* v' *interchanged*

3. *if* $Z(H,k,v)$ *and* $Z(H,k,v')$ *are both forward zones that overlap, and* $\texttt{WriteOp}(k,v)$ *is concurrent with* $\texttt{WriteOp}(k,v)$*, then* $\chi(H,k,v,v') =$

$$\min\left\{\min\left(\frac{1}{2}\left(\max(Z(H,k,v)) - \min(Z(H,k,v'))\right), \frac{1}{2}\left(\max(Z(H,k,v')) - \min(Z(H,k,v))\right)\right),\right.$$
$$\left.\min\left(\frac{1}{2}\left(\max(Z(H,k,v)) - \min(Z(H,k,v))\right), \frac{1}{2}\left(\max(Z(H,k,v')) - \min(Z(H,k,v'))\right)\right)\right\}$$

4. *if* $Z(H,k,v)$ *is a forward zone,* $Z(H,k,v')$ *is a backward zone, and* $Z(H,k,v)$ *is a superset of* $Z(H,k,v')$ *then* $\chi(H,k,v,v') =$

$$\min\left\{\frac{1}{2}\left(\max(Z(H,k,v)) - \max(Z(H,k,v'))\right), \min(Z(H,k,v')) - \min(Z(H,k,v))\right\}$$

5. *if* $Z(H,k,v')$ *is a forward zone,* $Z(H,k,v)$ *is a backward zone, and* $Z(H,k,v')$ *is a superset of* $Z(H,k,v)$*, then the formula is as in the previous case, with* v *and* v' *interchanged*

6. *otherwise*

$$\chi(H,k,v,v') = 0$$

Proof. We must show that the given formula indeed equals the regular t-value of $H|k, v, v'$.

Case 1: $Z(H, k, v)$ and $Z(H, k, v')$ are both forward zones that overlap, and $\texttt{WriteOp}(k, v)$ happens before $\texttt{WriteOp}(k, v')$. Then there exists a point in time t_c such that $\texttt{WriteOp}(k, v)$ and $\texttt{WriteOp}(k, v')$ both finish before t_c, and such that there exist reads $\texttt{ReadOp}(k, v)$ and $\texttt{ReadOp}(k, v')$ that both start after t_c. This scenario, illustrated in Figure 1, constitutes a regularity anomaly, and is

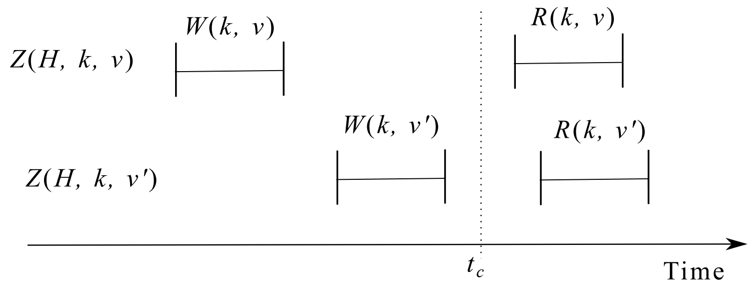

Figure 1: Example scenario in Case 1.

resolved in any t-relaxation where t_c does not exist with respect to H_t. There are three ways in which such a resolution may occur: (i) the t-relaxation narrows the gap between the writes and the reads by $2t$ since the response time of the write is increased by t and the invocation time of the read is decreased by t; (ii) the t-relaxation turns one of the forward zones into a backward zone; and (iii) the t-relaxation turns both of the forward zones into backward zones. In scenario (i), the t-relaxation must be in the following amount:

$$\min\left(\frac{1}{2}\left(\max(Z(H, k, v)) - \min(Z(H, k, v'))\right), \frac{1}{2}\left(\max(Z(H, k, v')) - \min(Z(H, k, v))\right)\right)$$

Scenario (ii) is applicable only to $Z(H, k, v')$, since a t-relaxation that converts $Z(H, k, v)$ into a backward zone requires a greater t than the one in scenario (i). In this case t must be at least $\frac{1}{2}\left(\max(Z(H, k, v')) - \min(Z(H, k, v'))\right)$ to make $Z(H_t, k, v')$ a backward zone, and at least $\texttt{start}(\texttt{WriteOp}(k, v')) - \texttt{fin}(\texttt{WriteOp}(k, v))$ to allow $\texttt{WriteOp}(k, v')$ to take effect before $\texttt{WriteOp}(k, v)$. Thus, scenario (ii) corresponds to a t-relaxation of

$$\max\left(\texttt{start}(\texttt{WriteOp}(k, v')) - \texttt{fin}(\texttt{WriteOp}(k, v)), \frac{1}{2}\left(\max(Z(H, k, v')) - \min(Z(H, k, v'))\right)\right)$$

Scenario (iii) need not be considered because the regularity anomaly is resolved by (i) with a smaller value of t. Thus, the value of the score function $\chi(H, k, v, v')$ is the minimum of the values for scenarios (i) and (ii), in agreement with the stated formula.

Case 2: $Z(H, k, v)$ and $Z(H, k, v')$ are both forward zones that overlap, and $\texttt{WriteOp}(k, v')$ happens before $\texttt{WriteOp}(k, v)$. The analysis is analogous to Case 1 with v and v' interchanged.

Case 3: $Z(H, k, v)$ and $Z(H, k, v')$ are both forward zones that overlap, and `WriteOp`(k, v) is concurrent with `WriteOp`(k, v'). This scenario, illustrated in

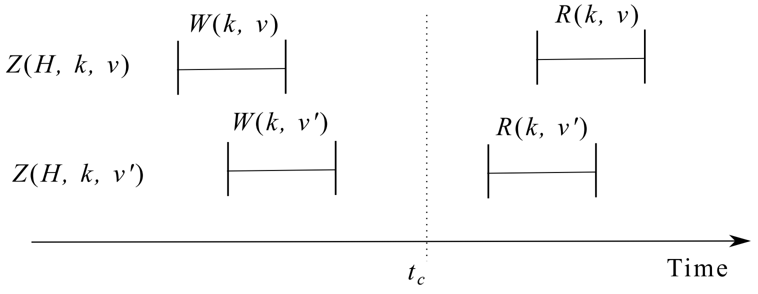

Figure 2: Example scenario in Case 3.

Figure 2, constitutes a regularity anomaly similarly to Case 1. A t-relaxation removes it in one of three scenarios. Scenario (i) is analyzed in the same way as in Case 1, leading to the same formula:

$$\min\left(\frac{1}{2}\left(\max(Z(H,k,v)) - \min(Z(H,k,v'))\right), \frac{1}{2}\left(\max(Z(H,k,v')) - \min(Z(H,k,v))\right)\right)$$

Scenario (ii) is applicable only to either $Z(H, k, v)$ or $Z(H, k, v')$. In this case t must be at least $\frac{1}{2}(\max(Z(H,k,v)) - \min(Z(H,k,v)))$ to make $Z(H_t, k, v)$ a backward zone, or at least $\frac{1}{2}(\max(Z(H,k,v')) - \min(Z(H,k,v')))$ to make $Z(H_t, k, v')$ a backward zone. Thus, the relaxation in scenario (ii) is in the amount of

$$\min\left(\frac{1}{2}\left(\max(Z(H,k,v)) - \min(Z(H,k,v))\right), \frac{1}{2}\left(\max(Z(H,k,v')) - \min(Z(H,k,v'))\right)\right)$$

Scenario (iii) once again need not be considered because the regularity anomaly is resolved by (i) with a smaller value of t. Thus, the value of the score function $\chi(H, k, v, v')$ is the minimum of the values for scenarios (i) and (ii), in agreement with the stated formula.

Case 4: $Z(H, k, v)$ is a forward zone, $Z(H, k, v')$ is a backward zone, and $Z(H, k, v)$ is a superset of $Z(H, k, v')$. Then `WriteOp`(k, v) happens before `WriteOp`(k, v'), which happens before some `ReadOp`(k, v). This scenario, illustrated in Figure 3, constitutes a regularity anomaly that can be resolved using a t-relaxation in two ways: (i) `WriteOp`(k, v) becomes concurrent with `WriteOp`(k, v'); and (ii) the `ReadOp`(k, v) operation with the latest invocation time becomes concurrent with `WriteOp`(k, v'). In scenario (i), the t-relaxation narrows the gap between the two writes by t, and must be in the amount of

$$\min(Z(H,k,v')) - \min(Z(H,k,v))$$

In scenario (ii), the t-relaxation narrows the gap between `WriteOp`(k, v') and any `ReadOp`(k, v) by $2t$, and must be in the amount of

$$\frac{1}{2}\left(\max(Z(H,k,v)) - \max(Z(H,k,v'))\right)$$

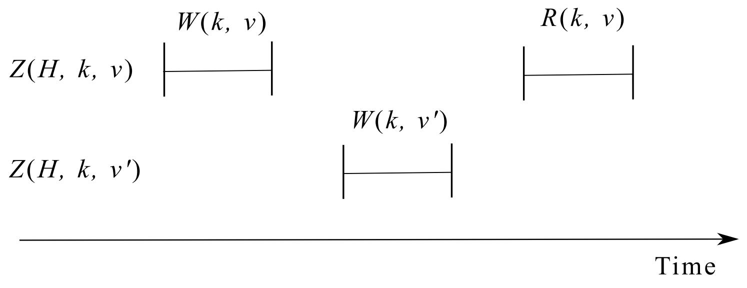

Figure 3: Example scenario in Case 4.

Thus, the value of the score function $\chi(H, k, v, v')$ is the minimum of the values for scenarios (i) and (ii), in agreement with the stated formula.

Case 5: $Z(H, k, v')$ is a forward zone, $Z(H, k, v)$ is a backward zone, and $Z(H, k, v')$ is a superset of $Z(H, k, v)$. The analysis is analogous to Case 4 with v and v' interchanged.

Case 6: Under any other combination of $Z(H, k, v)$ and $Z(H, k, v')$, it is straightforward to show that $H|k, v, v'$ is regular. In particular, the regular total order over operations is obtained by first arranging a regular permutation of the operations in the zone with the smaller min endpoint, followed by a regular permutation of the operations in the other zone. Thus, the regular t-value of $H|k, v, v'$, and hence the scoring function $\chi(H, k, v, v')$, is zero. □ □

4. Spectral Shifting

In this section we present a framework called SPECSHIFT for trading off operation latency against consistency by slowing down operations using artificial delays [10, 3]. Such explicit delays are similar qualitatively to implicit delays arising from client-server interactions in distributed protocols, for example where a process requests data from a majority quorum of replicas instead of reading or writing locally. Specifically, longer delays tend to improve consistency similarly to larger partial quorums [11]. In eventually consistent systems where replicas are updated asynchronously, an artificial delay equal to the sum of the processing delay and one-way network delay is, informally speaking, sufficient to counteract the latency of the replication protocol and ensure regularity. In comparison, quorum operations require two network delays or one round trip. However, if the network and processing delays are unbounded in the worst case, protocols based on artificial delays cannot guarantee regularity deterministically, in contrast to quorum-based protocols. Instead, artificial delays can in some cases provide an attractive *probabilistic* consistency-latency trade-off whereby regularity is attained for a large fraction of the workload at a latency that is substantially lower than using quorum operations.

Our approach to probabilistic consistency-latency tuning is a feedback control mechanism that combines empirical measurement with probabilistic analysis. Before explaining the details, we first introduce some relevant definitions.

Definition 11. *Let H be a history of operations on key k where m distinct values are written: $v_1, v_2, ..., v_m$. Let χ_i denote the score $\chi(H, k, v_i)$ for $i \in [1, m]$ (see Definition 3). Let $\phi(H) = m$ denote the total number of scores for H, counted with multiplicity.* The frequency of a score $j \in \mathbb{Z}^{\geq 0}$, *denoted $freq(j, H)$ is the number of scores in $\chi_1, \chi_2, ..., \chi_m$ equal to j. The* score set *$S(H) = \{\chi_1, \chi_2, ..., \chi_m\}$ is the set of unique scores in a history H.*

Definition 12. *The* score histogram *for a given history H of operations is a collection of* bins, *$b_0, b_1, ..., b_{\max(S(H))}$, where bin $b_i = freq(i, H)$ for $0 \leq i \leq \max(S(H))$.*

The score histogram captures the full "spectrum" of regularity anomalies arising in a history H, and enables a precise calculation of the optimal artificial delay (AD) with respect to a given consistency target defined as a particular proportion of positive scores. The actual proportion of positive scores in a history H is denoted by $I(H) = \frac{\phi(H) - freq(0,H)}{\phi(H)}$, and may be higher than or lower than the target. If $I(H)$ exceeds the target then the AD must be increased to boost consistency at the expense of greater latency. On the other and, if $I(H)$ is below the target then the AD can be decreased to reduce latency while maintaining the desired level of consistency. The optimal AD establishes equality between $I(H)$ and the target, and may change in response to variations in network conditions and the workload mixture. For example, a rise in the network delay or processing delay due to a load spike may increase the optimal AD, requiring more latency to meet the same consistency target, whereas a decrease in the arrival rate of storage operations may lower the optimal AD, allowing a latency reduction.

The tuning framework injects the computed artificial delay d at the end of a `WriteOp` and at the beginning of a `ReadOp`, which stretches the boundaries of these operations. In practical terms, this is achieved by a adding a thin layer of software on top of a distributed storage system that delays the execution or reads and the response of writes either at clients or at servers. The effect of the AD on the consistency of the storage system is analogous to a t-relaxation (see Definition 1) with $t = d$. Specifically, a t-relaxation reduces the score $\chi(H, k, v, v')$ (and similarly $\chi(H, k, v)$) by t time units if the score was $> t$, or else reduces the score to zero if it was $\leq t$, and so we expect intuitively that an AD of $d = t$ time units should have a similar effect on the actual behavior of the storage system. Thus, reasoning precisely about t-relaxations, which operate on histories at a conceptual level, allows us to compute the optimal AD, which in turn alters the histories actually generated by the storage system.

Using the above observation, we can roughly predict the effect of an AD of d milliseconds on the shape of the score histograms generated by the storage system. If we were to plot the histograms for a history H obtained from the system without ADs, and for a history H' obtained with ADs of d time units, we would expect the histogram for H' to resemble the "tail" of the histogram for H comprising bins $b_{d+1}, b_{d+2}, ...$ In other words, we expect an AD of d time units to shift the spectrum of scores to the left by d bins, hence the name *spectral shifting*.

Definition 13. *For a given history H and any $i, d \in \mathbb{Z}^{\geq 0}$, the* shifted frequency of j *is defined as:*

$$freq\text{-}s(i, H, d) = \begin{cases} \sum_{j=0}^{d} freq(i+j, H) & \text{if } \ i = 0 \\ freq(i+d, H) & \text{otherwise} \end{cases}$$

In general, our tuning framework cannot assume that the initial AD is zero since it must be capable of tuning the delay in either direction from an arbitrary starting point, as required to keep up with a dynamically changing environment. Thus, the goal is to predict the score histogram for a history H' obtained using an AD of d' time units, given as input the score histogram for a history H obtained using an AD of d time units. We refer to d as the *base delay*, and d' as the *target delay*.

Consider first the case where $d \leq d'$. The predicted score histogram for H' has a frequency of $freq\text{-}s(i, H, d' - d)$ for a score $i \in \mathbb{Z}^{\geq 0}$. The accuracy of the prediction is contingent on H and H' reflecting, informally speaking, the same workload, meaning that the read and write invocation rates and inter-invocation times are identically distributed. We expect this correspondence to hold approximately in an *open* system where the latency of operations does not affect the random process that generates these operations. (We comment on open versus closed systems in greater detail later on in Section 5.) The proportion of positive scores for H' can then be predicted using the following formula:

$$I'(H, d) = \frac{\phi(H) - freq\text{-}s(0, H, d' - d)}{\phi(H)}$$

Figure 4 illustrates spectral shifting by presenting score histograms for two histories obtained with AD = 0ms and 15ms. The histogram on the right roughly resembles the tail of the histogram on the left, starting at bin 15. We observe that lower scores have higher frequency and vice-versa. Both the histograms have long tails, indicating that large scores, though rare, exist. Most of the area of both histograms is concentrated towards the left, which indicates that most of the regularity anomalies can be eliminated with smaller delays. However, as we increase the value of the injected AD, there is a diminishing return in terms of reduction in the proportion of positive scores. To eliminate all anomalies, we would have to inject a relatively large AD, resulting in a considerable sacrifice in terms of operation latencies. This underscores the need for intelligent consistency-latency tuning to find the optimal AD to be injected without sacrificing latency needlessly.

The case when $d > d'$ (i.e., the base delay exceeds the target delay) is, on first impression, similar to the case when $d < d'$ since it entails shifting the score histogram in the opposite direction, namely from left to right. However, we cannot simply apply the spectral shifting technique in reverse because the *freq-s* function (see Definition 13) is undefined in this case. More concretely, *freq-s* does not determine frequencies for new bins that appear at the left end of the score spectrum following a shift, whereas in the previous case this frequency

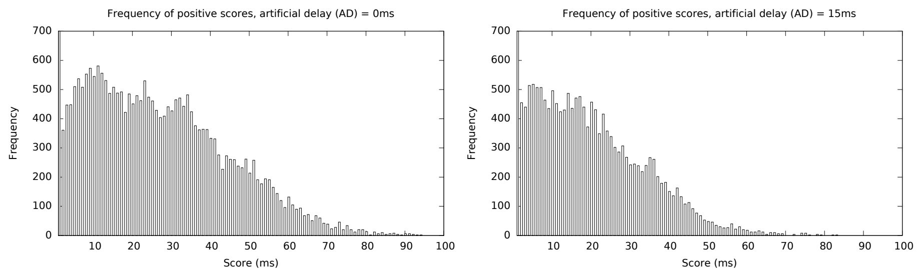

Figure 4: Histograms illustrating the effect of increasing the artificial delay (AD) on the frequency of non-zero $\chi(H, k, v_i)$ scores. (Left: AD = 0ms. Right: AD = 15ms.)

was known to be zero for any bins inserted to the right of the tail. We describe a solution to this problem in the next section.

4.1. Inner-Outer Consistency

To enable bidirectional spectral shifting, we propose a novel technique that captures additional information in the operation history H, enabling a transformation from H to a history H' that has the same read and write invocation rates as well as inter-invocation times, and where the AD is zero. Recall from earlier in Section 4 that the ADs are injected at the beginning of a `ReadOp` and at the end of a `WriteOp`. For read operations, our technique records the time when the AD finishes at the beginning of a `ReadOp`, in addition to the start and finish times. For writes, we record the time when the AD starts at the end of a `WriteOp`. We use these additional timestamps to define *inner* and *outer* operations:

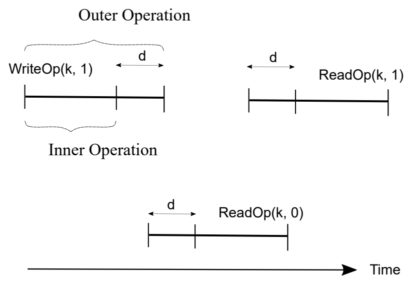

Figure 5: Example of inner and outer operations with an artificial delay of d.

Definition 14. *The* inner operation *for a given* $\texttt{ReadOp}(k, v)$ *with an injected AD of* d *is an operation reading* v *from* k *in the time interval* $[\texttt{start}(\texttt{ReadOp}(k, v))+ d, \texttt{fin}(\texttt{ReadOp}(k, v))]$. $\texttt{ReadOp}(k, v)$ *is the* outer operation *in this context.*

Definition 15. *The* inner operation *for a given* $\texttt{WriteOp}(k, v)$ *with an injected AD of* d *is an operation writing* v *to* k *in the time interval* $[\texttt{start}(\texttt{WriteOp}(k, v)), \texttt{fin}(\texttt{WriteOp}(k, v)) - d]$. $\texttt{WriteOp}(k, v)$ *is the* outer operation *in this context.*

Figure 5 illustrates inner and outer operations in a history H comprising one write and two reads with an AD of d. All three operations are on the same key k with an initial value of 0. The outer operations form a regular history because $\texttt{WriteOp}(k, 1)$ is concurrent with $\texttt{ReadOp}(k, 0)$, which allows the read to return the initial value. However, the history of inner operations, which is similar to H but with an AD of 0 instead of d, has one consistency anomaly because `WriteOp`$(k, 1)$ happens before `ReadOp`$(k, 0)$.

4.2. Adaptive Tuning Framework

We can use SPECSHIFT to construct an adaptive tuning framework that adjusts ADs to meet a target proportion of consistency anomalies while minimizing the ADs to reduce average operation latency. For each iteration of tuning, we take a history of operations H, the current AD d injected to each operation, and a target proportion of positive scores P_t as input. We use these inputs to predict the target AD d_t required to achieve the target proportion P_t. A new history H' is then recorded under the updated AD d_t, and the inputs for the next iteration are d_t, H' and P_t. The process is repeated in a loop until convergence to P_t occurs. The framework uses both physical artificial delays for controlling the behavior of the storage system with respect to consistency and latency, and conceptual artificial delays while reasoning about t-relaxations to compute the optimal correction to the length of the physical delay.

The calculation of d_t given P_t is the dual problem of the one solved by SPECSHIFT, which predicts the proportion of positive scores from the delay. We solve the dual problem as follows, with H denoting the most recently measured history and d denoting the current delay. If the proportion $P(H)$ of positive scores for H matches the target P_t then d is optimal and $d_t = d$. If $P(H) > P_t$, then d is too small, and must be increased. Then d_t is computed (as explained shortly) using the outer operations in H. On the other hand, if $P(H) < P_t$, then d is too large, and must be decreased. Then d_t is computed using the history H_{inner} of inner operations in H. The adjustment to the delay is determined using the following function, with either H itself or H_{inner} used as the input history G:

Definition 16. *For a history G of operations and a target proportion (of positive scores) of P_t, the delay prediction function $D(G, P_t)$ is defined as the smallest non-negative integer d_p that satisfies the following inequality:*

$$\sum_{i=1}^{d_p-1} freq(i, G) \quad \leq \quad \phi(G) - freq(0, G) - P_t \times \phi(G) \quad \leq \quad \sum_{i=1}^{d_p} freq(i, G)$$

The intuition underlying Definition 16 is as follows. The number of positive scores in the input history G is equal to $\phi(G) - freq(0, G)$. In comparison, the desired number of positive scores to meet the target P_t is $P_t \times \phi(G)$. The difference between $\phi(G) - freq(0, G)$ and $P_t \times \phi(G)$ is positive by our choice of G, and represents the number of additional positive scores that must be eliminated by adjusting the delay. A delay adjustment of $+b_p$ is predicted to eliminate positive scores in bins $b_1, b_2, ..., b_p$, and so a rolling total over b_i yields the minimum d_p that is sufficient to reduce the proportion of positive scores below P_t.

The output d_p of the delay prediction function is applied as follows to compute the target delay d_t for the next round of consistency-latency tuning. If G

comprises the outer operations of H (d too small), then $d_t = d + d_p$, otherwise G comprises the inner operations of H (d too large) and $d_t = d_p$.

5. Experimental Evaluation

In this section we compare the convergence of the SPECSHIFT adaptive tuning framework, the PCAP *multiplicative control loop* [3], and a binary search for the optimal AD over the constrained range $[0, 71]$ using a Apache Cassandra deployed in Amazon's Elastic Compute Cloud (EC2). Six Cassandra servers were deployed across three Amazon regions: Oregon, Ireland, and Tokyo.

We ran 20 experiments on a Cassandra cluster, each with a distinct positive integer value of starting delay in the range [0, 90] and a target proportion of consistency anomalies in the range of [0.02, 0.05]. The target proportions are chosen to be small enough to be tolerated in a real-world application. The starting delays are chosen to always be less than the largest one-way network delay between regions, which is 106 ms (between Ireland and Tokyo).

SPECSHIFT, PCAP, and binary search are all implemented as feedback control loops that first measure consistency in a given iteration while holding the AD constant, and then compute an adjusted AD for the next iteration. Each iteration is run for 30 seconds with a throughput of 6000 operations/s and a read-to-write proportion of 0.8. The workload is generated using the Yahoo Cloud Serving Benchmark (YCSB), and keys are drawn from the "latest" distribution, which favors recently chosen keys. We use the number of iterations required by each technique to obtain convergence to within 0.005 of the desired target proportion as the figure of merit for comparisons.

The PCAP multiplicative loop operates by starting with a unit step size and increasing it exponentially at each iteration until the control loop overshoots or undershoots, at which point the direction of the steps is reversed and the step size is reset to unity. The search interval selected for binary search is based on the intuition that the proportion of consistency anomalies is very close to zero when every operation is delayed by the sum of the one-way network delay and processing delay. Thus, the optimal AD to achieve a non-zero target proportion usually lies between 0 and the latter quantity.

We also experimented with a proportional-integral-differential (PID) controller for consistency-latency tuning, but this technique involves tuning additional control parameters k_p, k_d and k_i. Convergence, if achieved, with the values of these parameters suggested in [3] ($k_p = 1$, $k_d = 0.8$, $k_i = 0.5$) is extremely slow and so we have omitted the results.

Figures 6 and 7 illustrate the details for two of the 20 experiments. The target proportion is denoted by a solid horizontal line in the plots. Figure 6 shows the proportion of positive scores and the delay at each iteration on the vertical axis for a starting delay of 0 and a target proportion of 0.05. Figure 7 shows the same for a starting delay of 75 ms and a target proportion of 0.03. In both cases SPECSHIFT converges in one iteration, using *outer* operations in Figure 6 and *inner* operations in Figure 7 to compute the AD adjustment. PCAP is prone

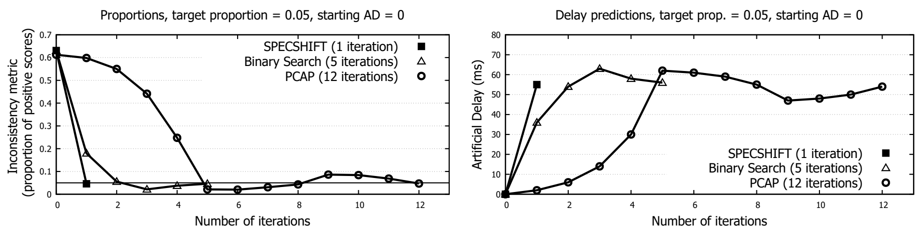

Figure 6: Convergence comparison for target proportion = 0.05, starting AD = 0.

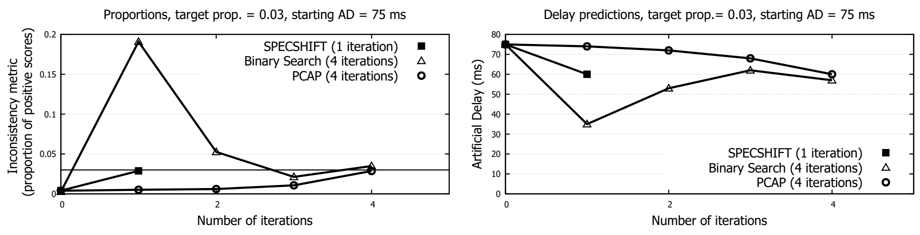

Figure 7: Convergence comparison for target proportion = 0.03, starting AD = 75 ms.

to oscillations and requires more than ten iterations to converge in the first case (Figure 6), though it reaches very close to the target value at the eighth iteration. Binary search is more predictable, and either meets or beats the performance of PCAP. Binary search and PCAP converge faster in the second experiment, partly because the initial and optimal delays are less far apart. The number of iterations required by these two techniques is more sensitive to the specific values of the starting delay and the target proportion.

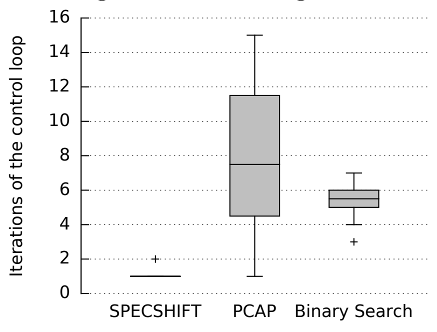

Figure 8: Boxplots showing number of iterations required for convergence by different tuning mechanisms over 20 experiments.

Figure 8 presents data for all 20 runs of the experiment, and shows that the PCAP multiplicative loop takes anywhere between 1 to 15 iterations to converge, with the mean value between 7 and 8. Binary search takes anywhere between 4 to 7 iterations to do the same, with a mean of almost 6. SPECSHIFT, however, takes only one iteration to converge in the vast majority of runs. The plots in Figure 8 shows outliers for SPECSHIFT and binary search. Outliers are defined as values that lie more than one and a half times the length of the box from either end of the box, as is the norm with box-and-whisker plots.

As pointed out earlier in Section 4, the accuracy of the prediction at each

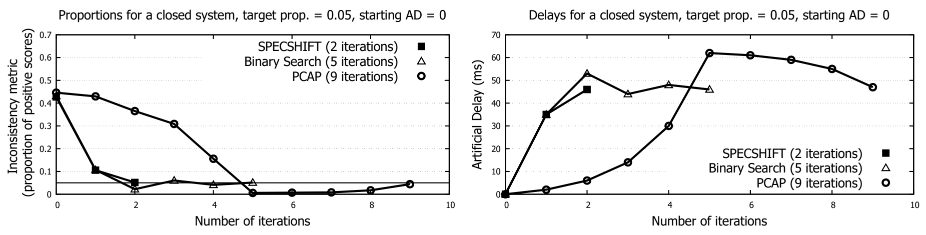

Figure 9: Convergence comparison for closed system analogous to Figure 6.

iteration of SPECSHIFT is contingent on the workload not changing across iterations. In the experiments above, we have assumed an open system where the overall throughput of the system remains unchanged at 6000 ops/s even if the latencies of individual operations vary due to variations in the injected ADs. Figure 9 presents an experiment which compares the three techniques in a closed system, where the individual storage servers operate at peak throughput and the overall throughput of the system decreases with an increase in operation latencies. The throughput drops by half (from 22 kops/s to 11 kops/s) between the starting point of each adaptive loop in figure 9 (AD = 0) and their point of convergence (AD roughly equal to 46ms). The starting delay and target proportion are as in Figure 6. Though SPECSHIFT takes one extra iteration (2 iterations total) to converge in this case, it still converges much more rapidly than PCAP (9 iterations) and constrained binary search (5 iterations).

Overall, SPECSHIFT exhibits the best convergence of the three control loops because it exploits the special structure of the tuning problem by examining the score histograms carefully at each iteration. The other two techniques are more general, but converge more slowly because they make decisions using a small subset of the information harvested using consistency measurements in each iteration, namely the proportion of positive scores. PCAP is based on the principle that the consistency target can be reached more quickly using larger steps, and indeed it crosses the horizontal line representing the target in Figures 6 and 7 about as quickly as binary search, but this does not guarantee fast convergence. At the point where PCAP crosses the target, its step size is relatively large and so it tends to undershoot or overshoot, leading to oscillations. In contrast, binary search uses larger steps initially and then smaller steps as it nears the target, similarly to SPECSHIFT in cases where it requires multiple iterations. The main drawback of binary search is that it must be restarted from the beginning if the optimal delay changes, for example due to a load spike, which causes disruption as the initial artificial delay can be far from optimal. SPECSHIFT and PCAP minimize disruption by adapting continuously, and are more appropriate in a practical environment.

6. Related Work

Recent research on consistency in distributed storage systems has addressed the classification of consistency models, consistency measurement, and the design of storage systems that provide precise consistency guarantees. This body of work is influenced profoundly by Brewer's CAP principle, which states that a distributed storage system must make a trade-off between consistency (C) and availability (A) in the presence of a network partition (P) [12]. The trade-off between consistency and latency is orthogonal to CAP, and comes into consideration even in the absence of failures [13].

Distributed storage systems use a variety of designs that achieve different trade-offs with respect to CAP. Amazon's Dynamo [1] and its derivatives (e.g., Cassandra [14], Voldemort and Riak) use a quorum-based replication scheme [15, 16] that can operate either in CP (i.e., strongly consistent but sacrificing availability) or AP (i.e., highly available but eventually consistent) mode depending on the size of the partial quorum used to execute read and writes, which is determined by client-side consistency settings. Other designs lack such tuning knobs and instead guarantee various forms of strong consistency [17, 18, 19, 20]. A handful of systems allow users to declare requirements with respect to consistency, and adjust parameters internally to fulfill these requirements when possible [21, 22, 2, 23].

Measuring consistency precisely is difficult because consistency anomalies arise from the interplay between multiple storage operations. As a result, some experimental studies measured the convergence time of the replication protocol, which is easier to quantify, rather than consistency actually observed by client applications (e.g., [24, 25]). Other works quantify the observed consistency by counting cycles in a dependency graph that represents the interaction of read and write operations, which is less intuitive than expressing staleness in units of time [26, 27]. This difficulty can be overcome by defining staleness precisely in terms of the additional amount of latency that must be added to storage operations to resolve consistency anomalies [9], which makes it possible to capture in natural way the consistency actually observed by client applications. The consistency metric used in this paper is an adaptation of this technique whereby consistency is defined relative to Lamport's regularity property [5]. The generalization of regularity to multiple writers used in this paper resembles closely the "MWRegWO" property introduced by Shao et al. in [28].

Mathematical models of consistency are generally rooted in the notion of probabilistic quorums [29, 30]. The basic model assumes that each read and write operation accesses a quorum chosen according to a randomized strategy, and no attempt is made to push updates to replicas outside of a write quorum. Thus, the probability that a read quorum intersects with the quorum of a past write operation depends only on the chosen strategy and the number of other write operations applied subsequently. The probabilistically bounded staleness (PBS) model of Bailis et al. matches more closely the behavior of a Dynamo-style storage system, and predicts the probability of reading a stale value t time units after a single write operation is applied [31].

Several lower and upper bounds are known on the latency of operations on read/write registers simulated using message passing. Lipton and Sandberg show that the sum of read and write latencies for a sequentially consistent register cannot be less than the one-way network delay in the worst case [32]. Attiya and Welch strengthen this result and prove a matching upper bound for linearizability in a model with timing assumptions [33]. These results separate protocols that use timing assumptions in the absence of failures, where one network delay suffices, from fault-tolerant asynchronous quorum-based protocols, which incur two network delays (one round trip) to access a quorum of replicas.

Adaptive consistency-latency tuning using artificial delays is proposed in two prior projects. Golab and Wylie propose *consistency amplification*, a feedback control mechanism for supporting probabilistic consistency guarantees by injecting artificial client-side or server-side delays whose duration is determined using consistency measurements [10]. This framework specifies concrete consistency metrics (based on [9]) for quantifying the consistency-latency trade-off, but does not state precisely how the delay should be calculated. Rahman et al. present a similar system called PCAP, where delays are calculated using known techniques: multiplicative and proportional-integral-derivative (PID) feedback control [3]. Their consistency metric ignores write latency and assumes that writes take effect in the order of invocation, hence lacks a precise connection to Lamport's formalism [5]. An earlier thesis by Nguyen demonstrates that the multiplicative control loop used in PCAP is prone to oscillations, and fails to converge at all in some runs even if the optimal delay duration is constant [34].

7. Discussion and Conclusion

In this paper we proposed and evaluated a framework for tuning the probabilistic consistency-latency trade-off in eventually consistent storage systems. Our novel spectral shifting technique analyzes the structure of the underlying optimization problem carefully to reach convergence in a much smaller number of iterations than a competing solution based on a multiplicative control loop [3]. The feedback control approach in general requires collecting operation histories at each iteration of the loop, which can lead to a performance bottleneck. A workaround is to collect histories for a subset of the keys and run the tuning framework at each iteration on the sample history. However, the correctness of predictions in this case would depend on the quality of sampling. The framework described in [3] addresses this problem to some extent by injecting storage operations artificially to gather consistency measurements at each data center, and by combining these measurements using mathematical composition rules. However, the effect of the workload on consistency is modeled less accurately in this approach, leading to a potentially sub-optimal consistency-latency trade-off.

8. Acknowledgments

We are grateful to the anonymous referees for their insightful comments and helpful feedback on this manuscript and its shorter conference version [35].

[1] G. DeCandia, D. Hastorun, M. Jampani, G. Kakulapati, A. Lakshman, A. Pilchin, S. Sivasubramanian, P. Vosshall, W. Vogels, Dynamo: Amazon's highly available key-value store, in: Proc. of the 21st ACM Symposium on Operating System Principles (SOSP), 2007, pp. 205–220.

[2] D. B. Terry, V. Prabhakaran, R. Kotla, M. Balakrishnan, M. K. Aguilera, H. Abu-Libdeh, Consistency-based service level agreements for cloud storage, in: Proc. of the 24th ACM Symposium on Operating Systems Principles (SOSP), 2013, pp. 309–324.

[3] M. R. Rahman, L. Tseng, S. Nguyen, I. Gupta, N. H. Vaidya, Characterizing and adapting the consistency-latency tradeoff in distributed key-value stores, ACM Transactions on Autonomous and Adaptive Systems 11 (4).

[4] M. Herlihy, J. M. Wing, Linearizability: A correctness condition for concurrent objects, ACM Transactions on Programming Languages and Systems 12 (3) (1990) 463–492.

[5] L. Lamport, On interprocess communication, Part I: Basic formalism and Part II: Algorithms., Distributed Computing 1 (2) (1986) 77–101.

[6] W. Golab, X. Li, M. A. Shah, Analyzing consistency properties for fun and profit, in: Proc. of the 30th ACM Symposium on Principles of Distributed Computing (PODC), 2011, pp. 197–206.

[7] W. Golab, M. R. Rahman, A. AuYoung, K. Keeton, I. Gupta, Client-centric benchmarking of eventual consistency for cloud storage systems, in: Proc. of the 34th International Conference on Distributed Computing Systems (ICDCS), 2014, pp. 493–502.

[8] P. Gibbons, E. Korach, Testing shared memories, SIAM Journal on Computing 26 (1997) 1208–1244.

[9] W. Golab, X. Li, M. A. Shah, Analyzing consistency properties for fun and profit, in: Proc. of the 30th ACM SIGACT-SIGOPS Symposium on Principles of Distributed Computing (PODC), 2011, pp. 197–206.

[10] W. Golab, J. J. Wylie, Providing a measure representing an instantaneous data consistency level, US Patent Application 20,140,032,504, filed 2012, published 2014.

[11] M. McKenzie, H. Fan, W. M. Golab, Fine-tuning the consistency-latency trade-off in quorum-replicated distributed storage systems, in: Proc. of the Scalable Cloud Data Management (SCDM) Workshop at the IEEE International Conference on Big Data, 2015, pp. 1708–1717.

[12] E. A. Brewer, Towards robust distributed systems (Invited Talk), in: Proc. of the 19th ACM SIGACT-SIGOPS Symposium on Principles of Distributed Computing (PODC), 2000.

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

[14] A. Lakshman, P. Malik, Cassandra: A decentralized structured storage system, SIGOPS Oper. Syst. Rev. 44 (2) (2010) 35–40.

[15] H. Attiya, A. Bar-Noy, D. Dolev, Sharing memory robustly in message-passing systems, J. ACM 42 (1) (1995) 124–142.

[16] D. K. Gifford, Weighted voting for replicated data, in: Proc. of the 7th ACM Symposium on Operating Systems Principles (SOSP), 1979, pp. 150–162.

[17] F. Chang, J. Dean, S. Ghemawat, W. C. Hsieh, D. A. Wallach, M. Burrows, T. Chandra, A. Fikes, R. E. Gruber, Bigtable: A distributed storage system for structured data, ACM Trans. Comput. Syst. 26 (2).

[18] B. F. Cooper, R. Ramakrishnan, U. Srivastava, A. Silberstein, P. Bohannon, H.-A. Jacobsen, N. Puz, D. Weaver, R. Yerneni, Pnuts: Yahoo!'s hosted data serving platform, PVLDB 1 (2) (2008) 1277–1288.

[19] J. C. Corbett, et al., Spanner: Google's globally-distributed database, in: Proc. USENIX Conference on Operating Systems Design and Implementation (OSDI), 2012, pp. 251–264.

[20] W. Lloyd, M. J. Freedman, M. Kaminsky, D. G. Andersen, Don't settle for eventual: Scalable causal consistency for wide-area storage with COPS, in: Proc. of the 23rd ACM Symposium on Operating Systems Principles (SOSP), 2011, pp. 401–416.

[21] M. S. Ardekani, D. B. Terry, A self-configurable geo-replicated cloud storage system, in: Symp. on Op. Sys. Design and Implementation (OSDI), 2014, pp. 367–381.

[22] S. Krishnamurthy, W. H. Sanders, M. Cukier, An adaptive quality of service aware middleware for replicated services, IEEE Transactions on Parallel and Distributed Systems 14 (2003) 1112–1125.

[23] H. Yu, A. Vahdat, Design and evaluation of a conit-based continuous consistency model for replicated services, ACM Trans. Comput. Syst. 20 (3) (2002) 239–282.

[24] D. Bermbach, S. Tai, Eventual consistency: How soon is eventual? An evaluation of Amazon S3's consistency behavior, in: Proc. of the 6th Workshop on Middleware for Service Oriented Computing (MW4SOC), 2011.

[25] H. Wada, A. Fekete, L. Zhao, K. Lee, A. Liu, Data consistency properties and the trade-offs in commercial cloud storages: the consumers' perspective, in: Proc. of the 5th Biennial Conference on Innovative Data Systems Research (CIDR), 2011.

[26] E. Anderson, X. Li, M. A. Shah, J. Tucek, J. J. Wylie, What consistency does your key-value store actually provide?, in: Proc. of the 6th Workshop on Hot Topics in System Dependability (HotDep), 2010.

[27] K. Zellag, B. Kemme, How consistent is your cloud application?, in: Proc. of the Third ACM Symposium on Cloud Computing (SoCC), 2012, p. 6.

[28] C. Shao, J. L. Welch, E. Pierce, H. Lee, Multiwriter consistency conditions for shared memory registers, SIAM J. Comput. 40 (1) (2011) 28–62.

[29] H. Lee, J. L. Welch, Randomized registers and iterative algorithms, Distributed Computing 17 (3) (2005) 209–221.

[30] D. Malkhi, M. K. Reiter, R. N. Wright, Probabilistic quorum systems, in: Proc. of the 16th ACM Symposium on Principles of Distributed Computing (PODC), 1997, pp. 267–273.

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

[32] R. J. Lipton, J. Sandberg, PRAM: A scalable shared memory, Tech. rep., Princeton University (1998).

[33] H. Attiya, J. L. Welch, Sequential consistency versus linearizability, ACM Trans. Comput. Syst. 12 (2) (1994) 91–122.

[34] S. Nguyen, Adaptive control for availability and consistency in distributed key-values stores, University of Illinois at Urbana-Champaign (2014).

[35] S. Chatterjee, W. Golab, Self-tuning eventually-consistent data stores, in: In Proc. of the 19th International Symposium on Stabilization, Safety, and Security of Distributed Systems (SSS), 2017, pp. 78–92.