Tuning and Predicting Consistency in Distributed Storage Systems

by

Shankha Subhra Chatterjee

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Master of Applied Science
in
Electrical and Computer Engineering

Waterloo, Ontario, Canada, 2017

I hereby declare that I am the sole author of this thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I understand that my thesis may be made electronically available to the public.

Abstract

Distributed storage systems are constrained by the finite speed of propagation of information. The CAP (which stands for consistency, availability, and partition tolerance) theorem states that in the presence of network partitions, a choice has to be made in between availability and consistency. However, even in the absence of failures, a trade-off between consistency and latency of operations (reads and writes) exists. Eventually consistent storage systems often sacrifice consistency for high availability and low latencies. One way to achieve fine-tuning in the consistency-latency trade-off space is to inject artificial delays to each storage operation.

This thesis describes an *adaptive tuning framework* that is able to calculate the values of artificial delay to be injected to each storage operation to meet a specific target consistency. The framework is able to adapt nimbly to environmental changes in the storage system to maintain target consistency levels. It consists of a feedback loop which uses a technique called *spectral shifting* at each iteration to calculate the target value of artificial delay from a history of operations. The tuning framework is able to converge to the target value of artificial delay much faster than the state-of-art solution.

This thesis also presents a probabilistic analysis of inconsistencies in eventually consistent distributed storage systems operating under weak (read one, write one) consistency settings. The analysis takes into account *symmetrical* (same for reads and writes) artificial delays which enable consistency-latency tuning. A mathematical formula for the percentage of inconsistent operations is derived from other environmental parameters pertaining to the storage system. The formula's predictions for the proportion of inconsistent operations match observations of the same from a stochastic simulator of the storage system running 10^6 operations (per experiment), and from a widely used key-value store (Apache Cassandra) closely.

Acknowledgements

I would like to thank my guide, Dr. Wojciech Golab for his keen insight and constant motivation that made this thesis possible. I would also like to thank Dr. Lukasz Golab and Dr. Ladan Tahvildari for reading my thesis and enhancing it with their inputs. Some parts of chapter 2, and the entire of chapter 3 was written jointly by me and Dr. Golab for a paper at PODC 2017[13]. Lastly, I would like to thank the anonymous referees from PODC 2017 and SSS 2017[14] for their useful feedback.

Dedication

This thesis is dedicated to my family - my loving parents, my brother and Ananya. Their love and encouragement kept me going.

Table of Contents

List of Figures

Chapter 1

Introduction

1.1 Motivation

Distributed storage systems are used widely to support essential online services like web search, e-mail, social networking and e-commerce. To protect against failures resulting in permanent data loss, data is replicated across multiple storage servers. Usually, such storage servers are located in different geographical locations to prevent the possibility of all of them failing at once in face of a massive disaster like a natural catastrophe. Such replicated systems are, however, constrained by the finite propagation speed of information. This means that 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. It is difficult for a business to measure the effects of clients observing stale data. The effects of high latencies in completion of client requests is relatively easier to measure. Informal online sources report that a 500ms latency leads to a 20% drop in traffic at Google, and a 100ms latency increase costs Amazon 1% in revenues.

The need for highly available storage systems that have the ability to sacrifice consistency for lower latencies is met by *eventually consistent* systems which satisfy the property that in the absence of updates and failures, all replicas of a given key eventually converge to the same value. Examples of these systems include Dynamo-like quorum-based storage systems (e.g., Cassandra [30]), and they are in widespread use by small startups and are gaining attention in large enterprises. In this thesis, I conduct a study of such systems, configured with latency-optimized settings that do not guarantee strong consistency, and which can be formalized using Lamport's correctness properties for read/write registers

[32]. Recent experimental analysis has shown that consistency actually observed by clients is quite workload-dependent [26], and can be tuned against latency using techniques that can be layered easily on top of existing storage systems [36]. In general, experimental evidence points to the conclusion that a storage system configured for weak consistency (and low latency) may satisfy strong consistency most of the time in a practical setting, and that consistency anomalies are observed infrequently. However, the cost of eliminating all consistency anomalies can be rather high, latency-wise.

1.2 Problem statement

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 [18] and its many open-source descendants. 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., [45]) 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.

Probabilistic models of eventually consistent storage systems have been inadequately explored in literature. Bailis et al. study expected bounds on staleness for Dynamo-style systems operating under partial quorums [8]. However, as stated before, partial quorums provide discrete levels of consistency-latency tuning, especially in geo-replicated systems. Such systems typically span over multiple continents and have one-way nework delays between servers in excess of 50 ms. It is in geo-replicated systems that the problem of consistency-latency tuning is of special importance - due to an increased probability of users observing stale reads under weak consistency settings, compared to single-location clusters. Quantifying the consistency-latency trade-off and defining the currency of exchange between consistency and latency in the form of a mathematical equation is still an open research problem. The problem has been approached experimentally in the form of a feedback control loop in [40], but the tuning mechanism used is slow and unsophisticated.

1.3 Research contributions

In this thesis, I quantify the consistency latency trade-off using a combination of experimental measurement and mathematical analysis. The technical contribution is two-fold:

1. Responding to a real world need for flexible performance tuning in distributed storage systems, a technique for automated control over a probabilistic consistency-latency trade-off is proposed. The proposed technique builds on top of a similar idea introduced by Golab and Wylie [27]. The framework can be layered on top of any key-value storage system that provides read and write operations, and supports eventual consistency. 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. A novel technique called *spectral shifting* is introduced 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. A software tool called WatCA [19] (co-developed by me, Dr. Golab and Hua Fan) that analyzes operation histories is used to implement the framework, SPECSHIFT [14]. Microbenchmark experiments using a widely used key-value store (Apache Cassandra) show that the framework achieves superior convergence as compared to a state-of-the-art solution [40].

2. A probabilistic analysis of consistency in an abstract model of Amazon's Dynamo operating under weak client-side consistency settings (read-one and write-one) is performed. Reads and writes are modeled in these systems with equal (symmetric) amounts of injected artificial delays. Specifically, a mathematical expression is derived for the probability that a given value read from or written to the storage system participates in a consistency anomaly, which captures precisely the relationship between the workload (characterized by the arrival rates of read and write operations), the environment (characterized by the network latency), and consistency. The analysis extends a similar one performed by Dr. Golab [13] by capturing the effect of delays in operations on the observed consistency. The probability of observing an anomaly calculated from my model is compared to the proportion of inconsistent operations observed in a) a stochastic simulation of my storage system, and b) a widely used key-value store (Apache Cassandra). The numbers from the mathematical model, the stochastic simulation and from Cassandra trace each other closely.

1.4 Document organization

The rest of the document is organized as follows. Chapter 2 reviews related literature in the field. Chapter 3 introduces the model used in the rest of the work formally and defines terminology which is used in the later chapters. Chapter 4 describes SPECSHIFT, a novel adaptive tuning framework to tune consistency-latency rapidly. Chapter 5 describes a probabilistic model to connect environment parameters of a distributed storage system to the expected staleness, where each storage operation is injected with a constant artificial delay. Chapter 6 summarizes the thesis and the learnings from the conducted research, and points to potential future work.

Chapter 2

Background and Related Work

2.1 Trade-offs in distributed storage systems

Recent research on consistency in distributed storage systems has addressed the classification of consistency models and consistency measurement. There has been research to design storage systems that provide precise consistency guarantees. This work is influenced greatly 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) [10]. However, this work focuses on the trade-off between consistency and latency in the absence of network partitions. This trade-off often has a more direct influence on several well-known distributed storage systems compared to the one stated in the CAP principle. A better representation of such trade-offs involved in distributed storage systems compared to CAP has been proposed in the form of PACELC [1]. In the presence of network partitions (P), the trade-off a storage system has to make is between availability (A) and consistency (C). Else (E), the trade-off is between latency (L) and consistency (C).

Distributed storage systems have different designs in place to achieve the above mentioned trade-offs. Amazon's highly available key-value store, Dynamo [18] uses a quorum-based replication scheme [7, 22]. A read collects a quorum of r votes and a write collects a quorum of w votes. To ensure serial consistency, there has to be a non-null intersection in the two sets of votes which is guaranteed if $r + w$ is greater than the total number of votes. Dynamo and its derivatives (e.g., Cassandra [30], Voldemort and Riak) provide client-side consistency settings which determine the values of r and w. This enables tuning of consistency to create either CP (i.e., strongly consistent but sacrificing availability) or AP (i.e.,

highly available but eventually consistent) systems. Other designs lack such tuning knobs and instead guarantee various forms of strong consistency, like Yahoo's PNUTS, Google's Bigtable etc. [12, 15, 17, 34]. A few systems allow users to declare requirements with respect to consistency, and adjust parameters internally to fulfill these requirements when possible [6, 29, 45, 50].

2.2 Measuring consistency

Measuring consistency precisely is difficult because consistency anomalies arise from the interplay between multiple storage operations. In the case of data-stores on the cloud, a customer may not have access to logs from the storage system and often sees the storage system as a black-box. Under these circumstances, some experimental studies measured the convergence time of the replication protocol by measuring how long it takes from issuing an update to still being able to read the old version. This is easier to quantify, rather than consistency actually observed by client applications (e.g., [9, 49]). 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 [5, 51].

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 [24], which makes it possible to capture in a natural way the consistency actually observed by client applications. The algorithms defined by Golab et al. can be used for online monitoring of a live storage system or for offline analysis from operation histories. Aiyer et al. propose the notion of version-based staleness where inconsistency is measured in terms of the number of intervening writes [2]. A history of operations is defined to be *k-atomic* iff there exists a valid total order of the operations such that every read obtains the value of one of the k latest writes before it in the total order. A k-quorum protocol is proposed and it is proved to achieve k-atomic semantics.

Lamport defines three consistency semantics for communicating processes, namely safety, regularity and atomicity. Safety is the weakest of the three providing arbitrary synchronization, followed by regularity and atomicity as stronger forms of synchronization [32]. The consistency metric used in this paper is an adaptation of the technique described in [24] whereby consistency is defined relative to Lamport's regularity property. The generalization of regularity to multiple writers used in this paper resembles closely the "MWRegWO" property introduced by Shao et al. in [42]. This work studies the re-

lationships between different definitions of multi-writer regularity, which are motivated by differences in quorum-based algorithms for implementing them.

2.3 Adaptive consistency-latency tuning

In this work, we achieve consistency-latency tuning by injecting artificial delays into every read and write operation, which has an effect of improving the overall observed consistency at the cost of increased operation latencies. 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 [27]. This framework specifies concrete consistency metrics (based on [24]) 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 [40]. In a single data-center setup, the multiplicative loop is used for tuning read delays which serve as the primary control knob for tuning. Other tuning knobs like read repair rate and discrete client-side consistency settings (e.g one, quorum and all) are also used in the adaptive control loop. Read repair works by updating the values of stale replicas in a read-quorum, which improves consistency while sacrificing latency. A parameter controlling the proportion of reads that are repaired serves as a tuning knob for consistency-latency tuning. However, both read repair rate and discrete client-side consistency settings are less effective for fine-tuning consistency compared to artificial delays. In a geo-replicated environment, their feedback control uses a PID-based approach to avoid increased oscillations of the values of latency and consistency. The consistency metric used by them ignores write latency and assumes that writes take effect in the order of invocation, hence lacks a precise connection to Lamport's formalism [32]. 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 [38].

Another technique to achieve consistency-latency called Continuous Partial Quorums (CPQs) has been suggested in [36]. This technique involves assigning consistency levels on a per-operation basis. Every read and write chooses between a strong and a weak consistency setting with a tunable probability parameter. It is shown that in a single data-center setup, using the CPQ tuning technique gives a better consistency-latency trade-off compared to artificial delays. However, for a geo-replicated setup, the technique of

injecting artificial delays for tuning results in slightly lower and more predictable values of latency compared to latencies achieved using CPQ for similar values of consistency. The consistency-latency trade-off in key-value stores is amplified in geo-replicated environments with one-way network latencies in excess of 50 ms between data centers. We explore these environments in this work and so we use artificial delays for consistency-latency tuning.

The experiments for the automated adaptive tuning framework presented in this work were carried out using a geo-replicated cluster of servers on Amazon EC2 running Apache Cassandra. A software tool called the Waterloo Consistency Analyzer (WatCA) [19] was used to run these experiments and analyze operation histories. WatCA was developed with online and offline analysis of distributed key-value stores (like Apache Cassandra and Riak) in mind. It contains an implementation of a consistency metric based on Lamport's regularity property (see definition 4). This metric enables developers to detect the presence of and measure the magnitude of consistency anomalies from histories of operations. These histories can be obtained offline, or as a stream from running storage servers. It is also equipped with a number of client-side settings that help define the environment of the storage servers and include parameters like client-side consistency settings, peak throughput of servers, distribution of workloads, read and write percentage, values of read and write artificial delays, etc. The WatCA source is publicly accessible on Github.

The adaptive tuning framework presented in this thesis is described in a paper which is currently accepted for publication in the 19th International Symposium on Stabilization, Safety, and Security of Distributed Systems (SSS 2017) [14].

2.4 Mathematical models of consistency

Mathematical models of consistency are generally rooted in the notion of probabilistic quorums [33, 35]. 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. The performance of a system implemented with such probabilistic quorums can be strengthened using a well-defined *diffusion mechanism* or a *gossip protocol*, as is used by Dynamo-style storage systems. When updates are rare, the probability of inconsistency expected to be observed in such systems approaches zero. 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. Reasoning about probabilistic

quorums, the model predicts *t-visibility*, which is the probability of reading a stale value t time units after a single write operation is applied. It also predicts *k-staleness* which is the probability of reading a value which is k versions stale from the latest committed version when the read starts. A combination of these two measures of staleness, $< k, t >$*-staleness* is predicted similarly and models the probability that a read returns a value that is k versions stale t time units after the corresponding write operation is applied. Though the predictions can be made simply when expanding partial quorums due to *anti-entropy* are not taken into account, a closed-form solution when considering the same is not achieved. A model WARS is proposed to model this behavior which reasons about the interplay between the events of a write (W), its acknowledgment (A), a read (R) to the same key and the corresponding response (S). Predictions for t-visibility and k-staleness are made in the WARS model using Monte Carlo simulations. It is observed that eventually consistent systems often exhibit fairly consistent behavior (t-visibility in tens of milliseconds) due to the resilience of Dynamo-style protocols.

In this body of work, we reason about consistency observed in a Dynamo-style storage system and make predictions about the percentage of inconsistent operations expected to be observed in a cluster of distributed key-value storage systems based on probabilistic calculations, similar to PBS. However, we investigate the technique of injecting artificial delays to individual operations under a weak client-side consistency setting (read-one write-one) to control consistency, instead of partial quorums. This technique enables a finer tuning of the consistency latency trade-off compared to discrete client-side consistency settings which are available in Dynamo-style storage systems [36]. We adapt a purely mathematical approach to achieve the predictions instead of simulations, which provides insight into the mechanism of interplay between read and write events at different replicas giving rise to consistency anomalies. PBS does not distinguish between local and remote storage operations. In contrast, our model predicts the probability of observing a consistency anomaly in the broader context of a stochastic workload.

A brief announcement based on my joint work with Dr. Golab [13] describes a simple probabilistic model of inconsistency in distributed storage systems under weak consistency settings. The model described there is largely similar to the model presented in this thesis, except that it does not account for injected artificial delays. The predictions for percentage of values participating in anomalies obtained with the simple model, compared with a custom-built simulator of the storage system, were accurate to 0.001. The percentage of inconsistent values obtained from the model also closely represented the actual percentage of inconsistent operations observed in a real-world storage system (Apache Cassandra), when the overall throughput was low. However, for high throughputs, the processing delays of operations become non-negligible and the predictions for percentages of values

participating in anomalies were pessimistic when compared to a similar percentage observed from the Cassandra cluster.

2.5 Summary

The literature around eventually consistent systems is dense and a number of approaches have been suggested to quantify and measure consistency in such systems. Though the trade-off between availability and consistency in the presence of partitions, as stated by the famous CAP theorem [10], is widely accepted - the trade-off between consistency and latency in the absence of partition is often overlooked [1]. Experimental approaches to consistency-latency tuning have been explored in [40], but the prediction model used is rudimentary and tuning is slow. Mathematical models of inconsistency in partial quorums have been explored in [8]. However, partial quorums are often inadequate for fine-tuning consistency [36] in geo-replicated distributed storage systems. Probabilistic models around the techniques to fine-tune consistency-latency are still unexplored.

Chapter 3

Formal Model and Important Definitions

3.1 System model

I model a distributed storage system abstractly as a collection of processes 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 the focus in this thesis is on the behavior of the system in failure-free executions where processing and network delays are bounded. It is assumed that clocks are tightly synchronized, which allows the protocol to use timestamps for concurrency control. Any process may read or write any key, and stores the values of a subset of the keys. Since I assume a failure-free model, the purpose of replication is to enable consistency-latency tuning rather than to protect the system against permanent data loss.

A *history* of operations executed by a distributed storage system is a sequence of *steps*, representing the invocations and responses of reads and writes. (as in [28]). 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 I assume that steps are tagged with sufficient information so that all

matching pairs can be identified. I assume that every history H is *well-formed* meaning that it satisfies two properties:

1. 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

2. 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 $\texttt{WriteOp}(k, v)$, and a read of value v from key k is denoted by $\texttt{ReadOp}(k, v)$. The invocation and response times of an operation are denoted by the functions `start` and `finish`. Given two operations op_1 and op_2, we say that op_1 *happens before* op_2 in a history H if $\texttt{finish}(op_1) < \texttt{start}(op_2)$, 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 read returns the value assigned by the most recent write (preceding the read) to the same key, or the special value $\perp$ if there is no such write [28]. A history H is *regular* if it satisfies the requirements of linearizability with one exception: a read may (but is not required to) return the value assigned by any write with which the read is concurrent in H [32].

I am interested in quantifying how far a history deviates from a standard correctness properties for read/write registers, such as linearizability and regularity. I choose regularity in particular because it is the strongest property supported (in some configurations) by popular quorum-replicated storage systems, such as Dynamo [18] and its derivatives. Specifically, I use the methodology of Golab, Li and Shah [25] 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 read invocation event and increasing the time of every write 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 I 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 read is concurrent with a write 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.*

Following [26], it can be noted that the t-value for a history can be computed in polynomial time under the following assumption:

Assumption 1. *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 3. *For any history H that satisfies Assumption 1 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 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 [21]. Anomalies can therefore be quantified as follows with reference to one key and two values:

Definition 4. *For any history H, key k, and values distinct 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 the projection of H onto operations applied to key k that access value v or v'. Furthermore, $\chi(H, k, v)$ is defined as $\max_{v' \neq v} \chi(H, k, v, v')$.*

The scoring function described above is used to calculate *scores* per key and per unique value assigned to that key in a history. A distribution of these scores from a history of operations can be used by an *adaptive tuning framework* to predict the required value of delay to be injected into operations, to achieve a target value of consistency. This mechanism is explained in detail in the following chapter.

3.2 Summary

In this short chapter, I introduce the system model that is implied in the rest of the document. Operation histories and properties pertaining to them like 'well-formedness' are introduced. The concepts of a 'happens before' order between operations and of linearizability [28] are reviewed. In this thesis, however, I measure inconsistency of operations as their deviation from regularity [32] (and not linearizability), which is often the strongest property supported by quorum-replicated storage systems. The chapter also contains a detailed definition of the scoring function, which is used to calculate consistency anomaly scores per-key and per-assigned-value from an operation history.

Chapter 4

The SPECSHIFT Tuning Framework

4.1 Spectral shifting

In this chapter, I present a framework for trading off operation latency against consistency by slowing down operations using artificial delays [27, 36, 40]. Such explicit delays are similar qualitatively to the implicit delays incurred by quorum operations, in which operations wait for protocol messages from remote processes. Specifically, longer delays tend to improve consistency similarly to larger partial quorums [36]. 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.

An approach that combines probabilistic analysis with measurement to predict the frequency of consistency anomalies and achieve consistency-latency tuning, is adopted in this chapter. To that end, some relevant definitions are introduced below.

Definition 5. *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 $i \in [1, m]$ (score is defined in the previous chapter). 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 6. *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

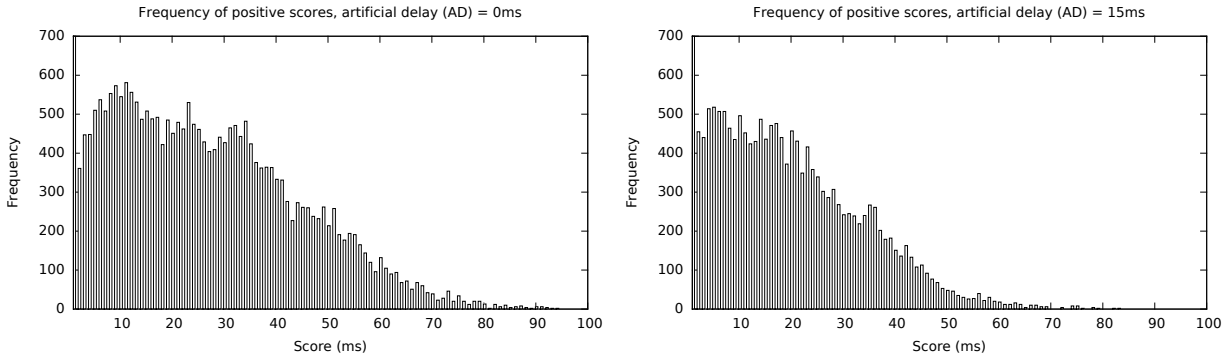
an AD of d time units to shift the spectrum of scores to the left by d bins, hence the name *spectral shifting*.

Definition 7. *For a given history H and any $i, d \in \mathbb{Z}^{\geq 0}$, the* shifted frequency of j *is defined as:*

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

The tuning framework (SPECSHIFT) described in this chapter predicts 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 $d \leq d'$ time units. d is referred to as the *base delay*, and d' as the *target delay*. 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 provided that the AD is the only parameter changed between the two executions that generate H and H'. The proportion of positive scores for H' can then be predicted using the following formula:

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



(a) AD = 0 ms (b) AD = 15 ms

Figure 4.1: Histograms showing the distribution of consistency anomaly scores for two different histories.

Figures 4.1a and 4.1b show histograms corresponding to two histories obtained with AD = 0ms and 15ms, respectively. Figure 4.1a roughly resembles the tail of figure 4.1b,

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 staleness 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 remainder of this section discusses in detail how SPECSHIFT can be used for consistency-latency tuning.

4.2 Inner-outer consistency

SPECSHIFT allows us to predict the score histogram for a *target* delay of d' from a *source* delay of d provided $d' > d$. However, we cannot use exactly the same shifting technique to predict the score histogram for a history with a higher *base* and a lower *target* delay (i.e., $d' < d$) because the *freq-s* function (see Definition 7) is undefined in this case. To overcome this limitation, we propose a 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 4.1 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:

Definition 8. *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* $[start(\texttt{ReadOp}(k, v))+d, finish(\texttt{ReadOp}(k, v))]$. $\texttt{ReadOp}(k, v)$ *is the* outer operation *in this context.*

Definition 9. *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* $[start(\texttt{WriteOp}(k, v)), finish(\texttt{WriteOp}(k, v))-d]$. $\texttt{WriteOp}(k, v)$ *is the* outer operation *in this context.*

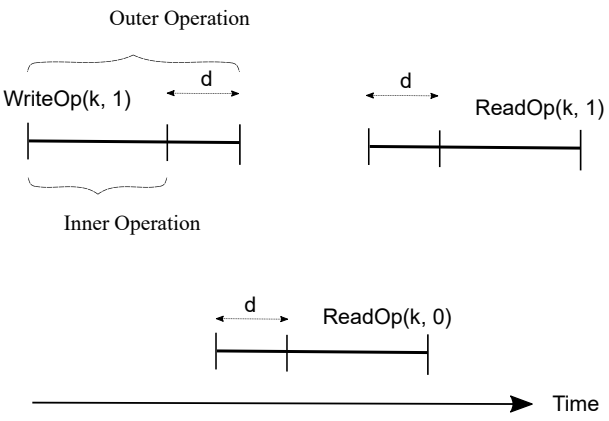


Figure 4.2: Inner-outer operations with an AD of d.

Figure 4.2 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, as $\texttt{WriteOp}(k, 1)$ is concurrent with $\texttt{ReadOp}(k, 0)$. However, the history of inner operations, corresponding to a history similar to H but with an AD of 0 instead of d, has one consistency anomaly as $\texttt{WriteOp}(k, 1)$ and $\texttt{ReadOp}(k, 0)$ are no longer concurrent.

4.3 Adaptive tuning framework

We can use *SPECSHIFT* to design 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 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 10. *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 10 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 $d_t = d_p$.

4.4 Experimental evaluation

4.4.1 Hardware and software environment

A *geo-replicated* cluster on Amazon EC2 consisting of six *m4.large* on-demand instances running *64-bit Ubuntu Server 14.0.4 LTS(HVM)* were used in the experimental setup for evaluating the SPECSHIFT adaptive control loop. Each of these instances were equipped with 2 Intel Xeon E5-2676v3 2.4 GHz cores 8 GB RAM and 8GB SSD local storage. The

six instances spanned over three availability zones: *US West (Oregon)*, *EU (Ireland)* and *Asia Pacific (Tokyo)* and were distributed uniformly at two per availability zone. Each of the two instances in an availability zone were placed on different racks. An *Apache Cassandra 2.2.7* installation was used on all the six hosts. *Yahoo! Cloud Serving Benchmark (YCSB) 0.10.0* was used for workload generation. A modified *YCSB* client running a single *YCSB* process on each host, with 128 client threads connecting to the *Cassandra* server on the same host was used to serve the workloads. We used the standard YCSB workload for all of our experiments. The *ycsb* keyspace was set up with the replication strategy *NetworkTopologyStrategy* and with a replication factor of one per availability zone. We used Cassandra's *Ec2MultiRegionSnitch* at each server. All hosts were included as seeds for *Cassandra*.

The average one-way network delay between us-west and asia pacific data centers was 45 ms, between us-west and eu was 62 ms and between eu and asia pacific was 106 ms. The average one-way network delay for the entire cluster can be estimated at 71 ms, which is the mean of the three values. Default external NTP servers were used for *clock synchronization* which provide synchronization to within 5-10 ms.

4.4.2 Experimental setup

I compare the convergence of the SPECSHIFT adaptive tuning framework, the PCAP *multiplicative control loop* [40], and a binary search for the optimal AD over the constrained range $[0, 71]$ using Apache Cassandra deployed in Amazon's Elastic Compute Cloud (EC2).

20 experiments were run 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 eu and asia-pacific, as mentioned earlier). In each experiment, three different tuning techniques are compared, namely: SPECSHIFT, the PCAP multiplicative control loop and a constrained binary search. All operations are delayed by the same starting delay for all three techniques and each of them tune the AD trying to achieve the target proportion of consistency anomalies. The number of iterations required by each technique to obtain convergence to a value of AD which achieves the target proportion can be compared, as a measure of the its efficiency. Each iteration is run for 30 seconds with a throughput of 6000 operations/s and a read proportion of 0.8. The keys are drawn from the YCSB distribution *latest*, which favors recently chosen keys.

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 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 average one-way network latency of the cluster. So, the optimal delay to achieve a non-zero target proportion must lie somewhere in between 0 and the average one-way network latency. It is important to note that this technique only works when the average one-way network latency of the cluster is known from beforehand, and the network does not change drastically. While it might be possible to use binary search otherwise, using a pessimistic upper bound on the artificial delay, the performance of the search in terms of iterations required to converge worsens with an increased range. If the network conditions change dynamically, it could be difficult to estimate an optimal upper bound. The other two tuning mechanisms require no such additional parameter for tuning.

I compared SPECSHIFT to a proportional-integral-differential (PID) controller for consistency-latency tuning. Using the PID controller involves tuning additional control parameters k_p, k_d and k_i. Convergence, if achieved, with the values of these parameters suggested in [40] ($k_p = 1$, $k_d = 0.8$, $k_i = 0.5$) is extremely slow and so the results have been omitted.

4.4.3 Obtained results

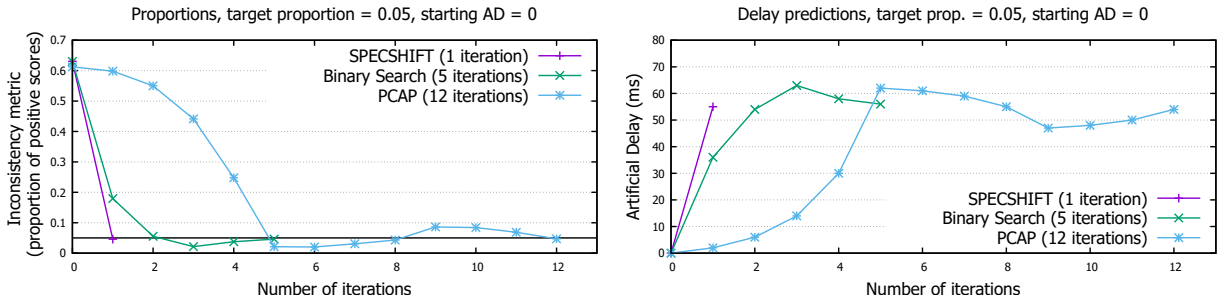


Figure 4.3: One experiment comparing three different tuning mechanisms, target proportion = 0.05, no starting AD.

Figures 4.3 and 4.4 illustrate details for two of the 20 experiments. The termination condition is defined as convergence to a proportion of positive scores within 0.005 of the target proportion (denoted by a solid horizontal line in the plots showing the proportions

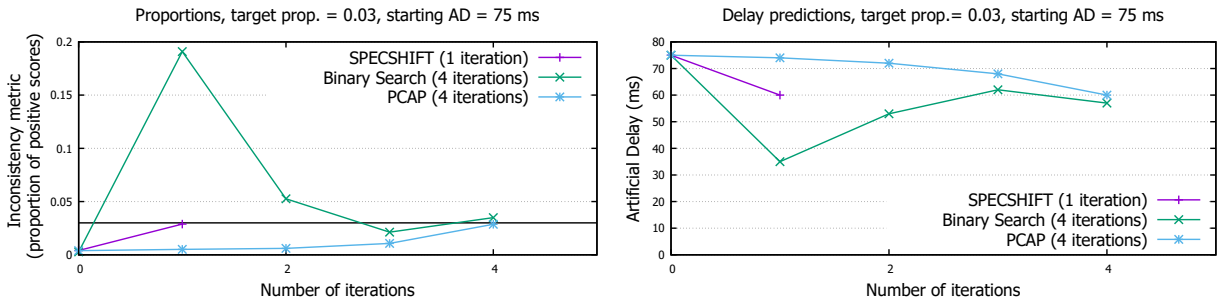


Figure 4.4: Another experiment comparing three different tuning mechanisms, target proportion = 0.03, starting AD = 75 ms.

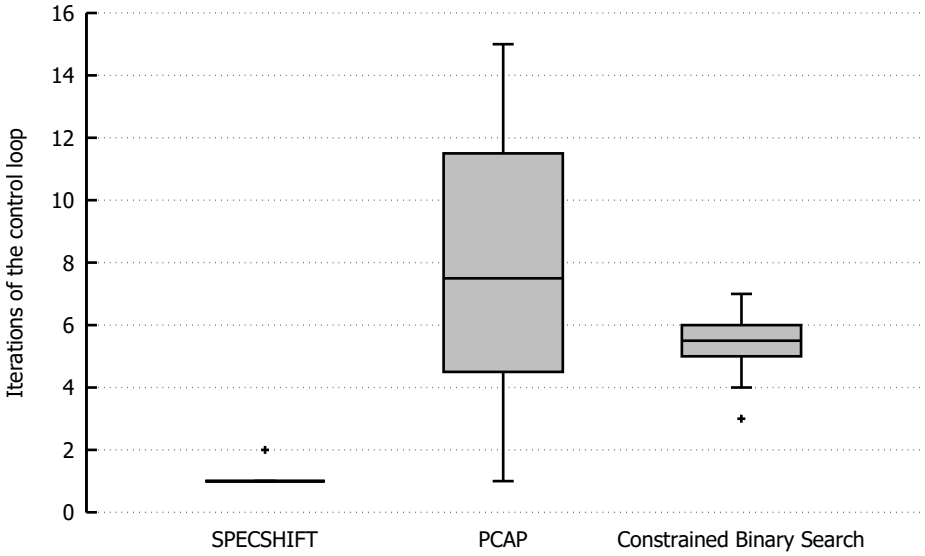


Figure 4.5: Boxplot showing range of iterations taken to converge by different tuning mechanisms over 20 experiments.

obtained at each iteration). Figure 4.3 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 4.4 shows the same for a starting delay of 75 ms and a target proportion of 0.03. We observe that the PCAP multiplicative control loop is prone to oscillations and requires more than ten iterations to converge in the first case (figure 4.3), though it reaches very close to the target value at the eighth iteration. The techniques other than SPECSHIFT converge faster in the second experiment (figure 4.3), partly because of the fact that the source and target delays are less far apart. Though SPECSHIFT requires only one iteration in both the cases to correctly converge to the target proportion with the required precision, the number of iterations required by the other two techniques depend on the specific values of the starting delay and the target proportion. It is interesting to note that in one of the two experiments, SPECSHIFT uses the *outer* history of operations and in the other it uses the *inner* history of operations to predict the value of the AD for the next iteration.

Figure 4.5 shows that the PCAP multiplicative loop takes anywhere between 1 to 15 iterations to converge to the target proportion of positive scores in our experiments, with the mean value at a little more than 7. Binary search in the constrained interval of [0, 71] takes anywhere between 4 to 7 iterations to do the same, with a mean of almost 6. SPECSHIFT however almost consistently takes one iteration to converge. The plot 4.5 shows outliers for SPECSHIFT and binary search. Outliers are determined as any value that lies 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.

The accuracy of the prediction at each iteration of SPECSHIFT is contingent on the workload not changing across iterations. In the experiments described so far in this section, I have assumed an open system [41] 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 4.6 demonstrates an experiment which compares 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 starting delay is 0 and the target proportion of positive scores is 0.05, similar to the experiment in figure 4.3. We find that the overall throughput of the system drops by half (from 22 kops/s to 11 kops/s) between the starting point of each adaptive loop in figure 4.6 (AD = 0 for all operations) and their point of convergence (AD roughly equal to 46ms for all operations).Though SPECSHIFT takes one extra iteration (2 iterations total) to converge in this case compared to the previous experiments, it performs better than the PCAP multiplicative loop (9 iterations total) and constrained binary search (5 iterations total).

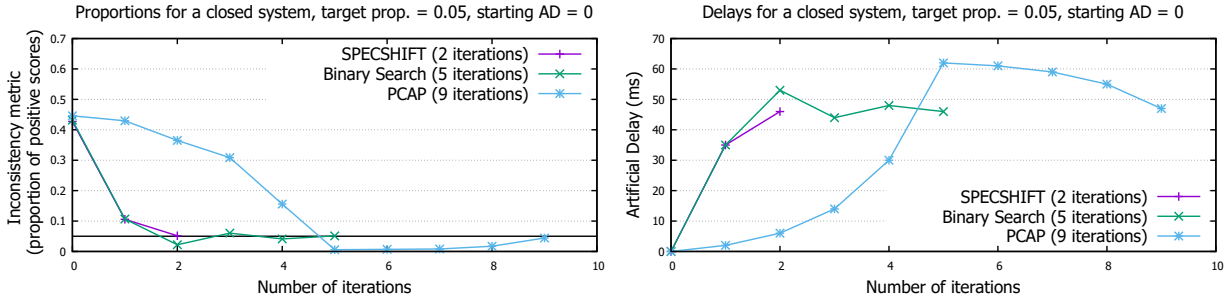


Figure 4.6: Comparison of the techniques for a closed system, target proportion = 0.05, starting AD = 0

4.4.4 Discussion

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 4.3 and 4.4 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.

4.5 Summary

In this chapter, a technique called *spectral shifting* is described, which suggests that if we shift the score histogram of a history of operations to the left by d bins, we get a score histogram for another history where each operation is delayed by d time units. I use the

phenomenon of spectral shifting to design an adaptive tuning framework (SPECSHIFT) that takes a target proportion of staleness as input, and predicts the value of AD to be injected to each operation to meet the target proportion of staleness. SPECSHIFT is compared to the PCAP multiplicative loop [40] and a feedback loop that uses a constrained binary search for predicting delays at each iteration. Experiments show that SPECSHIFT is able to achieve much faster convergence to the optimal value of AD, compared to the other tuning frameworks.

Chapter 5

A Probabilistic Analysis of Eventual Consistency

5.1 Notation and general assumptions

In this chapter, I perform a probabilistic analysis of eventual consistency on a theoretical distributed storage system which is similar to Amazon's Dynamo. Reads and writes are processed on every storage server using read/write quorums. For this analysis, we focus on the weak consistency setting in which the size of read and write quorums is one. To achieve desired levels of consistency, all read and write operations are injected with a tunable artificial delay. We assume that the clocks on all the storage servers are synchronized. Following our analysis, we are able to formulate a mathematical equation to determine the probability with which a given write participates in a consistency anomaly. The consistency metric used in the analysis is the same used for SPECSHIFT (see definition 4). We use the following notations in the analysis:

1. There are n storage servers.
2. The start and finish times of an operation (read or write) op is denoted by $start(op)$ and $finish(op)$ respectively.
3. There is a constant one-way network delay of L across n replicas.
4. A write is followed by a delay of d time units and a read is preceded by a delay of d time units.

5. The total throughput across all replicas is represented by λ.

6. λ_w represents the write throughput and λ_r represents the read throughput. The ratio λ_r/λ is the read-proportion and is represented by ρ. λ, λ_r, and λ_w follow the relationship: $\lambda = \lambda_r + \lambda_w$.

Following are some assumptions made in the the analysis:

1. Reads and writes follow a Poisson process and consequently, operations have exponential inter-arrival times. Such an assumption is meaningful when the arrivals of operations are memoryless and independent of each other.

2. All operations are on the same key.

3. Full replication is performed across the storage servers.

4. $L > 3d$, for simplicity of analysis.

5. Processing delays for reads and writes on the servers are considered to be zero, for simplicity of analysis. A write takes effect in a remote replica L time units after it starts in the replica in which it first arrived.

Under these assumptions, we can define the events that result in a write of value v, W_v at replica r_i participating in an anomaly with the write of another value x at any replica, W_x and their corresponding reads. The read of a value v is denoted by R_v and similarly, the read of a value x is denoted by R_x. Since all operations are on the same key, it has been omitted from all symbols in the analysis for the sake of clarity and simplicity. The 4 disjoint cases in which a subset of operations from $\{W_v, R_v, W_x, R_x\}$ can conspire to cause a consistency anomaly are enlisted below:

- V1 : There exists a write of value x, W_x that starts after the finish of write W_v and a read of the value v that starts after the finish of write W_x.

- V2 : There exists a write W_x that finishes before the start of write W_v and a read of the value x that starts after the finish of write W_v.

- V3 : There exists a write W_w that starts after the start of W_v but before the finish of W_v, a read of the value v that starts after the finish of W_x and a read of the value x that also starts after the finish of W_x.

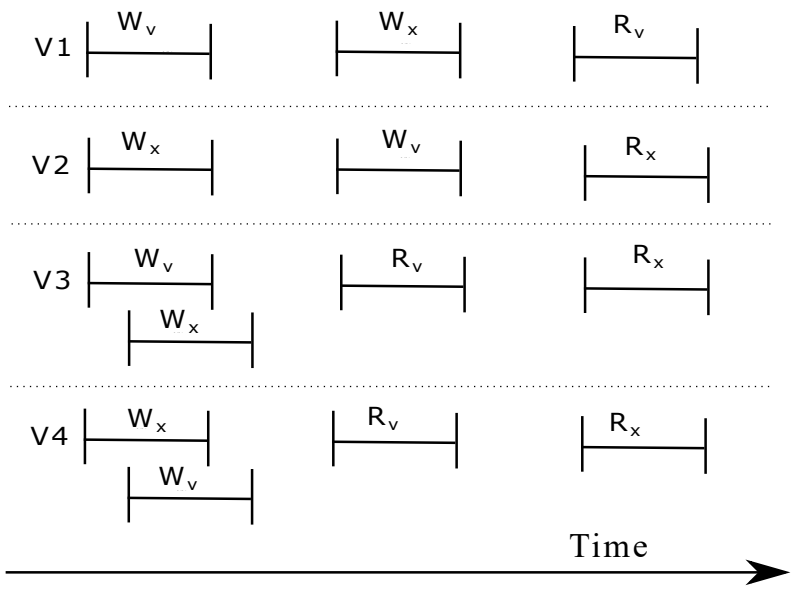


Figure 5.1: Anomaly cases between the writes of two values, v and x, and their reads.

- V4 : There exists a write W_x that finishes after the start of W_v but before the finish of W_v, a read of the value v that starts after the finish of W_v and a read of the value x that also starts after the finish of W_v.

Examples of the anomaly cases V1-V4 described above are illustrated in figure 5.1. The start and finish times of operations W_v, W_x, R_v and R_x are indicated by vertical lines and time is represented on the horizontal axis progressing onward from left to right. The figure shows one example of an anomaly for each case and there are other ways in which these cases can occur. For example, in the cases V3 and V4, the position of the reads R_v and R_x can vary arbitrarily as long as both of them start after the finish of both the writes W_v and W_x.

5.2 A simpler probabilistic model

This analysis builds on top of a similar analysis by Dr. Golab [13] which also assumes that operations take place instantaneously, but does not take the artificial delay d into account. The probabilistic analysis is simpler in such a case, and an elegant closed-form expression for the proportion of zero scores (see definition 4) is achieved, as shown below. The proportion of positive scores can be calculated by subtracting this probability from 1.

Theorem 1. *For any infinite history H of the system, and for any value $v \in \mathbb{Z}^{>0}$ written to the key k in H:*

$$
\begin{aligned}
P(\chi(H,k,v)=0) &= \left[\left(\frac{\lambda_w}{\lambda_r+\lambda_w}\right)+\left(\frac{\lambda_r}{\lambda_r+\lambda_w}\right)e^{-(\lambda_r/n+\lambda_w/n)L}\right]^{n-1} \\
&\times \left[\left(\frac{\lambda_w}{\lambda_w+\lambda_r\left(\frac{n-1}{n}\right)}\right)+\left(\frac{\lambda_r\left(\frac{n-1}{n}\right)}{\lambda_w+\lambda_r\left(\frac{n-1}{n}\right)}\right)e^{-\left(\lambda_w+\lambda_r\left(\frac{n-1}{n}\right)\right)L}\right]
\end{aligned}
$$

The simple model produces accurate predictions of staleness when compared to a real-world distributed storage system (Apache Cassandra) for low values of overall throughput. However, for high values of throughput, the predictions are pessimistic - because the simple model does not account for processing delays on the storage servers. In this chapter, I extend the simple model to account for constant and symmetric artificial delays. Artificial delays serve to mask the effect of processing delays, thus correcting the weakness of the simple model to an extent. They also serve as a consistency-latency tuning mechanism. The rest of this chapter explains the intuition and derivation of the extended mathematical model (including artificial delays) in detail.

5.3 Intuition and simplifying assumptions

We can see from figure 5.1 that three anomaly cases out of four (V1, V3 and V4) involve a read of the value v. We use this observation to break our analysis up into six cases depending on the position of the latest read of v in a given history. The latest read of v is important because if a set of operations containing a read of v causes one of anomaly case V1-V4, it would still cause the same anomaly case if the read of v was substituted with the latest read of v. This property allows us to reason about the latest read of v in the history and ignore reads of v prior to that. Broadly, we break each case up into three parts:

1. Calculating P(E1) where E1 is the event of the latest read of v finishing in a given interval of time.
2. Given E1 occurs, calculating P(E2), where E2 is the event of no write starting in time [0, L] from start(W_v) or read finishing in the same interval and causing one or more of cases V1-V4.
3. Given E1 and E2 occur, calculating P(E3) where E3 is the event of no write starting in time [L, ∞] from start(W_v) or read finishing in the same interval and causing one or more of cases V1-V4.

The anomaly cases V1-V4 arise out of interplay between multiple operations which may overlap in time. To limit the complexity of the analysis, a few simplifying assumptions have been made in each case. Results from the analysis with the simplifying assumptions do not consistently result in either an upper or a lower bound on the actual probability. The nature of these assumptions are largely similar across cases and are stated below, though each assumption is re-stated with the corresponding specifics in each case as well.

1. P(E2) is calculated in most cases by calculating the probability of no operation causing an anomaly in each replica individually and multiplying the resulting probability by the number of replicas. We can do this because operations starting in the time interval [0, L] from start(W_v) in one replica do not affect the probabilities of operations starting in the same interval in other replicas. Avoiding each of the anomaly cases V1-V4 impose slightly different restrictions on the intervals in which reads and writes can feature in each replica. However, in some specific cases, they have been treated similarly for the sake of simplicity.

2. P(E3) is calculated in some cases ignoring the effects of E1 and E2 on the probability of the respective reads and writes in this interval. However, the calculation of E3 does take into account that the latest read of v starts at a position specific to that case. If the throughput is high or if L is large, P(E3) tends to be close to 1 and relatively independent of E1 and E2.

5.4 Detailed analysis

It is assumed that W_v starts at time 0 at replica r_i, and finishes at time d. The analysis is divided into 6 disjoint cases, based on the position of the latest read of v (called R_{vl} henceforth). These cases are:

1. Case A : R_{vl} finishes in the time interval $[2d, 3d)$.

2. Case B : R_{vl} finishes in the time interval $[3d, \min(4d, L - 3d))$.

3. Case C : If $L > 4d$, R_{vl} finishes in the time interval $[4d, L - 4d))$.

4. Case D : R_{vl} finishes in the time interval $[L,L + d)$.

5. Case E : R_{vl} finishes in the time interval $[L + d,\infty)$

6. Case F : R_{vl} finishes in the time interval (d, $2d$) or no read of value v occurs at all.

In the rest of this section, I calculate the probability of none of V1-V4 happening in each case independently. Since the six cases are disjoint, we can add the individual probabilities from each case to get the probability of none of V1-V4 occuring in all the six cases. This probability is equal to the probability that W_v does not participate in a consistency anomaly.

5.4.1 Case A

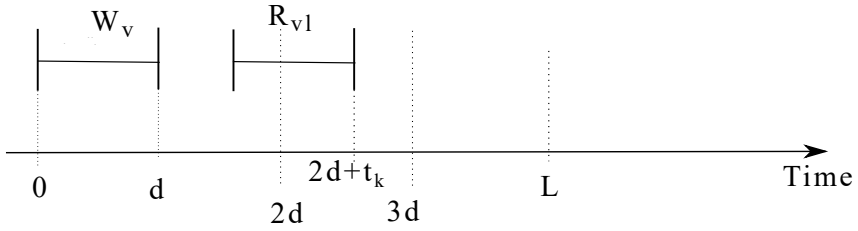


Figure 5.2: Case A.

R_{vl} finishes in the time interval $[2d, 3d)$: the probability of no anomaly happening in this case is the probability of R_{vl} finishing in the interval $[2d, 3d)$ and the probability of non-occurrence of any operation other than R_{vl} in any of the replicas that results in one or more of V1, V2, V3 and V4. In this case, anomalies happen due to the occurrence of one or more of V2, V3 and V4. V1 does not occur because $start(R_{vl}) - finish(W_v) < d$, which means that any write starting after $finish(W_v)$ will be concurrent with R_{vl}.

Let us assume that R_{vl} finishes at time $2d + t_k$ from time 0 (which corresponds to $start(W_v)$) , $0 \leq t_k < d$. Such a read can only happen in replica r_i because all replicas other than r_i can only read the value v L time units after time 0 and $2d+t_k < L$. Probability that a read starts in replica r_i at time t_k from $finish(W_v)$ is $\frac{\lambda_r}{n} e^{-(\lambda_r/n)t_k}$. The probability that there are no writes in replica r_i from time 0 to $finish(R_{vl})$ is $e^{-(\lambda_w/n)(t_k+2d)}$.

The probability that there is at least one read of the value v at r_i finishing in the time interval $(t_k + 2d, L]$ at time $t_k + 2d + t_l$, $0 \leq t_l \leq (L - t_k - 2d)$, is the probability of at least one read finishing in the above mentioned interval and the probability of no other write starting at r_i in the time interval $(0, t_k + 2d + t_l]$. Given there are no writes at r_i from time 0 to $finish(R_{vl})$, this probability can be calculated as below:

$$\int_{t_l=0}^{t_l=L-(t_k+2d)} (\lambda_r/n) e^{-(\lambda_r/n)t_l} e^{-(\lambda_w/n)(t_l)} dt_l$$

$$= \left[\left(\frac{-\lambda_r/n}{\lambda_r/n + \lambda_w/n} \right) e^{-(\lambda_r/n+\lambda_w/n)t_l} \Big|_{t_l=0}^{L-t_k-2d} \right] = \left(\frac{\lambda_r}{\lambda} \right) \left(1 - e^{-(\lambda/n)(L-t_k-2d)}\right)$$

The probability of no read of v finishing at r_i in the same time interval is:

$$1 - \left(\frac{\lambda_r}{\lambda} \right) \left(1 - e^{-(\lambda/n)(L-t_k-2d)}\right) = \left(\frac{\lambda_w}{\lambda} \right) + \left(\frac{\lambda_r}{\lambda} \right) e^{-(\lambda/n)(L-t_k-2d)} \tag{5.1}$$

The probability that v is read in the time interval $[L, \infty]$ is the probability of one read finishing at time $L+t_m$ from zero at any replica r_j, and the probability of no write starting in the interval $(0,\ t_m]$ in all replicas, and the probability of no write starting in the interval $(t_m,\ L + t_m]$ in the replica r_j. This probability can be calculated as below:

$$\int_{t_m=0}^{t_m=\infty} \lambda_r e^{-\lambda_r t_m} e^{-\lambda_w t_m} e^{-(\lambda_w/n)L} dt_m = \left[\left(\frac{-\lambda_r}{\lambda_r + \lambda_w} \right) e^{-(\lambda_w/n)L} e^{-(\lambda_r+\lambda_w)t_m} \Big|_{t_m=0}^{\infty} \right]$$

$$= \left(\frac{\lambda_r}{\lambda} \right) e^{-(\lambda_w/n)L} \tag{5.2}$$

The probability that no such read occurs is $1 - \left(\frac{\lambda_r}{\lambda}\right) e^{-(\lambda_w/n)L}$. For simplicity of analysis, we assume that the events - no read of v finishing in the time interval $(2d + t_k, L]$ and in time interval $(L, \infty]$ are independent. From the above calculations, the probability of R_{vl} finishing at time $2d + t_k$ from 0, $P(R_A)$, can then be represented by the equation below:

$$P(R_A) = \left(\frac{\lambda_r}{n} \right) e^{-(\lambda_r/n)t_k} e^{-(\lambda_w/n)(t_k+2d)} \left[\left(\frac{\lambda_w}{\lambda} \right) + \left(\frac{\lambda_r}{\lambda} \right) e^{-(\lambda/n)(L-t_k-2d)} \right] \times \left(1 - \left(\frac{\lambda_r}{\lambda} \right) e^{-(\lambda_w/n)L} \right) \tag{5.3}$$

In the time interval $[0,\ L]$, given R_{vl} finishes in $[2d, 3d]$, there is no write start or read finish in r_i that causes one or more of V1, V2, V3 and V4. Intuitively, this is because no write starts in r_i in the interval $(0,\ 2d + t_k]$ and any read that finishes at r_i in this interval reads v. Any read finishing in the interval $(2d + t_k,\ L]$ reads v or a value written by a write that started after $2d + t_k$, and hence is not concurrent with W_v. So, the probability of no anomaly in the interval $[0,\ L]$ at r_i is 1.

For any replica $r_z \neq r_i$, we divide the analysis up into two sub-cases:

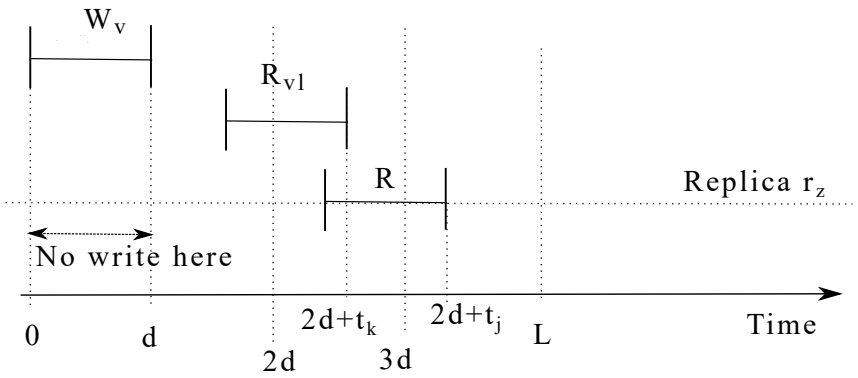


Figure 5.3: Subcase A1.

Subcase A1: No write starts in [0, d] at r_z. This happens with probability $e^{-(\lambda_w/n)d}$. V2 and V4 can happen at r_z in the interval [0, L]. Any read finishing at r_z in the interval [$2d$, L] at $2d + t_j, 0 \leq t_j \leq (L - 2d)$ reads a value written by a local or a remote write that either finishes before 0 (causing V2), or is concurrent with W_v (causing V4), if there is no write starting at r_z in [d, $2d + t_j$]. The probability of such a read finishing at time $2d + t_j$ is:

$$(\lambda_r/n)\, e^{-(\lambda_r/n)t_j} e^{-(\lambda_w/n)(t_j+d)}$$

Therefore, the probability of at least one such read is:

$$\int_{t_j=0}^{t_j=(L-2d)} (\lambda_r/n)\, e^{-(\lambda_r/n)t_j} e^{-(\lambda_w/n)(t_j+d)} dt_j = (\lambda_r/n)\, e^{-(\lambda_w/n)d} \int_{t_j=0}^{t_j=(L-2d)} e^{-(\lambda/n)t_j} dt_j$$

$$= \left(\frac{\lambda_r}{\lambda}\right) e^{-(\lambda_w/n)d} \left(1 - e^{-(\lambda/n)(L-2d)}\right)$$

The probability of no such read, also equal to the probability of no anomaly happening at r_z in the interval [0, L] for case A1, and represented by $P(X_{A1})$ is calculated as follows:

$$P(X_{A1}) = 1 - \left(\frac{\lambda_r}{\lambda}\right) e^{-(\lambda_w/n)d} \left(1 - e^{-(\lambda/n)(L-2d)}\right) \tag{5.4}$$

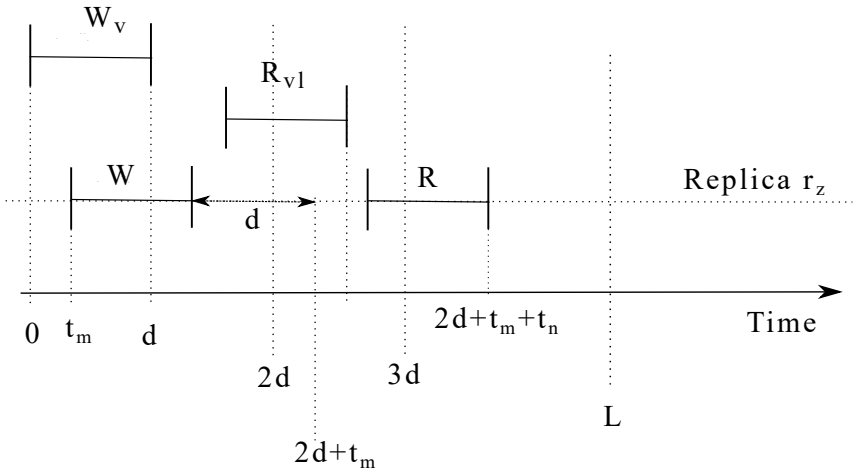


Figure 5.4: Subcase A2.

Subcase A2: There is at least one write starting in [0, d] at r_z. V3 can happen at r_z in the interval [0, L], but no start of a write or finish of a read at r_z in the interval [0, L] causes any one of V1, V2 or V4. Let the latest write concurrent with W_v start at time t_m, $0 < t_m < d$. The probability of the latest write concurrent with W_v arriving at time t_m, can be represented as below:

$$(\lambda_w/n)\, e^{-(\lambda_w/n)t_m} e^{-(\lambda_w/n)(d-t_m)} = (\lambda_w/n) e^{-(\lambda_w/n)d}$$

On further inspection, a read must start after the finish of the latest concurrent write, which happens at time $t_m + d$, to cause V3 in the interval [0, L]. Such a read finishes after $t_m + 2d$. From figure 5.4, we can reason that no finish of a read or start of a write causes V3 in the interval [0, L] if there is no read that finishes in the interval $(2d + t_m,\ L]$ at $2d + t_m + t_n, 0 < t_n \leq (L - 2d - t_m)$ without at least one write starting before it in the interval $[d,\ 2d + t_m + t_n]$. The probability of the finish of at least one read in the interval $(2d + t_m,\ L]$ causing V3 can be represented as below:

$$\int_{t_n=0}^{t_n=L-2d-t_m} (\lambda_r/n)\, e^{-(\lambda_r/n)t_n} e^{-(\lambda_w/n)(t_n+t_m+d)} dt_n$$

$$= \left(\frac{\lambda_r/n}{-\lambda/n}\right) e^{-(\lambda_w/n)(t_m+d)} \left[e^{-(\lambda/n)t_n}\Big|_{t_n=0}^{L-2d-t_m}\right] = \left(\frac{\lambda_r}{\lambda}\right) e^{-(\lambda_w/n)(t_m+d)} \left(1 - e^{-(\lambda/n)(L-2d-t_m)}\right)$$

Hence, the probability of no such read finishing in the same interval is:

$$1 - \left(\frac{\lambda_r}{\lambda}\right) e^{-(\lambda_w/n)(t_m+d)} \left(1 - e^{-(\lambda/n)(L-2d-t_m)}\right) \tag{5.5}$$

Using (5.5), the probability of no anomaly in the interval [0, L] at r_z for sub-case A2, represented by $P(X_{A2})$, can be calculated as below:

$$P(X_{A2}) = \int_{t_m=0}^{t_m=d} (\lambda_w/n)\, e^{-(\lambda_w/n)d} \left(1 - \left(\frac{\lambda_r}{\lambda}\right) e^{-(\lambda_w/n)(t_m+d)} \left(1 - e^{-(\lambda/n)(L-2d-t_m)}\right)\right) dt_m$$

$$= \left(\frac{\lambda_w}{n}\right) e^{-(\lambda_w/n)d} d - \left(\frac{\lambda_r}{\lambda}\right) e^{-2(\lambda_w/n)d}(1-e^{-(\lambda_w/n)d}) + \left(\frac{\lambda_w}{\lambda}\right) e^{-(\lambda/n)(L-2d)} e^{-(\lambda_w/n)d}(e^{(\lambda_r/n)d}-1) \tag{5.6}$$

The probabilities $P(X_{A1})$ and $P(X_{A2})$ depend only on r_z's local reads and writes. So, the probabilities for all r_z ($\neq r_i$) are independent of each other. The overall probability that there is no write start or read finish at any replica that causes an anomaly in the interval [0, L] given R_{vl} finishes at time $2d + t_k$ is:

$$(e^{-(\lambda_w/n)d} P(X_{A1}) + P(X_{A2}))^{n-1} \tag{5.7}$$

A read finishing in the interval (L, ∞) can cause V3 to happen if it reads the value written by a write that is concurrent with W_v. No write starting in this interval can cause an anomaly. We make two simplifications to minimize the complexity of the analysis here:

- We assume that the events of no write starting and no read other than R_{vl} finishing and causing one or more of V1, V2, V3 and V4 in the intervals [0, L] and [L, ∞] are independent of each other. This assumption allows us to calculate the probabilities in the intervals individually and multiply the individual probabilities to obtain the probability of not observing an anomaly involving W_v in the combined interval [0, ∞].
- We know that the presence of R_{vl} implies that there are no writes starting in r_i in the interval [0, $2d + t_k$], but we ignore this in our calculation of the probability of no anomalies caused in the interval [L, ∞].

Let us consider a write W_w, writing a value w at the replica r_w , starting in the time interval (0, d). W_w can happen in any replica in this interval except r_i, the replica where r_{vl} happens. The probability of W_w starting at time $t_p, 0 \leq t_p < d$ is $\left(\frac{n-1}{n}\right) \lambda_w e^{-\left(\frac{n-1}{n}\right)\lambda_w t_p}$.

Given W_w happens at time t_p, the probability that there is at least one read of w at any replica finishing in the interval [L, $L + t_p$] can be calculated as the probability of at

least one read finishing in the same interval and of no write starting before it in the same replica. The probability of a read finishing at $L+t_q, 0 \leq t_q \leq t_p$ in any replica r_w ($r_w \neq r_i$) and reading the value w can be represented as:

$$(\lambda_r/n)\, e^{-(\lambda_r/n)t_q} e^{-(\lambda_w/n)(L-t_p+t_q)}$$

The probability of at least one read of value w at replica r_w finishing in the interval [L, $L+t_p$] can hence be calculated as:

$$\int_{t_q=0}^{t_q=t_p} (\lambda_r/n)\, e^{-(\lambda_r/n)t_q} e^{-(\lambda_w/n)(L-t_p+t_q)} dt_q = \left(\frac{\lambda_r}{\lambda}\right) e^{-(\lambda_w/n)(L-t_p)} \left(1 - e^{-(\lambda/n)t_p}\right)$$

The probability that that there is a read of the value w at any replica finishing in the interval [$L+t_p$, ∞] is the same as that calculated in (5.2) and is equal to $\left(\frac{\lambda_r}{\lambda}\right) e^{-(\lambda_w/n)L}$. We make an independence assumption similar to the one stated above for events in the intervals (L, $L+t_p$] and ($L+t_p$, ∞). We can now calculate the probability of at least one read finishing in the interval [L, ∞] and causing V3 as follows:

$$\int_{t_p=0}^{t_p=d} \left(\frac{n-1}{n}\right) \lambda_w e^{-\left(\frac{n-1}{n}\right)\lambda_w t_p} \left[\left(\frac{\lambda_r}{\lambda}\right) e^{-(\lambda_w/n)(L-t_p)} \left(1 - e^{-(\lambda/n)t_p}\right)\right] \times \left(1 - \left(\frac{\lambda_r}{\lambda}\right) e^{-(\lambda_w/n)L}\right) dt_p$$

$$= \left(\frac{n-1}{n}\right) \lambda_w \left(1 - \left(\frac{\lambda_r}{\lambda}\right) e^{-(\lambda_w/n)L}\right) \times \int_{t_p=0}^{t_p=d} e^{-\left(\frac{n-1}{n}\right)\lambda_w t_p} \left[\left(\frac{\lambda_r}{\lambda}\right) e^{-(\lambda_w/n)(L-t_p)} \left(1 - e^{-(\lambda/n)t_p}\right)\right] dt_p$$

The integral

$$\int_{t_p=0}^{t_p=d} e^{-\left(\frac{n-1}{n}\right)\lambda_w t_p} \left(\left(\frac{\lambda_r}{\lambda}\right) e^{-(\lambda_w/n)(L-t_p)} \left(1 - e^{-(\lambda/n)t_p}\right)\right) dt_p$$

$$= \int_{t_p=0}^{t_p=d} \left(\left(\frac{\lambda_r}{\lambda}\right) e^{-(\lambda_w/n)L} e^{-\left(\frac{n-2}{n}\right)\lambda_w t_p} - \left(\frac{\lambda_r}{\lambda}\right) e^{-(\lambda_w/n)L} e^{-\left(\frac{(n-1)\lambda_w+\lambda_r}{n}\right)d}\right) dt_p$$

$$= \left(\frac{n\lambda_r}{(n-2)\,\lambda_w\lambda}\right) e^{-(\lambda_w/n)L}\left(1 - e^{-\left(\frac{n-2}{n}\right)\lambda_w d}\right) -$$
$$\left(\frac{n\lambda_r}{((n-1)\,\lambda_w + \lambda_r)\,\lambda}\right) e^{-(\lambda_w/n)L}\left(1 - e^{-\left(\frac{(n-1)\lambda_w+\lambda_r}{n}\right)d}\right)$$

So, the probability of no read in any replica finishing in the interval $(L, \infty]$ causing V3, represented by P(I), can be calculated as follows:

$$P(I) = 1 - \left(\frac{n-1}{n}\right)\lambda_w\left(1 - \left(\frac{\lambda_r}{\lambda}\right)e^{-(\lambda_w/n)L}\right) \times$$
$$\left[\left(\frac{n\lambda_r}{(n-2)\,\lambda_w\lambda}\right) e^{-(\lambda_w/n)L}\left(1 - e^{-\left(\frac{n-2}{n}\right)\lambda_w d}\right) - \right. \tag{5.8}$$
$$\left.\left(\frac{n\lambda_r}{((n-1)\,\lambda_w + \lambda_r)\,\lambda}\right) e^{-(\lambda_w/n)L}\left(1 - e^{-\left(\frac{(n-1)\lambda_w+\lambda_r}{n}\right)d}\right)\right]$$

The overall probability of no anomaly for case A (represented by P(A)), which is the probability that there is no write start or read finish at any replica in the interval $[0, \infty]$ that causes an anomaly, given R_{vl} finishes in the time interval $[2d, 3d]$, can be calculated from (5.3), (5.7) and (5.8) as follows:

$$P(A) = \int_{t_k=0}^{t_k=d} \left(P(R_A) \times (e^{-(\lambda_w/n)d}P(X_{A1}) + P(X_{A2}))^{n-1} \times P(I)\right) dt_k$$

$$= (e^{-(\lambda_w/n)d}P(X_{A1}) + P(X_{A2}))^{n-1} \times P(I) \times \int_{t_k=0}^{t_k=d} P(R_A)dt_k \tag{5.9}$$

From (5.3),

$$\int_{t_k=0}^{t_k=d} P(R_A)dt_k =$$
$$\int_{t_k=0}^{t_k=d}\left(\frac{\lambda_r}{n}\right)e^{-(\lambda_r/n)t_k}e^{-(\lambda_w/n)(t_k+2d)}\left[\left(\frac{\lambda_w}{\lambda}\right) + \left(\frac{\lambda_r}{\lambda}\right)e^{-(\lambda/n)(L-t_k-2d)}\right] \times$$
$$\left(1 - \left(\frac{\lambda_r}{\lambda}\right)e^{-(\lambda_w/n)L}\right)dt_k$$
$$= \left(\frac{\lambda_r}{n}\right)\left(1 - \left(\frac{\lambda_r}{\lambda}\right)e^{-(\lambda_w/n)L}\right) \times$$
$$\int_{t_k=0}^{t_k=d} e^{-(\lambda_r/n)t_k}e^{-(\lambda_w/n)(t_k+2d)}\left[\left(\frac{\lambda_w}{\lambda}\right) + \left(\frac{\lambda_r}{\lambda}\right)e^{-(\lambda/n)(L-t_k-2d)}\right]dt_k \tag{5.10}$$

The integral

$$\int_{t_k=0}^{t_k=d} e^{-(\lambda_r/n)t_k} e^{-(\lambda_w/n)(t_k+2d)} \left[\left(\frac{\lambda_w}{\lambda} \right) + \left(\frac{\lambda_r}{\lambda} \right) e^{-(\lambda/n)(L-t_k-2d)} \right] dt_k$$

$$= \int_{t_k=0}^{t_k=d} e^{-[(\lambda_r+\lambda_w)/n]t_k} e^{-2(\lambda_w/n)d} \left[\left(\frac{\lambda_w}{\lambda} \right) + \left(\frac{\lambda_r}{\lambda} \right) e^{-(\lambda/n)(L-t_k-2d)} \right] dt_k$$

$$= e^{-2(\lambda_w/n)d} \int_{t_k=0}^{t_k=d} e^{-(\lambda/n)t_k} \left[\left(\frac{\lambda_w}{\lambda} \right) + \left(\frac{\lambda_r}{\lambda} \right) e^{-(\lambda/n)(L-t_k-2d)} \right] dt_k$$

$$= e^{-2(\lambda_w/n)d} \times \left[\int_{t_k=0}^{t_k=d} \left(\frac{\lambda_w}{\lambda} \right) e^{-(\lambda/n)t_k} dt_k + \int_{t_k=0}^{t_k=d} \left(\frac{\lambda_r}{\lambda} \right) e^{-(\lambda/n)t_k} e^{-(\lambda/n)(L-t_k-2d)} dt_k \right]$$

$$= e^{-2(\lambda_w/n)d} \times \left[\int_{t_k=0}^{t_k=d} \left(\frac{\lambda_w}{\lambda} \right) e^{-(\lambda/n)t_k} dt_k + \int_{t_k=0}^{t_k=d} \left(\frac{\lambda_r}{\lambda} \right) e^{-(\lambda/n)(L-2d)} dt_k \right]$$

$$= e^{-2(\lambda_w/n)d} \times \left[\left(\frac{\lambda_w}{\lambda} \right) \left(\frac{n}{\lambda} \right) \left(1 - e^{-(\lambda/n)d} \right) + \left(\frac{\lambda_r}{\lambda} \right) e^{-(\lambda/n)(L-2d)} \int_{t_k=0}^{t_k=d} dt_k \right]$$

$$= e^{-2(\lambda_w/n)d} \times \left[\left(\frac{n\lambda_w}{\lambda^2} \right) \left(1 - e^{-(\lambda/n)d} \right) + \left(\frac{\lambda_r}{\lambda} \right) e^{-(\lambda/n)(L-2d)} d \right]$$

Therefore, equation 5.10 reduces to

$$\left(\frac{\lambda_r}{n} \right) \left[1 - \left(\frac{\lambda_r}{\lambda} \right) e^{-(\lambda_w/n)L} \right] e^{-2(\lambda_w/n)d} \times \left[\left(\frac{n\lambda_w}{\lambda^2} \right) \left(1 - e^{-(\lambda/n)d} \right) + \left(\frac{\lambda_r}{\lambda} \right) e^{-(\lambda/n)(L-2d)} d \right] \tag{5.11}$$

From (5.9) and (5.11),

$$P(A) = (e^{-(\lambda_w/n)d} P(X_{A1}) + P(X_{A2}))^{n-1} \times P(I) \times \left(\frac{\lambda_r}{n} \right) \left[1 - \left(\frac{\lambda_r}{\lambda} \right) e^{-(\lambda_w/n)L} \right] e^{-2(\lambda_w/n)d} \times \left[\left(\frac{n\lambda_w}{\lambda^2} \right) \left(1 - e^{-(\lambda/n)} \right) + \left(\frac{\lambda_r}{\lambda} \right) e^{-(\lambda/n)(L-2d)} d \right] \tag{5.12}$$

5.4.2 Case B

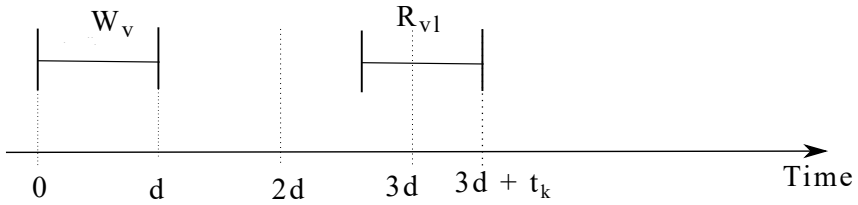


Figure 5.5: Case B.

R_{vl} finishes in the time interval $[3d, \min(4d, L-3d)]$: let us assume that R_{vl} finishes at $3d+t_k$, $0 < t_k \leq d$. (The symbol t_k is re-defined here and should not be confused to be the exact same symbol used in case A. We choose to use the same symbol in case B to draw parallels easily between the two cases). If (L - $3d$) ≤ 0, this case does not occur and so the probability of an anomaly involving W_v happening in this case is equal to the probability of no anomaly happening, which is equal to zero.

Otherwise, we can calculate the probability of R_{vl} finishing at time $3d + t_k$ from 0, represented by $P(R_B)$, similarly as in case A. The only difference from case A is that R_{vl} finishes at $3d + t_k$ instead of $2d + t_k$.

$$P(R_B) = \left(\frac{\lambda_r}{n}\right) e^{-(\lambda_r/n)t_k} e^{-(\lambda_w/n)(t_k+3d)} \left[\left(\frac{\lambda_w}{\lambda}\right) + \left(\frac{\lambda_r}{\lambda}\right) e^{-(\lambda/n)(L-t_k-3d)}\right] \times \left[1 - \left(\frac{\lambda_r}{\lambda}\right) e^{-(\lambda_w/n)L}\right] \tag{5.13}$$

Let the upper limit on the finish time of R_{vl} for this case be $U_B = \min(4d, L-3d)$. In the time interval [0, L], given R_{vl} finishes in $[3d, U_B]$, there is no start of a write or finish of a read at replica r_i that causes one or more of V1, V2, V3 and V4. For any replica $r_z \neq r_i$, any write starting in the interval [d, $d + t_k$] causes the anomaly V1. The probability of no such write starting in the same interval is $e^{-(\lambda_w/n)t_k}$.

Given no writes in this interval at any replica, neither of anomalies V2 and V4 happen in the interval [0, L] if any read finishing in the interval [$2d$, L] at $t_r, 0 \leq t_r \leq (L-2d)$ is preceded by at least one write starting at the same replica in the interval $[d + t_k, 2d + t_r]$. If a write concurrent with W_v starts after time 0 at $t_x, 0 \leq t_x \leq d$ at any replica, V3 does not happen in that replica in the interval [0, L] if any read finishing in the interval $[2d+t_x,$

$L]$ in the same replica is preceded by at least one write starting at the same replica in the interval $[d + t_k,\ 2d + t_r]$.

We see above that the constraining intervals for reads are a little different for the cases V2 and V4 and the case V3, while the constraining intervals for writes are the same. However, for simplicity of analysis, we assume that the intervals for reads to be the same for cases V2 and V4 and the case V3. The error introduced because of this assumption is limited by the fact that $d < L/3$ (see section 5.1) and $t_x \leq d$. The probability of one read finishing in the interval $[2d,\ L]$ at $2d + t_s, 0 \leq t_s \leq (L - 2d)$ and causing an anomaly can be calculated as:

$$\left(\frac{\lambda_r}{n}\right) e^{-(\lambda_r/n)t_s} e^{-(\lambda_w/n)(d-t_k+t_s)}$$

The probability of at least one such read finishing in the interval $[2d,\ L]$ can be calculated as:

$$\int_{t_s=0}^{t_s=L-2d} \left(\frac{\lambda_r}{n}\right) e^{-(\lambda_r/n)t_s} e^{-(\lambda_w/n)(d-t_k+t_s)} dt_s = \left(\frac{\lambda_r}{n}\right) e^{-(\lambda_w/n)(d-t_k)} \int_{t_s=0}^{t_s=L-2d} e^{-(\lambda/n)t_s} dt_s$$

$$= \left(\frac{\lambda_r}{\lambda}\right) e^{-(\lambda_w/n)(d-t_k)} \left(1 - e^{-(\lambda/n)(L-2d)}\right)$$

The probability of no such read in the same interval is:

$$1 - \left(\frac{\lambda_r}{\lambda}\right) e^{-(\lambda_w/n)(d-t_k)} \left(1 - e^{-(\lambda/n)(L-2d)}\right) \tag{5.14}$$

Using 5.14, given R_{vl} finishes at $3d + t_k$, the probability of no write starting or no read other than R_{vl} finishing in the interval $[0,\ L]$ and causing one or more of V1, V2, V3 and V4 is represented by $P(X_B)$, and can be calculated as below:

$$P(X_B) = e^{-(\lambda_w/n)t_k} \left(1 - \left(\frac{\lambda_r}{\lambda}\right) e^{-(\lambda_w/n)(d-t_k)} \left(1 - e^{-(\lambda/n)(L-2d)}\right)\right)$$

$$= e^{-(\lambda_w/n)t_k} - \left(\frac{\lambda_r}{\lambda}\right) e^{-(\lambda_w/n)d} \left(1 - e^{-(\lambda/n)(L-2d)}\right) \tag{5.15}$$

V3 can happen if a read finishes in the interval $[L,\ \infty]$ reading the value written by a write concurrent with W_v and starting after time 0, as in case A. Making similar assumptions as in case A, the probability of no read in any replica finishing in the interval $(L,\ \infty]$ causing

V3 in this case is also P(I) (see 5.8). We can now calculate the overall probability of no anomaly for case B similarly as in case A:

$$P(B) = \int_{t_k=0}^{t_k=U_B} P(R_B)\left(P(X_B)\right)^{n-1} P(I)dt_k = P(I) \times \int_{t_k=0}^{t_k=U_B} P(R_B)P(X_B)^{n-1}dt_k \quad (5.16)$$

From 5.13, we see that

$$P(R_B) = \left(\frac{\lambda_r}{n}\right) e^{-(\lambda_r/n)t_k} e^{-(\lambda_w/n)(t_k+3d)} \left[\left(\frac{\lambda_w}{\lambda}\right) + \left(\frac{\lambda_r}{\lambda}\right) e^{-(\lambda/n)(L-t_k-3d)}\right] \times \left[1 - \left(\frac{\lambda_r}{\lambda}\right) e^{-(\lambda_w/n)L}\right]$$

$$= \left(\frac{\lambda_r}{n}\right)\left[1 - \left(\frac{\lambda_r}{\lambda}\right) e^{-(\lambda_w/n)L}\right] e^{-(\lambda_w/n)3d} \left[\left(\frac{\lambda_w}{\lambda}\right) e^{-(\lambda/n)t_k} + \left(\frac{\lambda_r}{\lambda}\right) e^{-(\lambda/n)(L-3d)}\right]$$

$$= C_{B1}\left(C_{B2}e^{pt_k} + C_{B3}\right), \text{where}$$

$$C_{B1} = \left(\frac{\lambda_r}{n}\right)\left(1 - \left(\frac{\lambda_r}{\lambda}\right) e^{-(\lambda_w/n)L}\right) e^{-(\lambda_w/n)3d},\ C_{B2} = \left(\frac{\lambda_w}{\lambda}\right),\ p = -(\lambda/n)$$

$$\text{and } C_{B3} = \left(\frac{\lambda_r}{\lambda}\right) e^{-(\lambda/n)(L-3d)}$$

From (5.15),

$$P(X_B) = e^{-(\lambda_w/n)t_k} - \left(\frac{\lambda_r}{\lambda}\right) e^{-(\lambda_w/n)d}\left(1 - e^{-(\lambda/n)(L-2d)}\right)$$

$$= e^{mt_k} + C_{B4}, \text{ where } C_{B4} = -\left(\frac{\lambda_r}{\lambda}\right) e^{-(\lambda_w/n)d}\left(1 - e^{-(\lambda/n)(L-2d)}\right) \text{ and } m = -(\lambda_w/n)$$

Following binomial expansion,

$$P(X_B)^{n-1} = \left(e^{mt_k} + C_{B4}\right)^{n-1} = \left(e^{mt_k}\right)^{n-1} + \binom{n-1}{1}\left(e^{mt_k}\right)^{n-2} C_{B4} + \cdots + \binom{n-1}{n-2}\left(e^{mt_k}\right) C_{B4}^{n-2} + C_{B4}^{n-1}$$

Therefore, we can calculate P(B) from (5.16) as follows:

$$P(B) = P(I) \times \int_{t_k=0}^{t_k=U_B} P(R_B)P(X_B)^{n-1}dt_k$$

$$= P(I) \times \int_{t_k=0}^{t_k=U_B} C_{B1} \left(C_{B2} e^{pt_k} + C_{B3}\right) \left(e^{mt_k} + C_{B4}\right)^{n-1} dt_k$$

$$= P(I) \times C_{B1} \times (S_{B1} + S_{B2}), \text{ where} \tag{5.17}$$

$$S_{B1} = C_{B2} \times \sum_{i=0}^{n-1} \binom{n-1}{n-1-i} \left(\frac{e^{[m(n-1-i)+p]U_B} - 1}{m(n-1-i)+p}\right) C_{B4}^i, \text{ and}$$

$$S_{B2} = C_{B3} \times \sum_{i=0}^{n-1} \binom{n-1}{n-1-i} \left(\frac{e^{[m(n-1-i)]U_B} - 1}{m(n-1-i)}\right) C_{B4}^i$$

where i is the sequence number of terms in the series S_{B1} and S_{B2}, ranging from 0 to $(n-1)$.

5.4.3 Case C

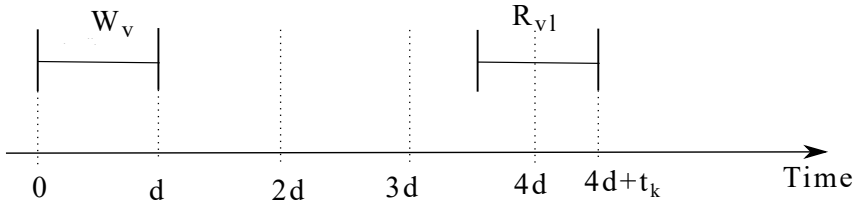


Figure 5.6: Case C.

R_{vl} finishes in the time interval $[4d, L-4d)]$: like in case B, if $(L$ - $4d) \leq 0$, this case does not occur and so the probability of no anomaly involving W_v happening is zero. Let us assume that R_{vl} finishes at $4d + t_k$, $0 < t_k \leq d$. The symbol t_k is re-defined in this case, as in case B.

If $(L-4d) > 0$, we can calculate the probability of R_{vl} finishing at time $4d + t_k$ from 0, represented by $P(R_C)$ similarly to (5.3).

$$P(R_C) = \left(\frac{\lambda_r}{n}\right) e^{-(\lambda_r/n)t_k} e^{-(\lambda_w/n)(t_k+4d)} \left[\left(\frac{\lambda_w}{\lambda}\right) + \left(\frac{\lambda_r}{\lambda}\right) e^{-(\lambda/n)(L-t_k-4d)}\right] \times$$
$$\left[1 - \left(\frac{\lambda_r}{\lambda}\right) e^{-(\lambda_w/n)L}\right]$$

Given R_{vl} finishes in $[4d,\ L-4d]$, there can be no start of a write or finish of a read in the interval $[0,\ L]$ at replica r_i that causes one or more of V1, V2, V3 and V4 . For any

replica $r_z \neq r_i$, any write starting in the interval $[d,\ 2d + t_k]$ causes the anomaly V1. The probability of no such write starting in the same interval is $e^{-(\lambda_w/n)(d+t_k)}$.

A read finishing at r_z in the interval $[2d,\ L]$ causes one of anomalies V2, V3 and V4 unless at least one write starts at the same replica after time d and before the finish of the read. However, we know that no writes can start in the interval $[d,\ 2d+t_k]$ for an anomaly to not occur. Consequently, there can be no reads finishing at r_z in the interval $[2d,\ 2d+t_k]$ for an anomaly to not occur, the probability of which is $e^{-(\lambda_r/n)t_k}$. In the interval $(2d+t_k,\ L]$, for an anomaly to not occur, no read can finish at r_z before the start of at least one write at the same replica after time d and before the finish of the read. Similarly to (5.4), this probability can be calculated to be:

$$\left(\frac{\lambda_w}{\lambda}\right) + \left(\frac{\lambda_r}{\lambda}\right) e^{-(\lambda/n)(L-2d-t_k)}$$

While the calculation above is true for avoiding anomaly cases V2 and V4, avoiding the anomaly V3 in the case of at least one write concurrent with W_v existing and starting after zero at r_z, requires no read finishing before at least one write in a sub-interval of $[2d,\ L]$. The head of the interval $[2d,\ L]$ shifts to the right in the sub-interval, depending on the start of the latest of such writes concurrent with W_v. However, like in case B, we ignore this nuance and consider the constraining intervals for reads at r_z to be the same for cases V2 and V4 and for case V3, for simplicity of analysis. Under the assumptions stated above, given R_{vl} finishes at $4d + t_k$, the probability of no write starting or no read other than R_{vl} finishing in the interval $[0,\ L]$ at any one replica and causing one or more of V1, V2, V3 and V4 is represented by $P(X_C)$, and can be calculated as below:

$$\begin{aligned} P(X_C) &= e^{-(\lambda_w/n)(d+t_k)} e^{-(\lambda_r/n)t_k} \left(\left(\frac{\lambda_w}{\lambda}\right) + \left(\frac{\lambda_r}{\lambda}\right) e^{-(\lambda/n)(L-2d-t_k)} \right) \\ &= \left(\frac{\lambda_w}{\lambda}\right) e^{-(\lambda_w/n)d} e^{-(\lambda/n)t_k} + \left(\frac{\lambda_r}{\lambda}\right) e^{-(\lambda_w/n)d} e^{-(\lambda/n)(L-2d)} \end{aligned} \tag{5.18}$$

Making similar assumptions as in case A, the probability of no read in any replica finishing in the interval $(L,\ \infty]$ causing V3 in this case is P(I) (see 5.8). We can now calculate the overall probability of no anomaly for case C similarly as in case B:

$$P(C) = \int_{t_k=0}^{t_k=(L-4d)} P(R_C) \left(P(X_C)\right)^{n-1} P(I) dt_k = P(I) \times \int_{t_k=0}^{t_k=(L-4d)} P(R_C) \left(P(X_C)\right)^{n-1} dt_k \tag{5.19}$$

From 5.4.3, we see that

$$P(R_C) = \left(\frac{\lambda_r}{n}\right) e^{-(\lambda_r/n)t_k} e^{-(\lambda_w/n)(t_k+4d)} \left[\left(\frac{\lambda_w}{\lambda}\right) + \left(\frac{\lambda_r}{\lambda}\right) e^{-(\lambda/n)(L-t_k-4d)}\right] \times \left[1 - \left(\frac{\lambda_r}{\lambda}\right) e^{-(\lambda_w/n)L}\right]$$

$$= \left(\frac{\lambda_r}{n}\right) \left[1 - \left(\frac{\lambda_r}{\lambda}\right) e^{-(\lambda_w/n)L}\right] e^{-(\lambda_w/n)4d} \left[\left(\frac{\lambda_w}{\lambda}\right) e^{-(\lambda/n)t_k} + \left(\frac{\lambda_r}{\lambda}\right) e^{-(\lambda/n)(L-4d)}\right]$$

$$= C_{C1}\left(C_{C2}e^{qt_k} + C_{C3}\right), \text{where}$$

$$C_{C1} = \left(\frac{\lambda_r}{n}\right) \left[1 - \left(\frac{\lambda_r}{\lambda}\right) e^{-(\lambda_w/n)L}\right] e^{-(\lambda_w/n)4d},\ C_{C2} = \left(\frac{\lambda_w}{\lambda}\right),\ q = -(\lambda/n)$$

$$\text{and } C_{C3} = \left(\frac{\lambda_r}{\lambda}\right) e^{-(\lambda/n)(L-4d)}$$

From (5.18),

$$P(X_C) = \left(\frac{\lambda_w}{\lambda}\right) e^{-(\lambda_w/n)d} e^{-(\lambda/n)t_k} + \left(\frac{\lambda_r}{\lambda}\right) e^{-(\lambda_w/n)d} e^{-(\lambda/n)(L-2d)}$$

$$= C_{C4}e^{st_k} + C_{C5},\ \text{where}$$

$$C_{C4} = \left(\frac{\lambda_w}{\lambda}\right) e^{-(\lambda_w/n)d}, C_{C5} = \left(\frac{\lambda_r}{\lambda}\right) e^{-(\lambda_w/n)d} e^{-(\lambda/n)(L-2d)} \text{ and } s = -(\lambda_w/n)$$

Following binomial expansion,

$$P(X_C)^{n-1} = \left(C_{C4}e^{st_k} + C_{C5}\right)^{n-1}$$

$$= (C_{C4})^{n-1}\left(e^{st_k}\right)^{n-1} + \binom{n-1}{1}(C_{C4})^{n-2}\left(e^{st_k}\right)^{n-2} C_{C5} + \cdots$$

$$+ \binom{n-1}{n-2}\left(C_{C4}e^{mt_k}\right)(C_{C5})^{n-2} + (C_{C5})^{n-1}$$

Therefore, we can calculate P(C) from (5.19) as follows:

$$P(C) = P(I) \times \int_{t_k=0}^{t_k=L-4d} P(R_C)\left(P(X_C)\right)^{n-1} dt_k$$

$$= P(I) \times \int_{t_k=0}^{t_k=L-4d} C_{C1} \left(C_{C2} e^{qt_k} + C_{C3}\right) \left(C_{C4} e^{st_k} + C_{C5}\right)^{n-1} dt_k$$

$$= P(I) \times C_{C1} \times (S_{C1} + S_{C2}), \text{ where} \tag{5.20}$$

$$S_{C1} = C_{C2} \times \sum_{i=0}^{n-1} \binom{n-1}{n-1-i} \left(\frac{e^{[s(n-1-i)+q](L-4d)} - 1}{s(n-1-i)+q}\right) (C_{C4})^{n-1-i} (C_{C5})^{i}, \text{ and}$$

$$S_{C2} = C_{C3} \times \sum_{i=0}^{n-1} \binom{n-1}{n-1-i} \left(\frac{e^{[s(n-1-i)](L-4d)} - 1}{s(n-1-i)}\right) (C_{C4})^{n-1-i} (C_{C5})^{i},$$

where i is the sequence number of terms in the series S_{C1} and S_{C2}, ranging from 0 to $(n-1)$.

5.4.4 Case D

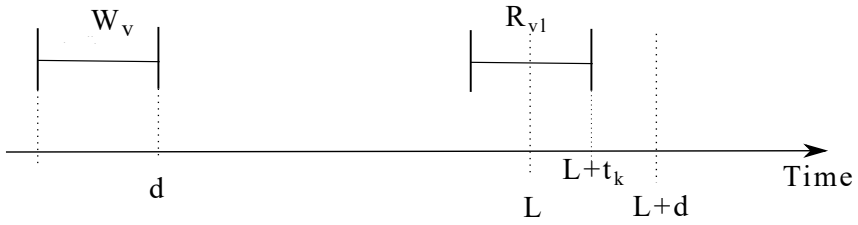


Figure 5.7: Case D.

R_{vl} finishes in the time interval [L,$L+d$]: let us assume that R_{vl} finishes at $L+t_k$, $0 < t_k \leq d$. The symbol t_k is re-defined in this case, as in the previous cases. A read of value v finishes in the interval [L, $L+d$] at time $(L+t_k)$ if a read finishes at a replica at time $(L+t_k)$ and no write starts at all other replicas in the interval [0, t_k] and no write starts in the replica the read finished in the interval [0, $L+t_k$]. The probability of a read of v at $(L+t_k)$ can thus be calculated as:

$$\lambda_r e^{-\lambda_r t_k} e^{-((n-1)/n)\lambda_w t_k} e^{-(\lambda_w/n)(L+t_k)} = \lambda_r e^{-\lambda_r t_k} e^{-\lambda_w t_k} e^{-(\lambda_w/n)L}$$

The probability of at least one read of value v in the interval [$L+t_k$, ∞] can be calculated similarly to (5.2) in case A and is equal to $\left(\frac{\lambda_r}{\lambda}\right) e^{-(\lambda_w/n)L}$. The probability of no such

read in the same interval is $1 - \left(\frac{\lambda_r}{\lambda}\right) e^{-(\lambda_w/n)L}$. We can now calculate $P(R_D)$, which is the probability of R_{vl} finishing at $L + t_k$ as follows:

$$P(R_D) = \lambda_r e^{-\lambda_r t_k} e^{-\lambda_w t_k} e^{-(\lambda_w/n)L} \left(1 - \left(\frac{\lambda_r}{\lambda}\right) e^{-(\lambda_w/n)L}\right) \tag{5.21}$$

Similarly to cases B and C, no anomaly happens at replica r_i. A write in any replica $r_z \neq r_i$ starting in the interval $(d,\ L + t_k - d)$ causes anomaly V1. Given no writes happen in replica r_z in the above interval, a read at r_z finishing in the interval $[2d,\ L + t_k]$ would read a value written by a write which is either concurrent or older than W_v, causing one of anomalies V2, V3 and V4 (similarly to cases B and C, we assume that the constraining intervals for reads are the same for cases V2 and V4 and for case V3 for simplicity of analysis, although they are not).

It is important to note here that $0 < t_k \leq d$, which means that if a read finishes in the interval $[2d,\ L + t_k]$ at r_z and reads a value written by a remote write, the earliest possible start time of this write is $L + t_k - L = t_k$ which is less than d. This implies that the remote write is either concurrent or older than W_v and that the read would cause one of V2, V3 and V4. This property enables us to calculate the probability of no anomaly in each replica independently in the above mentioned interval. Given there are no writes in the interval $(d,\ L + t_k - d)$ at r_z, for no anomaly to happen, there should be no reads in the interval $(2d,\ L + t_k - d)$ and no read before at least one write in the interval $[L + t_k - d,\ L + t_k]$. The probability of no read before at least one write in the interval $[L + t_k - d,\ L + t_k]$ can be calculated similarly to (5.1) and is equal to:

$$\left(\frac{\lambda_w}{\lambda}\right) + \left(\frac{\lambda_r}{\lambda}\right) e^{-(\lambda/n)d}$$

The probability that no anomaly occurs at r_z in the interval $[0,\ L]$ given R_{vl} finishes at $L + t_k$, represented by $P(X_D)$ can now be calculated as follows:

$$\begin{aligned} P(X_D) &= e^{-(\lambda_w/n)(L+t_k-2d)} e^{-(\lambda_r/n)(L+t_k-3d)} \left(\left(\frac{\lambda_w}{\lambda}\right) + \left(\frac{\lambda_r}{\lambda}\right) e^{-(\lambda/n)d}\right) \\ &= e^{-(\lambda_w/n)(L-2d)} e^{-(\lambda_r/n)(L-3d)} \left(\left(\frac{\lambda_w}{\lambda}\right) + \left(\frac{\lambda_r}{\lambda}\right) e^{-(\lambda/n)d}\right) e^{-(\lambda/n)t_k} \end{aligned} \tag{5.22}$$

In the interval $[L + t_k,\ \infty]$, the only possible anomaly is V3, and the probability of V3 not happening can be calculated similarly to (5.8) in case A and this probability is $P(I)$. We ignore the fact that R_{vl} happens at r_i in the calculation of $P(I)$ to simplify our

analysis. Also, like in the previous cases, we assume that the events of no anomaly in the interval $[0,\ L + t_k]$ and no anomaly in the interval $[L + t_k,\ \infty]$ are independent of each other. We can now calculate the overall probability of no anomaly in case D, represented by $P(D)$ as below:

$$P(D) = \int_{t_k=0}^{t_k=d} \left(P(R_D) \times (P(X_D))^{n-1} \times P(I)\right) dt_k$$

$$= P(I) \int_{t_k=0}^{t_k=d} \left[\lambda_r e^{-\lambda_r t_k} e^{-\lambda_w t_k} e^{-(\lambda_w/n)L} \left(1 - \left(\frac{\lambda_r}{\lambda}\right) e^{-(\lambda_w/n)L}\right)\right] \times$$
$$\left[e^{-(\lambda_w/n)(L-2d)} e^{-(\lambda_r/n)(L-3d)} \left(\left(\frac{\lambda_w}{\lambda}\right) + \left(\frac{\lambda_r}{\lambda}\right) e^{-(\lambda/n)d}\right) e^{-(\lambda/n)t_k}\right]^{n-1} dt_k$$

$$= P(I) \times \left[\lambda_r e^{-\lambda_r t_k} e^{-(\lambda_w/n)L} \left(1 - \left(\frac{\lambda_r}{\lambda}\right) e^{-(\lambda_w/n)L}\right)\right] \times$$
$$\left[e^{-(\lambda_w/n)(L-2d)} e^{-(\lambda_r/n)(L-3d)} \left(\left(\frac{\lambda_w}{\lambda}\right) + \left(\frac{\lambda_r}{\lambda}\right) e^{-(\lambda/n)d}\right)\right]^{n-1} \int_{t_k=0}^{t_k=d} e^{-\lambda t_k} e^{-((n-1)/n)\lambda t_k} dt_k \tag{5.23}$$

$$\text{The integral } \int_{t_k=0}^{t_k=d} e^{-\lambda t_k} e^{-((n-1)/n)\lambda t_k} dt_k = \int_{t_k=0}^{t_k=d} e^{-((2n-1)/n)\lambda t_k} dt_k$$
$$= \left(\frac{n}{(2n-1)\lambda}\right) \left(1 - e^{-((2n-1)/n)\lambda d}\right)$$

From (5.23), we can then calculate $P(D)$ as follows:

$$P(D) = P(I) \times \left(\lambda_r e^{-\lambda_r t_k} e^{-(\lambda_w/n)L} \left(1 - \left(\frac{\lambda_r}{\lambda}\right) e^{-(\lambda_w/n)L}\right)\right) \times$$
$$\left[e^{-(\lambda_w/n)(L-2d)} e^{-(\lambda_r/n)(L-3d)} \left(\left(\frac{\lambda_w}{\lambda}\right) + \left(\frac{\lambda_r}{\lambda}\right) e^{-(\lambda/n)d}\right)\right]^{n-1} \times \tag{5.24}$$
$$\left[\left(\frac{n}{(2n-1)\lambda}\right) \left(1 - e^{-((2n-1)/n)\lambda d}\right)\right]$$

5.4.5 Case E

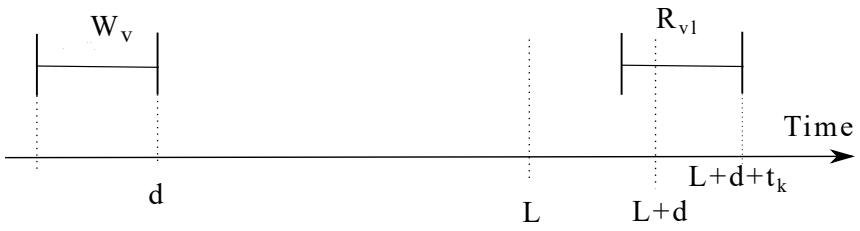


Figure 5.8: Case E.

R_{vl} finishes in the time interval $[L+d,\infty)$: let us assume that R_{vl} finishes at $L+d+t_k$, 0 $< t_k \leq d$. The symbol t_k is re-defined, as in the previous cases. A read of value v finishing after time $L+d$ implies no writes starting in the interval [0, d], which means V3 is not possible in this case. A read finishing in the interval [2d, L] can cause V2 or V3 if it reads a value written by a write that finishes before 0 or is concurrent with W_v respectively. V1 happens if a write starts in the interval (d, $L+t_k$).

A read finishing at $L+d+t_k$ at any replica r_f reads the value v if there is no write starting in all replicas in the interval [0, $d+t_k$) and there is no write starting at r_f in the interval [$d+t_k$, $L+d+t_k$). The probability of no read of v in the interval [$L+d+t_k$, ∞] can be calculated similarly to (5.2) and is equal to $1-\left(\frac{\lambda_r}{\lambda}\right)e^{-(\lambda_w/n)L}$. The probability of R_{vl} finishing at $L+t_k$, represented by $P(R_E)$ can now be calculated as follows:

$$P(R_E) = \lambda_r e^{-\lambda_r t_k} e^{-\lambda_w(t_k+d)} e^{-(\lambda_w/n)L}\left(1-\left(\frac{\lambda_r}{\lambda}\right)e^{-(\lambda_w/n)L}\right)$$

$$= \lambda_r e^{-\lambda_w d} e^{-(\lambda_w/n)L}\left(1-\left(\frac{\lambda_r}{\lambda}\right)e^{-(\lambda_w/n)L}\right)e^{-\lambda t_k} \tag{5.25}$$

As in the previous cases, no anomaly can happen in replica r_i. For any replica $r_z \neq r_i$, a write starting in the interval (d, $L+t_k$) causes V1. Given there are no writes in the interval, a read finishing in the interval [2d, L] causes one of V2 or V4. The probability that no anomaly occurs at replica r_z, represented by $P(X_E)$ can now be calculated as follows:

$$P(X_E) = e^{-(\lambda_w/n)(L+t_k-d)} e^{-(\lambda_r/n)(L-2d)} = e^{-(\lambda_w/n)(L-d)} e^{-(\lambda_r/n)(L-2d)} e^{-(\lambda_w/n)t_k} \tag{5.26}$$

The probability $P(X_E)$ only depends on the local reads and writes of replica r_z. So, the probability of no anomaly in all replicas other than r_i can be calculated by simply

multiplying the probabilities of no anomaly in individual replicas. The overall probability of no anomaly in case E, represented by $P(E)$ is calculated as follows:

$$\begin{aligned}
P(E) &= \int_{t_k=0}^{t_k=\infty} P(R_E) \times (P(X_E))^{n-1}\, dt_k \\
&= \int_{t_k=0}^{t_k=\infty} \left[\lambda_r e^{-\lambda_w d} e^{-(\lambda_w/n)L} \left(1 - \left(\frac{\lambda_r}{\lambda}\right) e^{-(\lambda_w/n)L}\right) e^{-\lambda t_k}\right] \times \\
&\quad \left(e^{-(\lambda_w/n)(L-d)} e^{-(\lambda_r/n)(L-2d)} e^{-(\lambda_w/n)t_k}\right)^{n-1} dt_k \\
&= \left[\lambda_r e^{-\lambda_w d} e^{-(\lambda_w/n)L} \left(1 - \left(\frac{\lambda_r}{\lambda}\right) e^{-(\lambda_w/n)L}\right)\right] \times \\
&\quad \left(e^{-(\lambda_w/n)(L-d)} e^{-(\lambda_r/n)(L-2d)}\right)^{n-1} \int_{t_k=0}^{t_k=\infty} e^{-\left(\lambda + \left(\frac{n-1}{n}\right)\lambda_w\right)t_k} dt_k
\end{aligned} \tag{5.27}$$

The integral $\int_{t_k=0}^{t_k=\infty} e^{-\left(\lambda+\left(\frac{n-1}{n}\right)\lambda_w\right)t_k} dt_k = \left(\frac{-1}{\lambda + \left(\frac{n-1}{n}\right)\lambda_w}\right)(0-1) = \left(\frac{n}{n\lambda + (n-1)\lambda_w}\right)$

From (5.27), $P(E)$ is calculated as follows:

$$\begin{aligned}
P(E) = &\left(\frac{n}{n\lambda + (n-1)\lambda_w}\right) \times \left(\lambda_r e^{-\lambda_w d} e^{-(\lambda_w/n)L} \left(1 - \left(\frac{\lambda_r}{\lambda}\right) e^{-(\lambda_w/n)L}\right)\right) \times \\
&\left(e^{-(\lambda_w/n)(L-d)} e^{-(\lambda_r/n)(L-2d)}\right)^{n-1}
\end{aligned} \tag{5.28}$$

5.4.6 Case F

R_{vl} finishes in the time interval $(d,\, 2d)$ or no read of value v occurs at all. If R_{vl} finishes in $(d,\, 2d)$, it is concurrent with W_v. In that case, or in the case of no read of v at all, the only anomaly that can happen is V2. All the other anomaly cases require a read of v starting after $finish(W_v)$. The probability of R_{vl} finishing in the time interval $(d,\, 2d)$ or no read of value v at all can be calculated as the probability of no read of v in the interval $[2d,\, \infty]$.

A read of value v in the interval $(2d,\, L]$ can only happen at replica r_i, if a read finishes in the interval and no write starts at r_i after time 0 and before the finish of the read. Let such a read finish at $t_u, 0 < t_u \leq L$. The probability of such a read finishing at t_u is $(\lambda_r/n)\, e^{-(\lambda_r/n)t_u} e^{-(\lambda_w/n)(2d+t_u)}$. The probability that at least one such read finishes in the interval $(2d,\, L]$ is:

$$\int_{t_u=0}^{t_u=(L-2d)} (\lambda_r/n)\, e^{-(\lambda_r/n)t_u} e^{-(\lambda_w/n)(2d+t_u)} dt_u = (\lambda_r/n)\, e^{-(\lambda_w/n)(2d)} \int_{t_u=0}^{t_u=(L-2d)} e^{-(\lambda/n)t_u} dt_u$$

$$= \left(\frac{\lambda_r}{\lambda}\right) e^{-(\lambda_w/n)(2d)} \left(1 - e^{-(\lambda/n)(L-2d)}\right) \tag{5.29}$$

In the interval $[L, \infty)$, the probability of at least one read of v can be calculated similarly to (5.2) in case A and is equal to $\left(\frac{\lambda_r}{\lambda}\right) e^{-(\lambda_w/n)L}$. Making the simplifying assumption that the events of no read of v finishing in the interval $[2d, L)$ and in the interval $[L, \infty)$ are independent of each other, the probability of no read of v finishing in the composite interval $[2d, \infty)$ can be calculated by multiplying the probabilities of no read of v finishing in the individual intervals. This probability, represented by $P(R_F)$, is calculated as follows:

$$P(R_F) = \left(1 - \left(\frac{\lambda_r}{\lambda}\right) e^{-(\lambda_w/n)(2d)} \left(1 - e^{-(\lambda/n)(L-2d)}\right)\right) \left(1 - \left(\frac{\lambda_r}{\lambda}\right) e^{-(\lambda_w/n)L}\right) \tag{5.30}$$

The finish of a read reading a value written by a write that finishes before $start(W_v)$ has to be before time L after which all reads in any replica read value v or newer. Reads of a value written by a write that finishes before $start(W_v)$ will cause anomaly V2 if the read finishes in the interval $[2d, L]$. In the interval $[2d, L-d]$, a value written by a write that finishes before $start(W_v)$ and also happens at a remote replica cannot be read. So, in this time interval, we only have to reason about reads and writes that happen locally in each replica to calculate the probability of V2 not happening. If $(L - 3d) \leq 0$, this case does not happen and the probability of no anomaly in this interval, $P(X_F) = 0$. In a replica r_z, in the interval $[2d, L-d]$, V2 happens if a read finishes at $t_v, 0 \leq t_v \leq (L - 3d)$ and there is no write starting in the interval $[-d, 2d + t_v]$. The probability of this is $(\lambda_r/n)\, e^{-(\lambda_r/n)t_v} e^{-(\lambda_w/n)(3d+t_v)}$. The probability of at least one read finishing in the interval $[2d, L-d]$ and causing V2 is:

$$\int_{t_v=0}^{t_v=(L-3d)} (\lambda_r/n)\, e^{-(\lambda_r/n)t_v} e^{-(\lambda_w/n)(3d+t_v)} dt_v = (\lambda_r/n)\, e^{-(\lambda_w/n)(3d)} \int_{t_v=0}^{t_v=(L-3d)} e^{-(\lambda/n)t_v} dt_v$$

$$= \left(\frac{\lambda_r}{\lambda}\right) e^{-(\lambda_w/n)(3d)} \left(1 - e^{-(\lambda/n)(L-3d)}\right)$$

The probability of no such read, or the probability of no anomaly in the interval $[2d, L-d]$ is then calculated as follows:

$$P(X_F) = 1 - \left(\frac{\lambda_r}{\lambda}\right) e^{-(\lambda_w/n)(3d)} \left(1 - e^{-(\lambda/n)(L-3d)}\right) \tag{5.31}$$

A read finishing in the interval $[L - d, L]$ can read a value written by a write that finishes before 0 and happens either at the local or a remote replica. The probability of a

read finishing at any replica r_x in the interval $[L-d,\ L]$ at time $t_w, 0 \leq t_w \leq d$ is $\lambda_r e^{-\lambda_r t_x}$. Given that there is a read in the above mentioned time interval, V2 happens if there is no write at r_x in the interval $[-(d-t_x),\ L-d+t_x]$ and if there is no write in all other replicas in the time interval $[-d,\ -(d-t_x)]$. The probability of at least one such read can hence be calculated as:

$$\int_{t_x=0}^{t_x=d} \lambda_r e^{-\lambda_r t_x} \left(e^{-\lambda_w t_x} e^{-(\lambda_w/n)L}\right) dt_x = \lambda_r e^{-(\lambda_w/n)L} \int_{t_x=0}^{t_x=d} e^{-\lambda t_x} dt_x$$

$$= \left(\frac{\lambda_r}{\lambda}\right) e^{-(\lambda_w/n)L}(1 - e^{-\lambda d})$$

The probability that no such read occurs is represented by $P(Y_F)$ and can hence be calculated as follows:

$$P(Y_F) = 1 - \left(\frac{\lambda_r}{\lambda}\right) e^{-(\lambda_w/n)L}(1 - e^{-\lambda d}) \tag{5.32}$$

We assume an independence of the events of no anomaly in interval $[2d,\ L-d]$ and the interval $[L-d,\ L]$ like in the previous cases, to have a simpler analysis. The overall probability that there is no anomaly for case E is represented by $P(F)$ and can be calculated as follows:

$$P(F) = P(R_F) \times P(X_F) \times P(Y_F) \tag{5.33}$$

The overall probability that W_v does not participate in an anomaly, represented by $P(N)$, can be calculated as the sum of probabilities of the disjoint cases A, B, C, D, E and F as follows:

$$P(N) = P(A) + P(B) + P(C) + P(D) + P(E) + P(F)$$

. The probability that W_v participates in an anomaly is $1 - P(N)$.

5.5 Evaluation

5.5.1 Experimental setup

The results of the mathematical model were compared against a custom built simulator written in Java. The simulator takes the number of replicas n, the average throughput λ, the proportion of reads ρ, the average one-way latency L and the symmetrical artificial delay added to each operation d as input. It then simulates 10^6 reads and writes (according to the read-proportion ρ) to the same key, each following exponential inter-arrival times in

n replicas and records the history of these operations into a file. We assume that the clients triggering the reads and writes are co-located with the storage servers. The proportion of positive scores is then calculated using the regular metric for consistency (see definition 4).

The predictions of staleness of the model were also compared against an Apache Cassandra cluster. The Cassandra cluster is configured similarly as described in chapter 4, section 4.4.1. However, in this case, we use one *m4.large* ec2 instance per zone across four availability zones. The availability zones used are: *US West (Oregon)*, *EU (Ireland)*, *Asia Pacific (Tokyo)* and *Asia Pacific (Sydney)*. The average one-way network delay (L) is measured to be 85 ms.

5.5.2 Obtained results

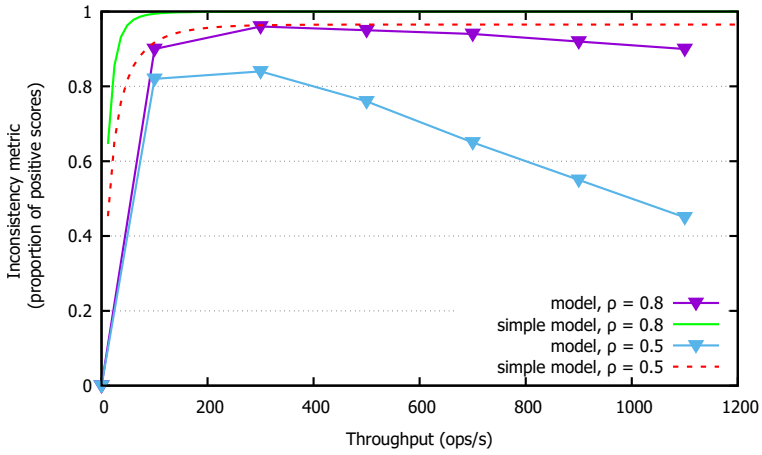
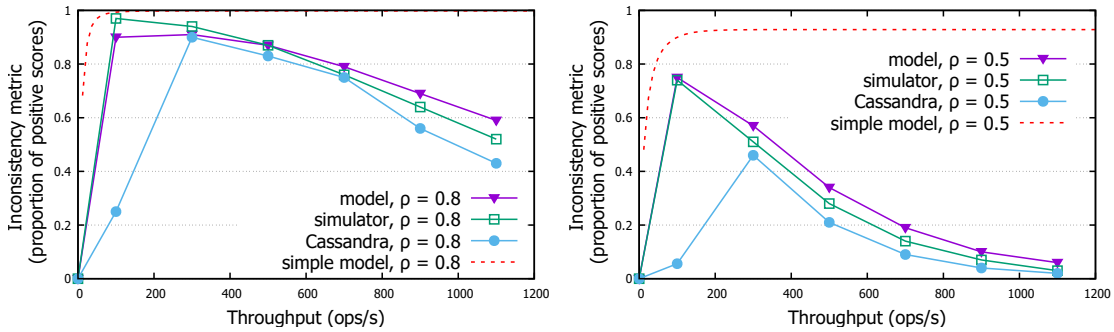


Figure 5.9: Comparison between the new and a simpler mathematical model, AD = 5 ms.

The probability obtained in the mathematical model is equal to the proportion of total operations participating in anomalies, for an infinite history. The figure 5.9 compares the predictions for the proportion of inconsistent operations obtained from the mathematical model presented in the thesis, to predictions from a simpler model [13], with a small value of d (= 5 ms). It must be noted here that the simple model does not take the parameter d into account, and d only affects the predictions of the newer model (presented in this thesis). The number of replicas, n is set to 5 and the average one-way latency L is set at 71 time units, which is the measured one-way latency for our Cassandra cluster used in SPECSHIFT (see chapter 4). We see that the simple model predicts a much higher

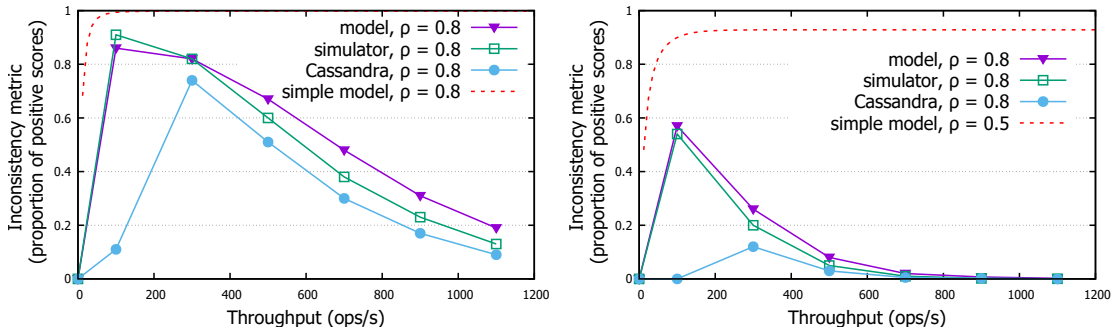
proportion of inconsistent operations compared to the new model, especially for higher values of throughput.

The figures 5.10a, 5.10b, 5.11a, and 5.11b compare predictions for proportion of inconsistent operations obtained from the simple model, the new mathematical model, from a simulator running 10^6 operations, and from an Apache Cassandra cluster. The value of n is set at 4 and the proportion of inconsistent operations is predicted/measured for increasing values of overall throughput. The proportion of inconsistent operations is computed/measured for ADs of 10 ms and 20 ms and for read-proportions of 0.8 and 0.5.



(a) $\rho = 0.8$. (b) $\rho = 0.5$.

Figure 5.10: Comparison of predicted proportion of positive scores between the mathematical model, the simulator and Cassandra, AD = 10 ms



(a) $\rho = 0.8$. (b) $\rho = 0.5$.

Figure 5.11: Comparison of predicted proportion of positive scores between the mathematical model, the simulator and Cassandra, AD = 20 ms

5.5.3 Discussion

We see that the predictions of the simple model are pessimistic with respect to the values obtained from Cassandra. The new model and the simulator trace each other closely and are also quite close to the values obtained from Cassandra for high throughputs. The disparity between the model (and simulator) and Cassandra for low throughputs could be due to the fact that operations arrivals at extremely low throughputs (a few tens of operations per server per second) may not follow Poisson arrivals strictly. This is conjecture though, and the exact reason for the disparity has not been explored in this body of work.

We see that the proportion of inconsistent operations observed in the Cassandra cluster is always lower than the values predicted by the model and the simulator. This is due to processing delays on the storage servers, which add to the effect of artificial delays and lower inconsistency. However, for higher values of d (roughly ≥ 10 ms), the effect of the artificial delay dominates over processing delays, which results in good predictions by the mathematical model. The model presented in this thesis can be used to calculate a fairly tight upper bound on the proportion of positive scores in a distributed storage system when all operations are delayed by the same time units. Qualitatively, the mathematical model is able to correctly predict that the proportion of positive scores decreases with an increase in λ beyond a point, and the rate of decrease of the proportion of positive scores is more at lower values of read-proportion ρ or at higher values of the artificial delay d.

5.6 Summary

In this chapter, I extend an existing simple probabilistic model [13] of consistency in distributed storage systems operating under weak consistency settings to account for ADs. I assume that the delays are constant for all operations and are injected before reads and after writes. Operation arrivals are assumed to follow a Poisson process. I derive a mathematical formula that relates the proportion of stale reads to environment parameters like proportion of reads (and writes), the number of storage servers, the network latency between them, the read/write throughput, and the constant AD. The predictions of staleness from the mathematical model are compared against a custom-built simulator of the storage system written in Java, and a real world storage system (Apache Cassandra). Experiments show that the predictions of staleness from the model match the values from the simulator closely, and also match the values from Cassandra for high values of total throughput.

Chapter 6

Conclusion

Distributed systems are constrained by the finite propagation speed of information. This means that in distributed key-value storage systems, stale reads are unavoidable under weak consistency settings, even in the absence of failures. In geo-replicated systems, the probability of stale values being read is amplified due to high network latencies between storage servers. Using strong client-side consistency settings (in the form of strict quorums) can help eliminate consistency anomalies but increases operation latencies manifold, especially when one-way network latency between the storage servers is large. This is typically the case when the storage servers span over multiple continents. Some modern businesses have preferred high availability over strict consistency requirements which has lead to an increased use of Dynamo-like eventually consistent key value stores in the industry. However, eventual consistency is often poorly understood and the guarantees provided by eventually consistent systems are not well defined. This leads to a pressing need for a middle ground and systems to tune consistency and operation latencies according to the specific requirements of a business.

6.1 Research contributions

This work introduces an adaptive tuning framework (SPECSHIFT) which controls the overall percentage of stale reads observed in a cluster by injecting artificial delays to read and write operations. The system is able to adjust latency of operations in a distributed cluster to meet a specific target proportion of positive (consistency anomaly) scores. The prediction of the delay to be injected at each iteration of the control loop is calculated by analyzing a distribution of consistency anomaly scores from operation histories obtained

from the storage servers. Although similar adaptive control loops have been designed before [40], the predictions are made in such systems using generic mechanisms without using insights from the history of operations that are available at each iteration other than the overall percentage of positive scores. Specifically, the distribution of these scores contain information which can be used to make very accurate predictions. Owing to this, the SPECSHIFT feedback loop achieves convergence to the optimum value of delay to meet the target proportion much faster than the state-of-art solution (PCAP).

This work also presents a probabilistic analysis of eventual consistency and a mathematical formula that predicts the probability of a given write participating in a consistency anomaly, from a number of parameters that define the environment of the storage system. A mathematical formulation of inconsistency in distributed storage systems operating under weak consistency settings was first introduced by Dr. Golab, the results of which were published in a joint work [13]. This work extends the simple mathematical model described there by accounting for artificial delays, which would allow fine-grained control over the consistency and latency of operations observed in a storage system. We see from chapter 5 that the probabilistic analysis of consistency anomalies can be tedious because anomalies are caused by the interplay of multiple operations which may overlap in time. Nevertheless, the mathematical model is able to match the values of expected staleness calculated from a custom built simulator of the storage system running 10^6 operations (per experiment) closely. The staleness predicted by the model is also matches the staleness observed in a widely used real-world distributed database (Apache Cassandra) closely for high values of throughput.

The extended probabilistic model helps us estimate the effect of processing delays at the storage servers on overall inconsistency, thus correcting the pessimistic predictions (compared to a real-world storage system) of the simple model at high values of throughput, to an extent. The mathematical formula also enables us to study the relationships between the environment parameters (overall throughput, read/write proportion, read and write delays, one-way network latency, number of replicas, etc.) and consistency in a storage system, without the need to perform the tedious and resource-intensive task of setting up a distributed cluster.

6.2 Learnings

The broad learnings from my research on the consistency-latency trade-off can be summarized as below:

1. The distribution of consistency anomaly scores in a history (see histograms in the figure 4.1)contains information which could be vital to understanding the consistency-latency trade-off. The experiments on the adaptive tuning framework described in chapter 4 validates *spectral shifting* (experimentally), but the phenomenon in itself could have broader implications. For example, spectral shifting allows developers to estimate the value of latency required to get rid of all consistency anomalies less than or equal to a particular score, by looking at a score histogram. I found that score histograms typically have a long 'tail' - which implies that there would be a large latency cost associated with a system that tolerates no staleness whatsoever. It also points to diminishing gains in consistency with increasing latency beyond a point. SPECSHIFT predicts a value of latency given a particular target proportion of staleness. The score histograms can guide developers on how to set the target value of staleness in the first place.

2. It is possible to model consistency in eventually consistent systems mathematically using probabilistic calculations. I assume memoryless arrivals (that follow a Poisson process) of operations in chapter 5, but the derivation can be extended to assume other distributions on operation arrivals as well. Such a mathematical calculation can be tedious because of the interplay of multiple operations that causes an anomaly. However, a mathematical formula is invaluable to understanding consistency in distributed storage systems and how it relates to not only latency of operations, but also other environment parameters like proportion of reads (and writes), the number of storage servers, the network latency between them, and the read/write throughput.

6.3 Future work

Figure 4.1 in chapter 4 shows score histograms for two different histories of operations. The shape of the score histogram and how it relates to the environment parameters under which a corresponding history is collected has not been studied in detail yet. A model to predict the shape of the score histogram accurately would help us answer questions like - does the score histogram always have a long 'tail'? If indeed there is always a diminishing gain in consistency on increasing latency beyond a point, what is the rate of this decay and what are the factors that control it?

The mathematical model presented in this thesis assumes full replication and a symmetric constant delay to all operations. It can be extended to incorporate partial replication or unequal read and write delays. Such a model will enable us to understand how delaying

reads or writes exclusively affects consistency. A similar analysis could be done for other fine-tuning mechanisms like CPQ [36] which would enable us to compare these techniques comprehensively without having to rely on experimental data. Such an analysis can go a long way in quantifying the trade-offs related to distributed storage systems concretely, and allowing application developers to estimate the consistency and latency of operations in eventually consistent storage systems conveniently and accurately.

References

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

[2] A. Aiyer, L. Alvisi, and R. A. Bazzi. On the availability of non-strict quorum systems. In *Proc. of the 19th International Symposium on Distributed Computing (DISC)*, pages 48–62, 2005.

[3] Amazon's SimpleDB. Available at http://aws.amazon.com/simpledb.

[4] E. Anderson, X. Li, A. Merchant, M. A. Shah, K. Smathers, J. Tucek, M. Uysal, and J. J. Wylie. Efficient eventual consistency in Pahoehoe, an erasure-coded key-blob archive. In *Proc. of the 2010 IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, pages 181–190, 2010.

[5] E. Anderson, X. Li, M. A. Shah, J. Tucek, and 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)*, pages 1–16, 2010.

[6] M. S. Ardekani and D. B. Terry. A self-configurable geo-replicated cloud storage system. In *11th USENIX Symposium on Operating Systems Design and Implementation (OSDI 14)*, pages 367–381, 2014.

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

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

[9] D. Bermbach and 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)*, pages 1:1–1:6, 2011.

[10] E. A. Brewer. Towards robust distributed systems (Invited Talk). In *Proc. of the 19th ACM SIGACT-SIGOPS Symposium on Principles of Distributed Computing (PODC)*, page 7, 2000.

[11] J. F. Cantin, M. H. Lipasti, and J. E. Smith. The complexity of verifying memory coherence and consistency. *IEEE Transactions on Parallel and Distributed Systems*, 16(7):663–671, July 2005.

[12] F. Chang, J. Dean, S. Ghemawat, W. C. Hsieh, D. A. Wallach, M. Burrows, T. Chandra, A. Fikes, and R. E. Gruber. Bigtable: A distributed storage system for structured data. *ACM Trans. Comput. Syst.*, 26(2):4:1–4:26, 2008.

[13] S. Chatterjee and W. Golab. Brief announcement: A probabilistic performance model and tuning framework for eventually consistent distributed storage systems. In *Proc. of the ACM Symposium on Principles of Distributed Computing (PODC)*, pages 259–261, 2017.

[14] S. Chatterjee and W. Golab. Self-tuning eventually-consistent data stores. In *19th International Symposium on Stabilization, Safety, and Security of Distributed Systems (SSS)*, 2017.

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

[16] B. F. Cooper, A. Silberstein, E. Tam, R. Ramakrishnan, and R. Sears. Benchmarking cloud serving systems with YCSB. In *ACM Symposium on Cloud Computing (SoCC)*, pages 143–154, 2010.

[17] J. C. Corbett, J. Dean, M. Epstein, A. Fikes, C. Frost, J. J. Furman, S. Ghemawat, A. Gubarev, C. Heiser, P. Hochschild, W. Hsieh, S. Kanthak, E. Kogan, H. Li, A. Lloyd, S. Melnik, D. Mwaura, D. Nagle, S. Quinlan, R. Rao, L. Rolig, Y. Saito, M. Szymaniak, C. Taylor, R. Wang, and D. Woodford. Spanner: Google's globally-distributed database. In *Proc. USENIX Conference on Operating Systems Design and Implementation (OSDI)*, pages 251–264, 2012.

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

[19] H. Fan, S. S. Chatterjee, and W. Golab. Watca: The waterloo consistency analyzer. In *IEEE 32nd International Conference on Data Engineering (ICDE)*, pages 1398–1401, 2016.

[20] A. Fekete, S. N. Goldrei, and J. P. Asejo. Quantifying isolation anomalies. In *Proc. of the 35th International Conference on Very Large Data Bases (VLDB)*, pages 467–478, August 2009.

[21] P. Gibbons and E. Korach. Testing shared memories. *SIAM Journal on Computing*, 26:1208–1244, August 1997.

[22] D. K. Gifford. Weighted voting for replicated data. In *Proc. of the 7th ACM Symposium on Operating Systems Principles (SOSP)*, pages 150–162, 1979.

[23] W. Golab, J. Hurwitz, and X. Li. On the k-atomicity-verification problem. In *Proc. of the 33rd International Conference on Distributed Computing Systems (ICDCS)*, pages 591–600, 2013.

[24] W. Golab, X. Li, and 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)*, pages 197–206, 2011.

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

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

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

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

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

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

[31] L. Lamport. How to make a multiprocessor computer that correctly executes multiprocess programs. *IEEE Transactions on Computers*, C-28(9):690–691, September 1979.

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

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

[34] W. Lloyd, M. J. Freedman, M. Kaminsky, and 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)*, pages 401–416, 2011.

[35] D. Malkhi, M. K. Reiter, and R. N. Wright. Probabilistic quorum systems. In *Proc. of the 16th ACM Symposium on Principles of Distributed Computing (PODC)*, pages 267–273, 1997.

[36] M. McKenzie, H. Fan, and 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*, pages 1708–1717, 2015.

[37] J. Misra. Axioms for memory access in asynchronous hardware systems. *ACM Transactions on Programming Languages and Systems*, 8(1):142–153, January 1986.

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

[39] S. Patil, M. Polte, K. Ren, W. Tantisiriroj, L. Xiao, J. López, G. Gibson, A. Fuchs, and B. Rinaldi. YCSB++: benchmarking and performance debugging advanced features in scalable table stores. In *Proc. ACM Symposium on Cloud Computing (SoCC)*, pages 9:1–9:14, 2011.

[40] M. R. Rahman, L. Tseng, S. Nguyen, I. Gupta, and N. H. Vaidya. Characterizing and adapting the consistency-latency tradeoff in distributed key-value stores. *ACM Transactions on Autonomous and Adaptive Systems*, 11(4):20:1–20:36.

[41] B. Schroeder, A. Wierman, and M. Harchol-Balter. Open versus closed: A cautionary tale. In *Proceedings of the 3rd Conference on Networked Systems Design & Implementation - Volume 3*, page 18, 2006.

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

[43] A. Singla, U. Ramachandran, and J. Hodgins. Temporal notions of synchronization and consistency in Beehive. In *Proc. of the Ninth ACM Symposium on Parallel Algorithms and Architectures (SPAA)*, pages 211–220, June 1997.

[44] R. N. Taylor. Complexity of analyzing the synchronization structure of concurrent progreams. *Acta Informatica*, 19(1):57–84, April 1983.

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

[46] F. J. Torres-Rojas, M. Ahamad, and M. Raynal. Timed consistency for shared distributed objects. In *Proc. of the 18th ACM Symposium on Principles of Distributed Computing (PODC)*, pages 163–172, May 1999.

[47] W. Vogels. Eventually consistent. *Communications of the ACM*, 52(1):40–44, January 2009.

[48] Voldemort. Available at http://project-voldemort.com/.

[49] H. Wada, A. Fekete, L. Zhao, K. Lee, and 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)*, pages 134–143, January 2011.

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

[51] K. Zellag and B. Kemme. How consistent is your cloud application? In *Proc. of the Third ACM Symposium on Cloud Computing (SoCC)*, page 6, 2012.