Achieving Probabilistic Atomicity With Well-Bounded Staleness and Low Read Latency in Distributed Datastores

Lingzhi Ouyang, Yu Huang, Hengfeng Wei, and Jian Lu

Abstract—Although it has been commercially successful to deploy weakly consistent but highly-responsive distributed datastores, the tension between developing complex applications and obtaining only weak consistency guarantees becomes more and more severe. The almost strong consistency tradeoff aims at achieving both strong consistency and low latency in the common case. In distributed storage systems, we investigate the generic notion of almost strong consistency in terms of designing fast read algorithms while guaranteeing Probabilistic Atomicity with well-Bounded staleness (PAB). This problem has been explored in the case where only one client can write the data. However, the more general case where multiple clients can write the data has not been studied. In this article, we study the fast read algorithm for PAB in the multi-writer case. We show the bound of data staleness and the probability of atomicity violation by decomposing inconsistent reads into the read inversion and the write inversion patterns. We implement the fast read algorithm and evaluate the consistency-latency tradeoffs based on the instrumentation of Cassandra and the YCSB benchmark framework. The theoretical analysis and the experimental evaluations show that our fast read algorithm guarantees PAB, even when faced with dynamic changes in the computing environment.

Index Terms—Probabilistic atomicity, well-bounded staleness, fast read algorithm, quorum-replicated datastore

1 Introduction

Nowadays cloud-based distributed datastores are expected to provide always-available and highly responsive services for millions of user requests across the world [1], [2], [3]. To this end, data replication is typically employed. By replicating data into multiple replicas across different machines or even across data centers, distributed datastores can not only reduce response time of user requests, but also tolerate certain degree of software/hardware failures and network partitions [4], [5]. Since cloud-based datastores must tolerate network partitions, according to the CAP theorem, once the datastore replicates data, the tradeoff between data consistency and data access latency comes up [6], [7]. Many real-world web services, such as Google, Amazon, EBay, etc., aim to provide an "always-on" experience and overwhelmingly favor availability and low latency over strong consistency [8]. It is claimed that a slight increase in user-perceived latency translates into concrete revenue loss [9].

Although it is widely used and commercially successful to deploy weakly consistent but highly responsive replicated datastores, the tension between developing complex upper-layer user applications and obtaining only weak consistency guarantees becomes more and more severe [9], [10]. Provided with only weak consistency guarantees such as eventual consistency [11] and session guarantee [12], the application developers suffer from anomalies in reading stale data and difficulties in resolving conflicting updates into common states [13]. The developers expect strong consistency guarantees, which shield them from the underlying reality of large-scale distributed systems and provide the illusion of sequential programming on one single copy of local data [14], [15].

The *almost strong consistency* tradeoff aims at helping the application developers out of the dilemma of choosing either low latency or strong consistency [16]. The tradeoff achieves the best of both worlds: strong consistency and low latency in the common case. As for data access latency, the almost strong consistency tradeoff adopts "fast" data access algorithms, i.e., algorithms requiring one single round-trip of communication between the clients and the server replicas [17]. This is obviously optimal in terms of data access latency. As for data consistency, fast data access protocols cannot strictly guarantee strong consistency according to the impossibility results [6], [7], [18]. However, the impossibility results can be circumvented by the concept of "almost strong" consistency. Here, the abstract term "almost" can be interpreted in two orthogonal dimensions. On the one hand, "almost strong" means that the data accessed can be stale, but the staleness should be well-bounded. On the other hand, the probability of accessing stale data should be quite small. Combining both dimensions, clients are able to access highly consistent data in most circumstances.

We illustrate the basic idea of the almost strong consistency tradeoff with several examples. In a logistics management

• *The authors are with the State Key Laboratory for Novel Software Technology, and Department of Computer Science and Technology, Nanjing University, 210023, China. E-mail: lingzhi.ouyang@outlook.com, {yuhuang, hfwei, lj}@nju.edu.cn.*

Manuscript received 26 Mar. 2020; revised 17 Sept. 2020; accepted 16 Oct. 2020. Date of publication 28 Oct. 2020; date of current version 10 Nov. 2020. (Corresponding author: Yu Huang.) Recommended for acceptance by W. Yu. Digital Object Identifier no. 10.1109/TPDS.2020.3034328

scenario, data items are naturally organized centering around the commodities. During the shipment, different divisions of the commodity transportation system may update the location of the commodity. Meanwhile, the inventory tracking system and the customers all need to frequently query status of the commodities. Though data consistency is a desirable property, the users (i.e., all types of entities which need to query status of the commodity) are more concerned about how long they have to wait before the queries can be served. Thus, the users may be willing to trade certain data consistency for low latency, as long as the inconsistency is bounded and the application can still access highly consistent data most of the time [16], [19]. In collaborative working scenarios, cloud storage system can also be used to support a shared working space. The quick responses to user inputs are of primary importance in collaborative working scenarios. However, to enable concurrent but meaningful updates to the shared working space, data inconsistency should be bounded and be rare [20], [21].

The almost strong consistency tradeoff can also serve as a new option for the consistency SLA of cloud storage services. For example, Amazon DynamoDB [1] provides either eventually or strongly consistent reads for users. The almost strong consistency provides a new option for users who want to read data with low latency as well as achieve better user experience than eventual consistency. Microsoft's Azure Cosmos DB [22], [23] provides a consistency option of bounded staleness, which bounds the staleness of version and timeliness. The almost strong consistency tradeoff can enrich the quantification dimension with probability.

In distributed (key-value) storage systems, we investigate the generic notion of almost strong consistency in terms of *Probabilistic Atomicity with well-Bounded staleness (PAB)* [1]. We instantiate the notion of strong consistency as atomicity, since it is the de facto correctness criterion for shared data [14], [15], [24]. Atomicity is an ideally strong consistency condition, requiring the execution to be equivalent to a legal sequential execution. However, it has been theoretically proved that atomicity generally does not admit low-latency implementations. Guaranteeing low latency in the first place, PAB circumvents the impossibility result by limiting the staleness of data and the probability of accessing stale data.

The notion of PAB has been explored in the *single-writer* case, where only one single client can write the data while multiple clients can read the data in our previous work [16]. In the single-writer case, it is shown that, when the clients read data using only one round-trip of communication with the server replicas, the clients can miss at most one data update (i.e., achieve 2-atomicity according to the definition of [25]) and the probability of reading stale data is quite small. However, the more general *multi-writer* case where multiple clients can write and read the data has not been studied. Moreover, the fast read algorithm for PAB in the single-writer case is only evaluated in a mobile data sharing scenario. The more important scenario of cloud-based replicated datastores has not been studied experimentally.

In this work, we investigate fast (i.e., one round-trip) algorithms which can guarantee PAB for multi-writer replicated data objects. We model the client-server interaction as a quorum system [26], [27]. The latency of data reads and writes are mainly decided by the number of round-trips of communications between the clients and the servers. Since two round-trips of both read and write operations are sufficient to strictly guarantee atomicity [28], we can tune the read, the write or both operations to one round-trip in order to reduce the read and write latency. We find, by both theoretical analysis and experimental evaluations, that the write operation must employ two round-trips of communications. Otherwise, the data inconsistency cannot be bounded from the perspectives of data staleness and the probability of accessing stale data (see detailed discussions in Section 3 in the appendix [29] [2]).

According to the discussions above, we focus on the *fast read* (i.e., one round-trip read and two round-trip write) algorithm in this work. In the logistics management scenario, the inventory tracking system may persistently and periodically query the status of the commodities. As long as the inconsistent read has bounded staleness and appears with limited probability, the tracking system can obtain up-to-date data most of the time, and obtain statistically accurate data. The tracking system is willing to sacrifice certain amount of consistency for low latency. Also, when multiple divisions of the transportation system update the data, the updates are much less frequent than the reads. The write latency is acceptable, especially when the two round-trip write is essential to enabling the (much more frequent) highly consistent read. In the collaborative working scenario, multi-writer shared data objects can be used to indicate the status of shared resources. They can also be used to support more complex coordination. For example, in the mutual exclusion algorithm, a weakly consistent multi-writer register can be used to indicate which one currently has the priority of entering the zone ahead of other competitors [30], [31]. Low latency in collaborative work scenarios is of primary importance, while bounded staleness greatly facilitate meaningful collaboration.

In order to explore whether the fast read algorithm can guarantee PAB, we first study through theoretical analysis the data staleness and the probability of inconsistent reads. As inspired by the theoretical analysis in the single-writer case, we first express data inconsistency as the atomicity violation pattern among read and write operations in the space-time diagram of shared data access. The violation pattern is further decomposed into two sub-patterns namely the *Write Inversion (WI)* and the *Read Inversion (RI)*. By the case-by-case analysis we prove that WI and RI will be incurred whenever an inconsistent read occurs.

Then we employ the WI and RI patterns to analyze the bound of data staleness and the probability of reading stale data. We derive the tight upper bound of data staleness by intentionally constructing executions with the maximum number of writes a read can possibly miss. We obtain the

1. Our use of "atomicity" concerns the correctness of concurrent objects. Note that it is different from the meaning of the all-or-none property in transactions in the database community. Linearizability [15] is equivalent to atomicity when restricted to read/write registers.

2. The appendix is provided in a separate supplementary file, which can be found on the Computer Society Digital Library at http://doi.ieeecomputersociety.org/10.1109/TPDS.2020.3034328, as required by the submission procedure. The appendix is also available online [29].

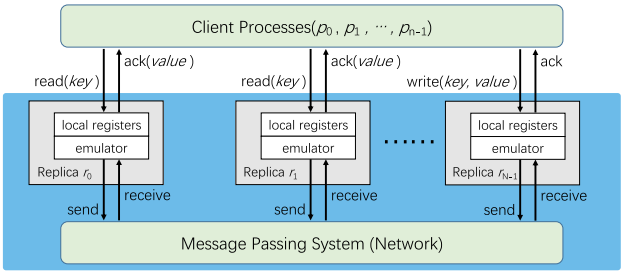

Fig. 1. System model of read/write register emulation.

upper bound of probability of reading stale data by bounding the probabilities of RI and WI. The theoretical analysis, together with the corresponding numerical results, show that the fast read algorithm can guarantee PAB in distributed storage systems.

We implement a distributed storage system as well as the fast read algorithm, in order to study whether PAB can be guaranteed via comprehensive experiments. The implementations are mainly based on instrumentation over the open-source datastore Cassandra [32]. We mainly reuse the quorum-replication framework and the data management modules in Cassandra. The YCSB benchmarking framework [33], [34] is used to generate various workloads, and we simulate changes of various environment factors for comprehensive experimental evaluations. The evaluation results further demonstrate that PAB can be guaranteed by fast read algorithms in replicated datastores, even when faced with dynamic changes in the computing environment. In the experiments, we also study the effects of practical system optimizations which are beyond accurate theoretical analysis.

The rest of this paper is organized as follows. Section 2 presents the quorum-based algorithm schema of multi-writer data objects. Sections 3 and 4 present the theoretical analysis and the experimental evaluations of the fast read algorithm respectively. Section 5 reviews the existing work. In Section 6, we conclude this work and discuss the future work.

2 Quorum-Based Algorithm Schema

In this section we first present the system model and the consistency model. Then we propose the quorum-based algorithm schema of distributed shared data objects. The schema enables us to thoroughly explore possible design options of fast algorithms potentially guaranteeing probabilistic atomicity with well-bounded staleness.

2.1 System Model

The replicated datastore consists of N *servers* that communicate with each other through point-to-point message communication. Each server stores one replica of the data item, and all the replicas collectively emulate the *shared register* abstraction for the *clients*, as shown in Fig. 1. The shared register is identified by its *key* and can be accessed through the *read* ($value \leftarrow read(key)$) and *write* ($write(key, value)$) operations. Since the data is replicated and can be updated concurrently, multiple versions of logically the same data may coexist. We assume the asynchronous non-Byzantine model, where messages can be delayed, lost or delivered out of order due to process or link failures, but they will not be corrupted or duplicated. Besides, any number of clients and any minority of servers (less than $\frac{N}{2}$) may crash at any moment.

2.2 Consistency Model

We define an *execution history* (or *history* for short) of the clients accessing the shared register as a sequence of events where each event is either the invocation or the response of a read or write operation. As for shared registers that appear in the history, we assume that all operations in the history are applied to the same register. Note that this assumption is not restrictive. The consistency models considered in this work (atomicity and k-atomicity defined below) are local, i.e., for a history with multiple shared registers, it is atomic/k-atomic if and only if for each register accessed, the sub-history is atomic/k-atomic [35]. Given the locality of the consistency models, we can manage each data item independently in the replicated datastore.

Each event in the history is tagged with a unique time, and events appear in the history in increasing order of their timestamps. For a history σ, we can define the partial order between operations. Let $o.s$ and $o.f$ denote the timestamps of the invocation and the response events of operation o respectively. We define $o_1 \prec_\sigma o_2$ if $o_1.f < o_2.s$. We define $o_1 \parallel_\sigma o_2$ if neither $o_1 \prec_\sigma o_2$ nor $o_2 \prec_\sigma o_1$ holds. A history σ is *sequential* if σ begins with an invocation, and each invocation is immediately followed by its matching response. A history σ is *well-formed* if for each client p_i, $\sigma|p_i$ (the subsequence of σ restricted on p_i) is sequential. Given the notations above, we can define atomicity:

Definition 2.1. *A replicated datastore satisfies* **atomicity** *if, for each of its well-formed histories σ, there exists a permutation π of all operations in σ such that π is sequential and satisfying the following two requirements:*

- ***[Real-time requirement]*** *If $o_1 \prec_\sigma o_2$, then o_1 appears before o_2 in π.*
- ***[Read-from requirement]*** *Each read returns the value written by the latest preceding write in π.*

For the sake of defining almost strong consistency in the data staleness dimension, we generalize the definition of atomicity to k-atomicity [35] by generalizing the *read-from* requirement:

Definition 2.2. *A replicated datastore satisfies* **k-atomicity** *($k \in \mathbb{Z}^+$) if, for each of its well-formed histories σ, there exists a permutation π of all the operations in σ such that π is sequential and satisfying the following two requirements:*

- ***[Real-time requirement]*** *If $o_1 \prec_\sigma o_2$, then o_1 appears before o_2 in π.*
- ***[Parameterized read-from requirement]*** *Each read returns the value written by one of the latest k preceding writes in π.*

Given the definitions above, it is obvious to see that atomicity is equivalent to 1-atomicity. Besides, if a history σ satisfies k-atomicity, then $\forall k' > k$, σ satisfies k'-atomicity ($k, k' \in \mathbb{Z}^+$). In the following sections, when we mention σ is k-atomic, we refer to the minimum k in terms of k-atomicity that σ satisfies.

2.3 The Algorithm Schema of Read/Write Register Emulation

We first present the basic primitives for client-server interaction. Based on tuning the round-trips of client-server interaction, we propose the algorithm schema that involves possible options of fast algorithms. We analyze the consistency guarantees provided by the algorithms in the schema, which are essential to the theoretical analysis in Section 3.

2.3.1 The "Diamond" Schema and 4 Concrete Algorithms

The clients can obtain data updates from server replicas via the *query* operation, and can modify the replicas via the *update* operation. When a writer client updates the data, a sequence number paired with the id of the writer client, forming the version $ver = (seq, id)$, will be attached to the data. Note that the version values are unique and totally ordered according to the lexicographical order, as in [28]. Upon receiving a query request from a client, the server replies to the client with the data. Upon receiving an update request, the server updates its replica if the data from the client is attached with a larger version and replies to the client with an ACK. The pseudo-code is presented in Algorithm 2.3.1. Note that the client-server interaction described here is abstract. Often it is implemented in such a way that the client contacts one replica, and this replica contacts other replicas and replies to the client on behalf of all these replicas, as in Cassandra [3], [32].

Algorithm 1. Client-server interaction

```
1  ▷ Code for client process p_i(0 ≤ i ≤ n − 1):;
2  function(query(key))
3    vals ← ∅;
4    pfor each server s_j ▷ pfor: parallel for)
5      send ['query', key] to s_j;
6      v ← [key, val, ver] from s_j;
7      vals ← vals ∪ v;
8    (until a majority of them respond)
9    returnvals;
10 function(update(key, value, version));
11   vals ← ∅;
12   pfor each (server s_j)
13     send ['update', key, value, version] to s_j;
14   wait for (['ACK']s from a majority of them)
15 ▷ Code for server process s_i(0 ≤ i ≤ N − 1):
16 upon(receive ['query', key] from p_j)
17   send ['query − back', key, val, ver] to p_j
18 upon(receive ['update', key, value, version] from p_j)
19   pick [k, val, ver] with k == key;
20   if ver < version then
21     val ← value;
22     ver ← version;
23   send ['ACK'] to p_j
```

The interaction between the clients and servers can be captured by a quorum system [26], [27]. Viewing a quorum system from the space dimension, clients need to contact multiple replicas to perform a query or an update, and the set of replicas contacted each time is called a query quorum or an update quorum respectively. All the query and update quorums form a quorum system if any two quorums have non-empty intersection. The intersection between quorums enables data updates to be propagated among the clients and the server replicas. In this work we adopt the simple but efficient majority quorum system, where each query or update quorum contains a majority (more than $\frac{N}{2}$) of replicas. Viewing the quorum system from the time dimension, clients may communicate with the replicas via one or more round-trips of communications. The number of round-trips is the most important factor deciding data access latency.

Given the primitives for client-server interaction, the replicas can collectively emulate the *write* and the *read* operations for clients, as shown in Algorithm 2.3.1 and 2.3.1. To emulate a multi-writer atomic register, both the write and the read operations require two round-trips of client-server interaction [28]. As for the write operation, the client first collects versions from a majority of replicas in the first round-trip. Then, it constructs a new version by increasing the sequence number of the largest returned version by 1, and replacing the id with its own id. In the second round-trip, the client updates the data together with the new version to a majority of replicas. As for the read operation, the client first collects data from a majority of replicas and select the data with the largest version. Then, the client employs an additional round-trip to write-back the data into a majority of replicas.

Algorithm 2. Write algorithms for client p_i

```
1 procedure(TwoRoundWRITE(key, value))
2   replicas ← querykey;
3   version ←( maxSeq (replicas)+1, i);
4   updatekey, value, version;
5 procedure(OneRoundWRITE(key, value))
6   localSeq[key] ← localSeq[key]+1;
7   version ← (localSeq[key], i);
8   updatekey, value, version;
```

Algorithm 3. Read algorithms for client p_i

```
1  procedure(TwoRoundREAD (key))
2    replicas ← query(key);
3    version ← maxVer(replicas);
4    value ← valWithMaxVer (replicas, version);
5    update(key, value, version);
6    returnvalue;
7  procedure( OneRoundREAD (key))
8    replicas ← query(key);
9    version ← maxVer(replicas);
10   value ← valWithMaxVer (replicas, version);
11   return value;
```

The two round-trips of both write and read operations are able to strictly guarantee atomicity. Bearing *Probabilistic Atomicity with Bounded-staleness* in mind, we can tune the write and/or read operations to one single round-trip in order to reduce latency. Specifically, the one round-trip write algorithm may omit the first round-trip of querying versions from replicas and directly update the replicas with a version constructed locally. The one round-trip read algorithm may omit the second round-trip of writing-back data and directly return the queried data that has the largest version.

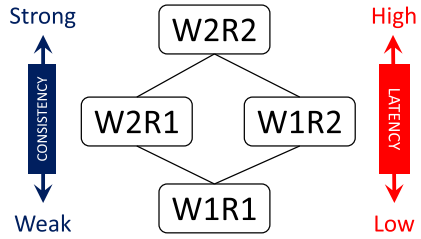

Fig. 2. Diamond schema of shared register emulation algorithms.

By tuning the number of round-trips of the write and/or the read operations, we can obtain four variants - namely W2R2, W2R1, W1R2 and W1R1 - of read/write register emulation algorithms. The W2R2 algorithm employs two round-trips for both read and write operations, and other three algorithms are named in a similar way. The four algorithms form a "diamond" lattice as shown in Fig. 2. The lattice in the figure can be viewed as the Hasse diagram of the four algorithms. The algorithms in the upper layer provide stronger consistency guarantees than those in the lower layer, which is discussed in detail in the following Section 2.3.2.

2.3.2 Consistency Guarantees Provided by the Four Algorithms

In general, the more round-trips employed in the write or read algorithms, the more consistency guarantees provided. To define a fine-grained metric for measuring different levels of consistency guarantees, we explore the *monotonicity* properties between read and write operations. Basically, monotonicity means that when the operations have certain temporal order in the history, they will have certain semantic order concerning the version values of replicas. For any two operations, each could be either a write or a read. Thus we have four combinations, and define four types of monotonicity. We study which types of monotonicity that the four algorithms under the diamond schema can guarantee, as shown in Table 1.

Write-read monotonicity. All four algorithms have the write-read monotonicity that $w \prec_\sigma r \Rightarrow ver(w) \leq ver(r)$. This is guaranteed by the intersection requirement of quorum systems. When a read starts after a write has finished, the read will contact at least one replica which has been modified by the write due to the quorum intersection property. Thus, the version obtained by the read is at least as large as that of the preceding write.

Write-write monotonicity. As for the monotonicity between writes for multi-writer registers, $w \prec_\sigma w' \Rightarrow ver(w) < ver(w')$ can be guaranteed only by the two round-trip write algorithms (i.e., W2R2 and W2R1). First employing one round-trip to obtain the largest version from a majority of replicas is key to avoiding assigning a version smaller than those of the previous writes.

Read-Read Monotonicity. As for the monotonicity between reads for multi-reader registers, $r \prec_\sigma r' \Rightarrow ver(r) \leq ver(r')$ can be guaranteed only by the two round-trip read algorithms (i.e., W2R2 and W1R2). The write-back process in the second round-trip is key to preventing later reads from returning more stale values.

Read-Write Monotonicity. As for the property $r \prec_\sigma w \Rightarrow ver(r) < ver(w)$, it can be guaranteed by the W2R2 algorithm only. As for the preceding read, the second round-trip writes back the newly acquired data. For the write, the first round-trip will first queries the replicas. The quorum intersection property guarantees that the version of the read is smaller than that of the write. If either the write or the read employs only one round-trip, this property cannot be guaranteed.

TABLE 1
Four Types of Monotonicity of the Four Algorithms

Properties	W2R2	W2R1	W1R2	W1R1
$w \prec_\sigma r \Rightarrow ver(w) \leq ver(r)$	✓	✓	✓	✓
$w \prec_\sigma w' \Rightarrow ver(w) < ver(w')$	✓	✓	×	×
$r \prec_\sigma r' \Rightarrow ver(r) \leq ver(r')$	✓	×	✓	×
$r \prec_\sigma w \Rightarrow ver(r) < ver(w)$	✓	×	×	×

The detailed proof of all the properties above can be found in Section 1 in the appendix [29] available in the online supplemental material. In the following Section 3, we study the consistency guarantees provided by algorithms following the diamond schema. We mainly discuss the fast read algorithm (i.e., the W2R1 algorithm), which guarantees probabilistic atomicity with well-bounded staleness. The W1R2 and W1R1 algorithms are briefly discussed since they cannot provide sufficient consistency guarantees.

3 Consistency Guarantees of the Fast Read Algorithm: Data Staleness and Violation Probability

In this section, we mainly analyze the consistency guarantees provided by the fast read, i.e., W2R1, algorithm. The keys to our analysis are the patterns named *read inversion* and *write inversion*. We first transform all inconsistent reads into these two patterns. Then we leverage these two patterns to analyze the bound of data staleness and the probability of atomicity violation. After the analysis of the W2R1 algorithm, we also briefly discuss the other algorithms in the diamond algorithm schema.

3.1 Atomicity Violation Patterns

In the one round-trip read operation of W2R1, the absence of the *write-back* phase before returning the data to the client may violate the read-read monotonicity and the read-write monotonicity, as shown in Table 1. According to the violation of these two monotonicity properties, we can define two essential patterns of inversions accordingly:

Definition 3.1 (Read Inversion). *The Read Inversion after a read (RI) involves two reads* r, r' *satisfying* $(r \prec_\sigma r') \wedge (ver(r) > ver(r'))$.

Definition 3.2 (Write Inversion). *The Write Inversion after a read (WI) involves a read* r *and a write* w *satisfying* $(r \prec_\sigma w) \wedge (ver(r) > ver(w))$.

The RI and WI patterns are essential to further analysis of both the bound of data staleness and the probability of atomicity violation, which is depicted by the following Theorem 3.1. When the clients read and write a Multi-Writer Multi-Reader (MWMR) register using the W2R1 algorithm and obtain a history σ, we have that:

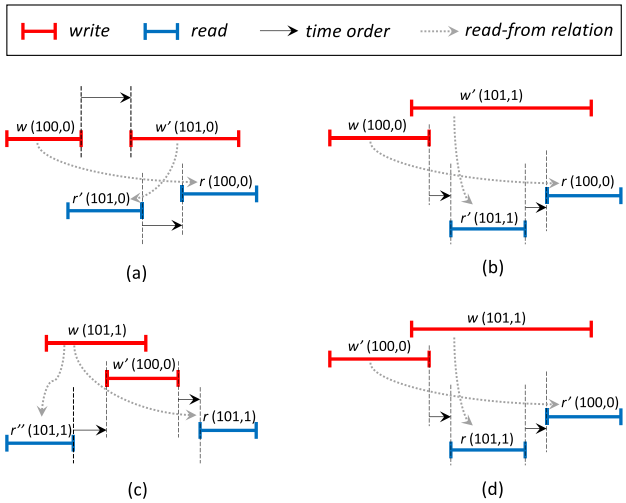

Fig. 3. Typical patterns of RI in Case 1 (Sub-fig (a) and (b)), and of WI (Sub-fig (c)) and RI (Sub-fig (d)) in Case 2.

Theorem 3.1. *If σ violates atomicity, then there exist some operations in σ that form either RI or WI.*

If σ violates atomicity, then for any permutation π of σ, we have a stale read r, the dictating write w of r and the interfering write w' satisfying: $w \prec_\pi w' \prec_\pi r$. To see why we inevitably have RI or WI, we exhaustively check all possible cases.

According to the definition of atomicity, two types of relations between operations are of our concern: the temporal real-time relation and the semantic read-from relation. We first enumerate all possible cases according to the semantic relation. Then for each case, we further enumerate all possible sub-cases according to the temporal relation.

Since w is the dictating write of r, we have that $ver(r) = ver(w)$. According to the semantic relation between versions of w and w', we have two complementing cases:

- Case 1: $ver(r) = ver(w) < ver(w')$.
- Case 2: $ver(r) = ver(w) > ver(w')$.

In each case, we then consider the temporal relation between operations, as shown in Fig. 3.

We discuss the basic rationale of the proofs of Case 1 and Case 2 in the following Section 3.2.1 and 3.2.2 respectively. Detailed proof of the theorem, which exhaustively checks all possible cases and constructs the RI or WI, is provided in Section 2.1 in the appendix [29] available in the online supplemental material.

3.1.1 *Case 1:* $ver(r) = ver(w) < ver(w')$

In Case 1, we further consider the temporal relation between w and w'. Since in the permutation π, $w \prec_\pi w'$, we have that in σ, $w \prec_\sigma w'$ or $w||_\sigma w'$, as shown in Figs. 3a and 3b respectively.

Note that r must be concurrent with w'. This can be proved by contradiction. If $r \prec_\sigma w'$, w' can never be the interfering write of r. If $w' \prec_\sigma r$, according to the write-read monotonicity (see Table 1), r must return the version of w' (or return an even larger version), contradicting the fact that $ver(r) = ver(w) < ver(w')$.

When linearly extending σ into π, we inevitably have $w \prec_\pi w' \prec_\pi r$. For concurrent operations w' and r, in order to "force" w' to appear before r in π, we must have a read operation r' dictated by w', satisfying $r' \prec_\sigma r$. Thus we construct a certificate of RI, i.e., $r' \prec_\sigma r$, but $ver(r') > ver(r)$.

3.1.2 *Case 2:* $ver(r) = ver(w) > ver(w')$

Since $ver(r) = ver(w) > ver(w')$, w must be concurrent with w'. This can also be proved by contradiction. If $w \prec_\sigma w'$, according to the write-write monotonicity (see Table 1), the version of w' must be larger. If $w' \prec_\sigma w$, we can never get $w \prec_\pi w' \prec_\pi r$ when linearly extending σ into π.

Given that w and w' are concurrent, we must "force" w' to appear after w in π. This can be achieved in several different ways.

In one subcase, we have $w' \prec_\sigma r$, as well as a read r'' dictated by w, satisfying $r'' \prec_\sigma w'$, as shown in Fig. 3c. Thus, we inevitably get $w \prec_\pi r'' \prec_\pi w' \prec_\pi r$ when linearly extending σ into π. In this subcase, we have a certificate of WI, i.e., $r'' \prec_\sigma w'$, but $ver(r'') > ver(w')$.

In another subcase, we have a read r' dictated by w', satisfying $w' \prec_\sigma r \prec_\sigma r'$, as shown in Fig. 3d. Besides, there exists no dictated read of w' that precedes w, so w' may appear after w in π and be the interfering write of r. Here, we have a certificate of RI, i.e., $r \prec_\sigma r'$, but $ver(r) > ver(r')$.

3.2 Bound of Data Staleness

In this section, we calculate the tight bound of data staleness when accessing a MWMR register using the W2R1 algorithm. Note that the operations are asynchronous. Denote the number of writer clients as n_w. Then we have that:

Theorem 3.2. *For any history σ, there exists a linear extension π of σ such that any read in π returns the value written by one of the latest B preceding writes. Here, $B = n_w + \frac{1}{2}n_w(n_w - 1) + 1$. Moreover, the bound B is tight, i.e., there exists a history σ and a linear extension π of σ in which some read returns the value written by the oldest write in the latest B preceding writes.*

The proof of Theorem 3.2 is based on an adversary argument: to insert as many interfering writes as possible for an inconsistent read. The proof also needs to consider the same two cases as defined in Section 3.1. Specifically, suppose r is an inconsistent read, and the dictating write of r is w. There must exist another interfering write w' such that $w \prec_\pi w' \prec_\pi r$.

According to the versions $ver(r)$ and $ver(w')$, we consider two cases. In Case 1 where $ver(r) = ver(w) < ver(w')$, we prove that there are at most $B_1 = n_w$ interfering writes which can inevitably be inserted between r and w in π. In Case 2 where $ver(r) = ver(w) > ver(w')$, we prove that there are at most $B_2 = \frac{1}{2}n_w(n_w - 1)$ interfering writes.

Since these two cases can appear at the same time in the history, we can construct a trace with $B_1 + B_2$ interfering writes. Counting the dictating write itself, we have that the read always returns the value written by one of the latest $B = B_1 + B_2 + 1$ preceding writes in π, i.e., the history σ always satisfies B-atomicity (in terms of the k-atomicity model [35]). During the proof, we explicitly construct the worst-case trace, which contains the read having $B_1 + B_2$ interfering writes. This proves that the bound is tight.

In the following Sections 3.2.1 and 3.2.2, we derive the bound B_1 and B_2 respectively. Detailed proof of the bound

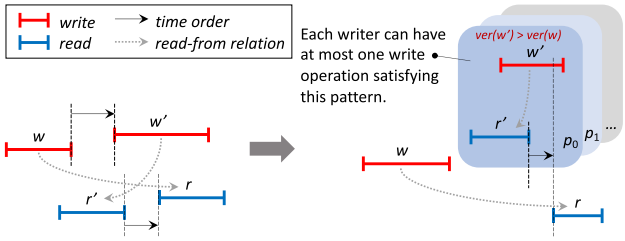

Fig. 4. Maximum number of interfering writes in Case 1.

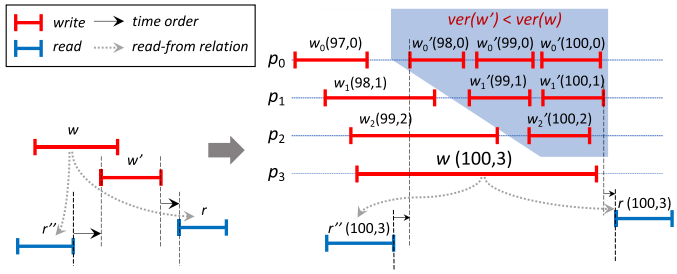

Fig. 5. Maximum number of interfering writes in Case 2.

B can be found in Section 2.2 in the appendix [29] available in the online supplemental material.

3.2.1 Bound of Staleness in Case 1 ($ver(r) < ver(w')$)

In this case, the pattern of operations which enables our construction of the trace with the most stale read is the pattern shown in Fig. 3a. The construction is illustrated in Fig. 4.

From the proof of Case 1 in Proposition 3.1, the interfering write w' must be concurrent with r, and w' dictates a read r' that precedes r. Note that the start time of w' must be earlier than that of r. Otherwise, when r' precedes r, r' will not have the chance to read from w'. Since each writer client can have at most one write operation that not only starts before the invocation of r but also stays concurrent with r, we could force each of the n_w writers (including the writer of w itself) to issue one interfering write w' separately. Thus we have the bound $B_1 = n_w$ in this case, and the bound B_1 is tight.

3.2.2 Bound of Staleness in Case 2 ($ver(r) > ver(w')$)

In this case, the pattern of operations which enables our construction of the worst-case trace is the WI pattern, as shown in Fig. 3c.

From the WI proof of Case 2 in Proposition 3.1, the interfering write w' must be concurrent with w, and w' not only starts after the finish time of a read r'' dictated by w, but also ends before the start time of r. Other than the writer of operation w, we can have at most $n_w - 1$ writers to generate interfering writes.

One challenge in the construction is that, some writers may have more than one interfering write. This is due to, when r'' is finished but w is still in process, the version $ver(w)$ may have not yet been propagated to a majority of replicas, so there may still exist a majority of replicas whose maximum version is smaller than $ver(w)$. For adversary argument, before w is finished, let the maximum version of a majority of replicas stay as small as possible. Then, other writer clients may have more than one chance to query a majority of replicas whose maximum version is smaller than $ver(w)$ and then write with an incremental version that is still smaller than $ver(w)$.

However, not all writes versioned smaller than $ver(w)$ can appear after the finish time of r''. Specifically, w will not have $ver(w) = (seq, id)$ unless in the first round-trip it queries a majority of replicas whose maximum version is $(seq - 1, i)$, which is written by the client p_i by assumption. Then, after the finish time of r'', p_i can only write with the version larger than $(seq - 1, i)$.

For the ease of illustration, we use a concrete example as shown in Fig. 5. Assume $n_w = 4$, and the writers are p_0, p_1, p_2 and p_3. Let the operation w issued by p_3 have the version $(100, 3)$. Client p_3 can write with this version only when it queries a majority of replicas whose maximum version is $(99, i)$, where i can be any writer's identifier. Let p_2 have the write operation with the version $(99, 2)$. Similarly, let p_1 have the write operation versioned $(98, 1)$, so p_2 can write with the version $(99, 2)$; let p_0 have the write operation versioned $(97, 0)$, so p_1 can write with the version $(98, 1)$.

Assume r'' reads from w versioned $(100, 3)$. Since r'' will not write back the acquired data into replicas, when r'' is finished there may still exist a majority of replicas whose maximum version is smaller than $(100, 3)$. Now we focus on what could happen after that. Assume that when p_0 finishes the write operation versioned $(97, 0)$, none of p_1, p_2 or p_3 has finished the write operation versioned $(98, 1)$, $(99, 2)$ or $(100, 3)$ respectively. Thus, p_0 may query a majority of replicas whose maximum version is $(97, 0)$, and then issue an interfering write operation versioned $(98, 0)$ and later $(99, 0)$ and $(100, 0)$. Similarly, assume that when p_1 finishes the write operation versioned $(98, 1)$, none of the write operations versioned larger than $(98, 1)$ has been finished. Then, a majority of replicas may still hold the maximum version $(98, 1)$, so p_1 may have the chance to issue an interfering write operation versioned $(99, 1)$ and later $(100, 1)$. As for p_2, after it finishes the write versioned $(99, 2)$ but no write operation versioned larger than $(99, 2)$ has been finished, a majority of replicas may still hold the maximum version $(99, 2)$. Thus p_2 may have the chance to make an interfering write operation versioned $(100, 2)$.

Note that all these interfering writes appear not only after the finish time of r'' but also before the start time of r, thus they have to be placed after w and before r in π, inevitably being the interfering writes of r.

Generalize the typical example above, we have that for all the $n_w - 1$ interfering writer clients, we can insert:

$$B_2 = 1 + 2 + \ldots + (n_w - 2) + (n_w - 1) = \frac{1}{2} n_w (n_w - 1),$$

interfering writes.

3.3 Probability of Atomicity Violation

The probability calculation of atomicity violation is also based on decomposing the atomicity violation into the RI and WI patterns. To further calculate the probabilities of RI and WI, we reduce the calculation in the multi-writer case to that in the single-writer case.

First note that the occurrence of either RI or WI is the necessary but not sufficient condition for atomicity violation (as shown in Fig. 6). Nevertheless, this will not prevent us from obtaining the upper bound of the violation probability:

Fig. 6. The relation between atomicity violation and the occurrences of RI and WI.

$$\begin{aligned}\mathbb{P}\{\text{Violation}\} &\leq \mathbb{P}\{\text{RI} \vee \text{WI}\} \\ &= \mathbb{P}\{\text{RI}\} + \mathbb{P}\{\text{WI}\} - \mathbb{P}\{\text{RI} \wedge \text{WI}\} \\ &\leq \mathbb{P}\{\text{RI}\} + \mathbb{P}\{\text{WI}\}\end{aligned} \tag{3.1}$$

To calculate the probabilities of RI and WI, we view RI and WI as the concurrent occurrences of multiple atomicity violation patterns in the single-writer case. According to our previous work [16], atomicity violation in the single-writer case is equivalent to the pattern named Old-New Inversion (ONI). We further decompose the ONI into the temporal Concurrency Pattern (CP) and the semantic Read Write Pattern (RWP). Then we obtain the probability of ONI:

$$\begin{aligned}\mathbb{P}\{\text{ONI}\} &= \sum_{m\geq 1} \mathbb{P}\{\text{CP} \mid R' = m\} \cdot \mathbb{P}\{\text{RWP} \mid R' = m\} \\ &\approx \sum_{m=1}^{n_r} \left(\left(\sum_{k=0}^{n_r-1} \binom{n_r}{k} \binom{m-1}{n_r-k-1} p_0^k r^{n_r-k} s^m \right) \right. \\ &\quad \cdot e^{-q\lambda_w t} \frac{\alpha^q B(q, \alpha(N-q)+1)}{B(q, N-q+1)} \\ &\quad \left. \cdot \left(1 - \left(\frac{J_1}{B(q, N-q+1)}\right)^m\right)\right)\end{aligned} \tag{3.2}$$

where R' is a random variable denoting the number of reads r' in Definition 3.1. Please refer to our previous work [16] for detailed explanations of Equation (2) [3].

From the proof of Theorem 3.1, we can decompose RI and WI incurred in W2R1 into CP and RWP in a similar way:

Definition 3.3 (The Decomposition of RI). *The RI in W2R1 involves one write w' and two reads r, r', satisfying*

- *the long-lived-write concurrency pattern(CP):*
  1) $r_{st} \in [w'_{st}, w'_{ft}]$ [4],
  2) $r'_{ft} \in [w'_{st}, r_{st}]$;
- *the non-monotonic read-write pattern(RWP):*
  1) $ver(r') = ver(w')$;
  2) $ver(r) < ver(w')$.

Definition 3.4 (The Decomposition of WI). *The WI in W2R1 involves two writes w, w' and one read r'', satisfying*

- *the long-lived-write concurrency pattern(CP):*
  1) $w'_{st} \in [w_{st}, w_{ft}]$,
  2) $r''_{ft} \in [w_{st}, w'_{st}]$;
- *the non-monotonic read-write pattern(RWP):*
  3) $ver(r'') = ver(w)$;
  4) $ver(w') < ver(w)$.

3. The derivation of Equation (2) is provided in Section 4 of our previous work [16], which is also available online at: https://github.com/Lingzhi-Ouyang/Almost-Strong-Consistency-Cassandra/blob/master/document/pa2ac.pdf.

4. The variables o_{st} and o_{ft} refer to the start time and the finish time of the operation o respectively .

Given the analysis of ONI in the single-writer case, we can see that ONI is a special case of RI when there exists only one writer client. By mapping w', w, r'' in Definition 3.4 to r, w', r' in Definition 3.3 respectively, WI can also be modeled as ONI.

By modeling the occurrences of multiple writers, we can calculate the probabilities of RI and WI based on that of ONI. We first model the occurrences of multiple writers with the following queuing model. The workload of each (reader or writer) client is modeled as an independent queue characterized by the rate of operations and the service time of each operation[5]. We assume a Poisson process with parameter λ for the scenario of each client issuing a sequence of read/write operations, and assume an exponential distribution with parameter μ for the service time of each operation. We then have n independent, parallel $M/M/1/1/\infty/FCFS$ queues Q_i $(1 \leq i \leq n)$ (i.e., a single-server exponential queuing system, whose capacity is 1 with the "first come first served" discipline) [38]. The independence assumption of queues characterizes the feature of operations from different clients proceeding independently without waiting for each other. All queues have arrival rate λ and service rate μ. For each queue, if there is any operation in service, no more operations can enter it. Let $X^i(t)$ be the number of operations in queue i at time t. Then $X^i(t)$ is a continuous-time Markov chain with two states: 0 when the queue is empty and 1 when some operation is being served. Its stationary distribution $P_s \triangleq P(X^i(\infty) = s), s \in \{0, 1\}$ is: $P_0 = \frac{\mu}{\mu+\lambda}$ and $P_1 = \frac{\lambda}{\mu+\lambda}$.

As for RI, given a read operation r in Q_i, let W'_{cr} be a random variable denoting the number of writes w' satisfying $r_{st} \in [w'_{st}, w'_{ft}]$ in RI. The probability that r starts during the service period of some write w' in Q_j equals the probability that when r arrives at Q_i, it finds Q_i empty, and finds Q_j full as a bystander (with the constraint that Q_j is a writer queue). Since the events in different queues are independent, by the PASTA property [38], we have:

$$\mathbb{P}\{W'_{cr} = x\} = \binom{n_w}{x} \left(\frac{\lambda}{\mu+\lambda}\right)^x \left(\frac{\mu}{\mu+\lambda}\right)^{n_w-x+1} \tag{3.3}$$

Conditional on $W'_{cr} = x$, the probability of RI is no more than the sum of each w' forming RI with r separately. By considering all possible choices of w', we have:

$$\begin{aligned}\mathbb{P}\{\text{RI}\} &= \sum_{x=1}^{n_w} \mathbb{P}\{W'_{cr} = x\} \cdot \mathbb{P}\{\text{RI} \mid W'_{cr} = x\} \\ &\leq \sum_{x=1}^{n_w} \mathbb{P}\{W'_{cr} = x\} \cdot x\mathbb{P}\{\text{ONI}\}\end{aligned} \tag{3.4}$$

As for WI, given a specific write w' in Definition 3.2, let $W_{cw'}$ be a random variable denoting the number of writes w satisfying $w'_{st} \in [w_{st}, w_{ft}]$ in WI. Similarly, we have:

$$\mathbb{P}\{W_{cw'} = x\} = \binom{n_w - 1}{x} \left(\frac{\lambda}{\mu+\lambda}\right)^x \left(\frac{\mu}{\mu+\lambda}\right)^{n_w-x} \tag{3.5}$$

5. The Poisson process, along with exponential distribution, has been widely used for modeling the arrival phenomena, such as request arrivals in storage systems [36] and packet arrivals in Internet traffic [37].

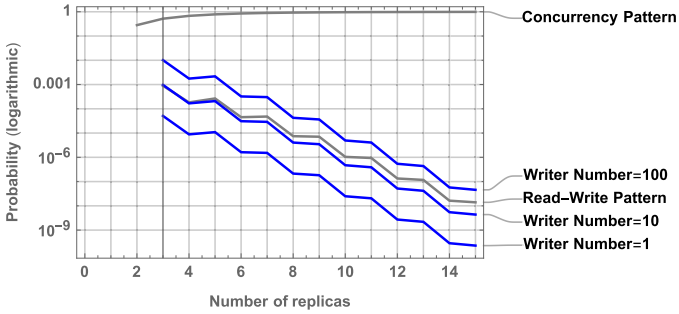

Fig. 7. The probabilities of atomicity violation.

Then we can bound the probability of WI by:

$$\begin{aligned}\mathbb{P}\{\mathrm{WI}\} &= \sum_{x=1}^{n_w-1} \mathbb{P}\{W_{cw'} = x\} \cdot \mathbb{P}\{\mathrm{WI} \mid W_{cw'} = x\} \\ &\leq \sum_{x=1}^{n_w-1} \mathbb{P}\{W_{cw'} = x\} \cdot x\mathbb{P}\{\mathrm{ONI}\}\end{aligned} \tag{3.6}$$

Substituting Formula (3.2), (3.3), (3.4), (3.5), (3.6) into (3.1), we obtain a bound of the probability of atomicity violation in Formula (3.7).

To better illustrate the theoretical analysis results, we present the numerical results in Fig. 7. The results are obtained when setting $\lambda = \mu = 10s^{-1}$ and $\lambda_r = \lambda_w = 20s^{-1}$. The source code for calculation of the numerical results can be found in our open source project on line[6]. More detailed numerical results can be found in Section 2.3 in the appendix [29] available in the online supplemental material. The numerical results show that the probability of atomicity violation is quite low (mostly below 0.1% and can be below 10^{-9}) and will decrease when the number of replicas increases. Moreover, the probability of the concurrency pattern is close to 1. The probability of the read-write pattern is significantly less and decreases as the number of replicas increases. The probability of the read-write pattern ensures that the probability of the atomicity violation is almost zero. We will conduct more comprehensive experimental evaluations in Section 4, which further confirm our theoretical analysis results.

$$\begin{aligned}&\mathbb{P}\{\mathrm{Violation}\} \leq \sum_{x=1}^{n_w} \mathbb{P}\{W'_{cr} = x\} \cdot x\mathbb{P}\{\mathrm{ONI}\} \\ &+ \sum_{x=1}^{n_w-1} \mathbb{P}\{W_{cw'} = x\} \cdot x\mathbb{P}\{\mathrm{ONI}\} \\ &\approx \frac{(2n_w-1)\lambda\mu}{(\lambda+\mu)^2} \cdot \sum_{m=1}^{n_r} \left(\left(\sum_{k=0}^{n_r-1} \binom{n_r}{k} \binom{m-1}{n_r-k-1} p_0^k r^{n_r-k} s^m \right) \right. \\ &\left. \cdot e^{-q\lambda_w t} \frac{\alpha^q B(q, \alpha(N-q)+1)}{B(q, N-q+1)} \cdot \left(1 - \left(\frac{J_1}{B(q, N-q+1)} \right)^m \right) \right)\end{aligned} \tag{3.7}$$

3.4 Discussions

We also analyze the bound of data staleness and the probability of atomicity violation for the W1R2 and W1R1 algorithms. Although for single-writer registers, the W1R2 algorithm (also recognized as the ABD algorithm in the literature [39]) guarantees atomicity and the W1R1 algorithm (also called the PA2AM algorithm in [16]) achieves the almost strong consistency tradeoff, for multi-writer registers we observe that neither the W1R2 algorithm nor the W1R1 algorithm can provide sufficient consistency guarantees in terms of the data staleness and the probability of atomicity violation.

The reason mainly lies in the one round-trip write algorithm. Since the writers try to directly update the replicas without querying the existing versions of them first, the updates may often take no effect on the replicas. Specifically, it occurs frequently that, after a majority of replicas have already been updated with a larger version, some writer still tries to update the data with a smaller version. Moreover, the data written by the no-effect writes will be "invisible" to any following read. Clients may miss an arbitrary number of updates, so the data returned by a read operation can be arbitrarily stale.

The theoretical analysis and numerical results for the W1R2 and W1R1 algorithms are provided in Sections 3.1 and 3.2 in the appendix [29] available in the online supplemental material. These theoretical analysis results are also confirmed by further experimental evaluations, as shown in Section 3.3 in the appendix available in the online supplemental material.

4 Experiments and Evaluations

In this section, we conduct experiments to study whether the fast read (W2R1) algorithm can guarantee PAB in quorum-replicated datastores. We first explain important implementation details and describe the experiment setup. Then we present the experiment design, followed by discussions on the experiment results.

4.1 Implementation

Our implementation[7] is based on the open-source distributed datastore Cassandra [3], [32], which enables high availability and low data access latency based on quorum replication. Our implementation basically reuses the quorum mechanism in Cassandra except that we implement the versioning of replicas. Cassandra relies on synchronized clocks among replica servers and uses timestamps following the UTC standard, while we use the discrete timestamp $ver = (seq, id)$. We transfer the 64-bit timestamp of Cassandra into a pair of integers, letting the higher 32-bits store the sequence number and the lower 32-bits store the process id. In this way, we can reuse the timestamp maintenance and comparison schemes of Cassandra.

Cassandra implements a variety of optimizations, which are beyond accurate modeling in the theoretical analysis. Thus, we implement knobs which enable users to turn on/off the optimizations. Tuning off all the optimizations enables us to validate our theoretical analysis with the experimental evaluations. Turning on one optimization (and turning off other optimizations) each time enables us to study in depth

6. https://github.com/Lingzhi-Ouyang/Almost-Strong-Consistency-Cassandra/tree/master/numerical_results

7. Our implementation is available online at: https://github.com/Lingzhi-Ouyang/Almost-Strong-Consistency-Cassandra/tree/master/experiment

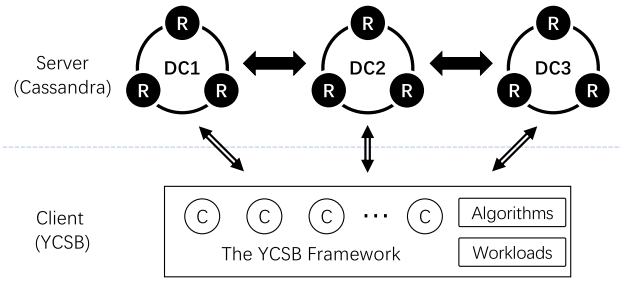

Fig. 8. Architecture of the experiment system.

the effect of this optimization. The optimizations we consider in the experiments include snitch, digest and read repair [32].

4.2 Experiment Setup

The experiments are conducted on a ThinkStation P710 server with the Intel Xeon(R) E5-2620 8-core 16-thread CPU (2.10 GHz), equipped with 64 GB DDR4 memory and 7200 RPM SATA disks. The operation system is Ubuntu Linux 18.04. We run up to 9 instances of Cassandra to simulate a cloud storage system with up to 3 data centers, each consisting of no more than 3 instances, as shown in Fig. 8. Each Cassandra instance, also denoted a node, acts as a server replica in our system model.

In order to accurately simulate a multi-data center storage systems, we derive the distribution of different types of message delays in a real geo-replicated environment. Specifically, we set up virtual machines in three different data centers (in north / south / east China) in the Alibaba Cloud [40]. We collect the ping traces for one week and use the statistics collected to derive the distribution of message delays[8].

We implement the client reading and writing algorithms using the database interface provided by the YCSB framework [33], [34]. We also use YCSB to generate a variety of workloads for our experiments. In the experiments, each client is represented by a YCSB instance using a single worker thread, guaranteeing that the operations issued by a client are sequential. A tunable number of YCSB instances representing multiple clients will execute write and read operations on one single shared register (key-value pair). This is due to the locality of atomicity and k-atomicity [35]. When each shared register provides k-atomicity, the storage system consisting of multiple registers is guaranteed to provide k-atomicity. Thus in our experiments, we only study the performance of reading/writing one register, which makes the microbenchmarks even more adverse to the algorithms. Also note that for one shared register, the replication factor (number of replicas for one shared register) typically ranges from 3 to 7. In our experiment, we study the replication factor up to 9.

We conduct our experiments in a simulated environment for two reasons. First, we tune the different types of environment factors in order to study whether PAB can be guaranteed in a variety of computing environments. Second, the definition of atomicity relies on the existence of a global clock, which is difficult to obtain in a geo-replication environment. In our experiments, we use the local clock of the physical machine to accurately approximate the global clock.

8. The ping traces are available online at: https://github.com/hengxin/aliyun-ping-traces

4.3 Experiment Design

We evaluate the fast read, i.e., W2R1, algorithm with no optimizations (denoted as W2R1), as well as the W2R1 algorithm with only one optimization turned on (denoted as Snitch, Digest and Repair). The W2R2 algorithm is also implemented mainly for the sake of performance comparison. Evaluation results concerning the W1R2 and W1R1 algorithms are presented in Section 3.3 in the appendix [29] available in the online supplemental material.

The performance metrics are naturally derived from our concern of the consistency-latency tradeoff: the read latency for the clients and the data consistency provided. As for data consistency, we consider both the staleness of accessed data (measured by the k-atomicity model) and the probability of atomicity violation, as indicated in the definition of PAB. Our k-atomicity-verification algorithm is mainly based on the idea of the atomicity verification algorithm of Gibbons and Korach proposed in [41].

In the experiments, we explore the effects of workload patterns, replica configurations and network conditions. In each experiment, only one parameter would be tuned while others remain the default values. The experiment configurations are listed in Table 2. The default workload pattern is 30 clients issuing 90,000 operations. Among all the operations, 90% are read operations and others are write operations. The operation issue rate per client varies from 5 to 10, which is mainly limited by the network delay we set. The injected one-way inter-data center delays of network communication are normally distributed with the average of 50ms and the standard deviation of 25ms (denoted as $\mathbb{N}(50, 25^2)$) by default, while the intra-data center delays follow $\mathbb{N}(5, 1^2)$. The delays between clients and servers also follow $\mathbb{N}(5, 1^2)$. The default replica configuration is one replica in each of the three data centers, and the default consistency level for both write and read operations are set the *QUORUM* option in Cassandra. Each experiment under the same environment setting is repeated 10 runs.

4.4 Evaluation Results

4.4.1 Decrease in Read Latency

We measure the ratio of the read latency of W2R1 to that of W2R2. As shown in Table 3, the read latency of W2R1 (with and without optimizations) is about 60% of that of W2R2 on average. This is mainly due to the one communication round-trip saved. The ratio of reduction is not exactly 50% because the read latency is also affected by other factors, e.g., I/O for logging, garbage collections, and thread and lock contentions [16].

With optimizations, the latency can be further influenced in different ways. In particular, the snitch optimization may further reduce the read latency by efficiently routing requests, while the repair optimization takes a little more time for replica synchronization. As for the digest optimization, the speculative nature and the possible retransmissions may increase the read latency, but the gain is mainly the saving of transmission bandwidth. Nevertheless, W2R1 with digest still has lower read latency by almost a quarter compared to W2R2.

We also find that, the variation of workloads, the number of replicas and the message delay between server replicas

TABLE 2
Experiment configurations

	Parameter	Tuning values	Default value
Workload	Client number	10 / 20 / 30 / 40	30
	Read ratio	0.5 / 0.6 / 0.7 / 0.8 / 0.9 /0.99	0.9
Replica	Replica configuration*	3 / 1_1_1 / 3_1_1 / 3_3_3	1_1_1
Network	Inter-DC delay/ms	$\mathbb{N}(\mu, \sigma^2)$, μ = 10 / 20 / 30 / 40 / 50, $\sigma = \frac{\mu}{2}$	$\mathbb{N}(50, 25^2)$

Replica configuration specifies the number of nodes that stores replicas in the Cassandra cluster. For example, 3 *means using 3 replicas and storing all of them in one single DC;* 3_1_1 *means using 5 replicas but storing 3, 1, 1 replicas in DC1, DC2, DC3, respectively.*

TABLE 3
Consistency-Latency Tradeoff Under the Default Setting

	k_{max}	$\mathbb{P}(k=1)$	$\mathbb{P}(k=2)$	$\mathbb{P}(k=3)$	$\mathbb{P}(violation)$	Read latency
W2R2	1	100%	0	0	0	100%
W2R1	3	99.9796%	0.0203%	0.0001%	0.0204%	53%
Snitch	2	99.9896%	0.0104%	0	0.0104%	52.3%
Digest	2	99.9926%	0.0074%	0	0.0074%	76.5%
Repair	2	99.9977%	0.0023%	0	0.0023%	53.9%

may affect the read latency in different ways. However, these environment factors have little impact on the ratio of reduced read latency. In general, the W2R1 algorithms (with and without optimizations) can reduce the average read latency by 23% to 48% compared to the W2R2 algorithm. More discussions on the effects of the environment factors are provided in Section 4 in the appendix [29] available in the online supplemental material.

4.4.2 Staleness and Probability of Data Inconsistency

We investigate data consistency from the perspectives of data staleness and the probability of atomicity violation. As shown in Table 3, under the default setting, W2R1 produces up to 3-atomic traces in the worst case, but the probabilities $\mathbb{P}(k=2)$ and $\mathbb{P}(k=3)$ are quite small [9]. The overall probability of violation is less than 0.02%, and the optimizations can further reduce the probability down to approximately 0.002%.

By varying the environment settings, we observe that data consistency can be impacted in different ways. However, we find that the probabilities of consistent reads in different environments are all over 99.7% and in most cases over 99.97%. The experimental results of probability coincide with the numerical results, which also confirm the validity of our probabilistic model. The data staleness in the worst-case is $k=4$. It is much less than the theoretical upper bound [10]. This is due to, the violation patterns for a larger data staleness value are much more complex and harder to be satisfied, i.e., the probability becomes lower and lower as the staleness increases. Since $\mathbb{P}(k=2)$ is already quite small, the probability for a larger staleness value under the same environment setting will be further lower than $\mathbb{P}(k=2)$ and closer to 0. Different optimizations have certain effects on the consistency guarantees, but, since the consistency guarantees are quite close to 100% atomicity, the effects of optimizations are limited. The experimental results are shown in Fig. 9.

First we investigate the impact of the client number. By varying the client number from 10 to 40, the degrees of data staleness and the probabilities of atomicity violation incurred by the W2R1 algorithms (with and without optimizations) rise slightly as shown in Fig. 9a. This verifies our theoretical analysis that the growth of the client number tends to intensify the concurrency complexity of operations. However, all of the W2R1 algorithms guarantee atomicity most of the time, with a probability that more than 99.97% read requests can obtain up-to-date data.

The occurrence of atomicity violation is also related to the ratio of read and write operations. Here, we use the read ratio (the ratio of reads to all operations) as a tunable parameter. By varying the read ratio from 0.50 to 0.99, we derive the results shown in Fig. 9b. As we can see, for all W2R1 algorithms, higher read ratio within certain range (0.5 - 0.9) results in higher atomicity violation probabilities. However, when the read ratio changes from 0.90 up to 0.99, the probabilities of consistent reads reversely increase except W2R1 with snitch. The reason is mainly two-fold. On the one hand, the increase of reads will raise the possibility of forming specific concurrency patterns and read-write patterns that involve stale read(s). On the other hand, the growth of the read ratio makes writes more rare and sparsely distributed, so it will be harder for reads to return stale written data. The only exception, W2R1 with snitch, leads to a monotonically decreasing consistency guarantee. With the snitch optimization, reads always select nearer replicas in priority for speeding up queries. Thus, the increase of the read ratio may cause more reads to miss the remote updates.

9. The metric k_{max} is the maximum value of k in terms of k-atomicity satisfied by the histories obtained from 10 runs of experiments under the same environment setting. The metric $\mathbb{P}(k=1)$ is the average probability of reading latest data over 10 runs of experiments under the same environment setting. Similarly, $\mathbb{P}(k=2)$ is the average probability of a read returning the second latest data, and so on.

10. The tight upper bound with 30 clients (at most 29 writer clients and 1 reader client) is $B = 29 + \frac{1}{2} \times 29 \times (29-1) + 1 = 436$. It can be proved that the probability to trigger the theoretical upper bound is extremely small.

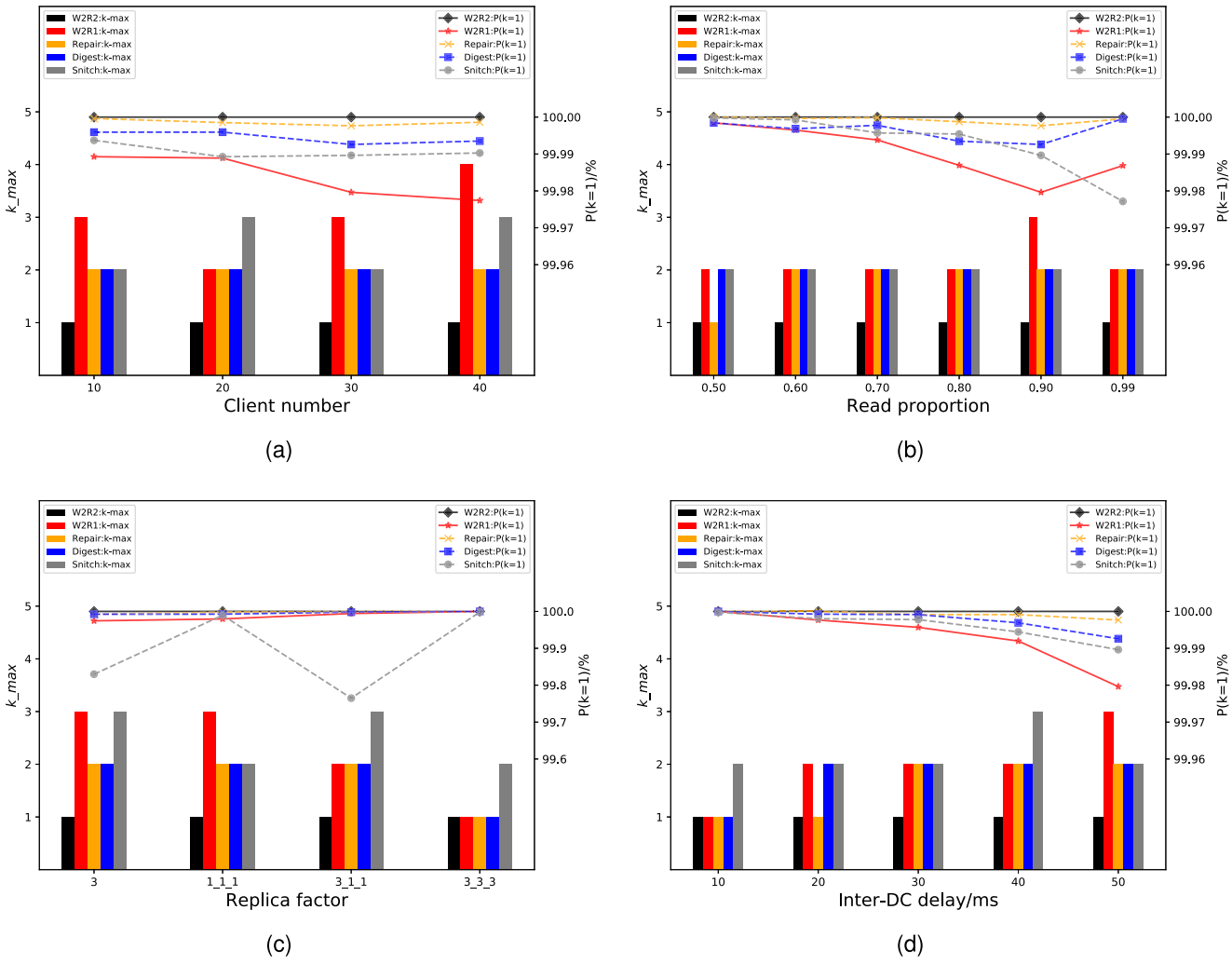

Fig. 9. Atomicity violation results under various environment parameters.

Replica configurations regarding the number and the distribution are critical to data consistency maintenance. The replica factor parameter in Cassandra specifies the number of replicas in each data center. In this experiment, the replica factor is varied in 3 (in one data-center), 1_1_1, 3_1_1, and 3_3_3. The consistency guarantees are shown in Fig. 9c. Overall, more replicas will improve the probabilities of consistent reads. This also confirms our numerical results that quorum-replicated algorithms implemented with higher replica numbers are able to guarantee lower probabilities of atomicity violation. The reason is that, with more replicas queried, it is more likely to access an replica that is already updated to the latest version. We also observe that W2R1 with snitch gets its maximum violation rate when the replica factor is 3_1_1. This is because more read operations will access the three replicas in the same data center and may miss the updates on the remote replicas.

As for network conditions, we mainly focus on the inter-data center network delay. The default one-way inter-data center delays are normally distributed with the average of 50ms and the standard deviation of 25ms (denoted as $\mathbb{N}(50, 25^2)$)). By varying μ from 10ms to 50ms, and σ from 5ms to 25ms, we obtain the results shown in Fig. 9d. The probabilities of atomicity violation rise slightly as the network delays grow higher. This is because larger jitters or variances of delays increase the gaps between updates on different replicas, giving more chances for reads to access out-of-sync replicas.

4.4.3 *Discussions*

From the evaluation results we can see that the W2R1 algorithm (including its optimizations) achieves the best of both worlds. It achieves both PAB and low latency in the common case. Although the details vary in different workload patterns, replica configurations or network conditions, low read latency and PAB are achieved in different environments.

Note that the W2R1 algorithm is only a fast read algorithm, and the write operations still require 2 round-trips of communications. However, both the theoretical analysis and the experimental evaluations show that the 2 round-trips of writes are necessary in the multi-writer case when using the version $ver = (seq, id)$. Fast (1 round-trip) write algorithms have poor consistency guarantees when multiple clients can write, no matter the read is fast or not.

5 Related Work

Consistency-latency tradeoff is the essential issue in replicated storage system design. Eventually consistent datastores which ensure low latency have been widely used and

commercially successful [1], [2], [3]. The ever growing demand of new application development also calls for datastores providing stronger consistency guarantees. Strong consistency models, e.g., atomicity or linearizability [15] and sequential consistency [42], may greatly ease application development. Spanner [43] supports linearizable distributed transactions, partly motivated by the complaints received from users that Bigtable [44] can be difficult to use for applications that need strong consistency in the presence of wide-area replication. Windows Azure Storage (WAS) [45] provides high availability with strong consistency for users, especially enterprise customers moving their line of business applications to the cloud. Clients of WAS always see the latest value that was written for a data object [46]. ZooKeeper [47] provides sequential consistency for users. The ZooKeeper Atomic Broadcast protocol [48] guarantees that updates from a client will be applied in the order that they were sent. Megastore [49] provides strong consistency guarantees by using semi-relational data model and synchronous replication.

Even when using weak consistency models, stronger (though not strong) consistency semantics may also greatly simplify the resolution of conflicts among replicas and thus ease the development of upper-layer applications. The Eiger system provides causal consistency, which is the strongest consistency guarantee possible when the system can be partitioned [9]. It also supports read-only and write-only transactions. The read-only transactions in Eiger normally complete in one round of local reads, and two rounds in the worst case. The concept of hybrid consistency is presented in [10], [50], with which strict strong consistency can be provided, but only for a selected part of important operations. For not-important operations, eventual consistency is provided. All existing works need to find a certain way to circumvent the impossibility results [6], [18], in order to achieve both strong consistency and low latency in certain sense. The almost strong consistency tradeoff mainly tackles the challenge by relaxing strictly strong consistency to almost strong consistency, as long as the data inconsistency perceived by the user is close to zero in the average case. In scenarios where low latency is important and the data reads are frequent, the statistically strong consistency guarantees may be well accepted, as long as the read latency can be significantly reduced.

Data consistency can be quantified from different dimensions, such as data versions, timeliness, numerical values and randomness [51], [52], [53]. The k-atomicity model [54] bounds the data version staleness for reads. Golab *et al.* studied the k-atomicity-verification problem [55], [56], [57], which is to check whether a given history is k-atomic. Taubenfeld [58] later re-defined k-atomic registers and analyzed them from the perspectives of computability and complexity. Time-based consistency conditions [59], [60] require writes to be globally visible within a period of time. The semantics of Δ-atomicity [55] allows reads to obtain data that are stale by up to Δ time units. Inspired by Δ-atomicity, the Γ-atomicity property [61] is arguably more accurate. TACT [51], a continuous consistency model, mixes the metric on numerical error with staleness. Both random registers [62] and PBS [52] allow one to obtain a probability distribution over the stale data versions that may be returned. However, they do not require deterministic worst-case guarantee on data staleness. Our probabilistic atomicity with well-bounded staleness, as well as almost strong consistency, integrate deterministically bounded staleness of versions with randomness. Further, the probability of reading stale data in our PAB algorithm is quantified in respect of atomicity instead of regularity (as in [62] and [52]), which is more challenging and we propose a stochastic queuing model for the analysis.

The emulation of atomic registers in distributed storage systems is the theoretical foundation of the fast read algorithm of probabilistic atomicity with well-bounded staleness. The ABD algorithm [39] uses quorum replication to emulate the atomic single-writer registers in unreliable, asynchronous networks where only a minority of replicas can fail. The ABD algorithm completes each read in two round-trips. For multi-writer registers, atomicity can be guaranteed by two round-trips of both read and write [28]. It is proved in [17] that when requiring both read and write to be fast (using only one round-trip of communication), it is not possible to guarantee atomicity. This impossibility result motivates us to propose the notion of "almost strong" consistency.

The consistency-latency tradeoff needs to and has been studied via comprehensive experiments. Many storage systems are equipped with tunable quorum mechanisms to meet a variety of consistency requirements in different scenarios [1], [3], [63], [64]. Many practical techniques for tradeoff tuning were developed over the above-mentioned tunable systems [65], [66], [67]. An adaptable SLA-aware consistency tuning framework for quorum-based stores [68] was also implemented and tested on Cassandra. In our previous work [16], the almost strong consistency tradeoff is only studied via experiments in a mobile file sharing scenario. In this work, we implement the diamond schema of four shared register emulation algorithms over Cassandra, mainly for the sake of exploration of the almost strong consistency tradeoff in the multi-writer register emulation scenarios.

6 Conclusion and Future Work

In this work we study the fast read algorithm which guarantees probabilistic atomicity with well-bounded staleness in replicated datastores where multiple clients can read and write data replicas. We propose the quorum-based algorithm schema to study the possible design options of fast read/write implementations. Then we analyze the consistency guarantees in the staleness and the probability dimensions.

We implement the algorithms following the diamond algorithm schema and evaluate the consistency-latency tradeoffs based on the instrumentation of Cassandra and the YCSB benchmark framework. The theoretical analysis and the experimental evaluations show that the fast read algorithm guarantees probabilistic atomicity with well-bounded staleness even when faced with various changes in the computing environment.

In our future work, we need to prove the lower bound that we inevitably need two round-trips for both reads and writes to strictly guarantee atomicity. We also plan to study the almost strong consistency tradeoff in large scale geo-replicated datastores, where software/hardware failures

and network partitions are common. More consistency metrics for data inconsistency measurement are also needed, in order to interpret the abstract notion of "almost strong" consistency in a variety of realistic application scenarios.

Acknowledgments

This work was supported in part by the National Natural Science Foundation of China (Grants 61932021, and 61772258), the Fundamental Research Funds for the Central Universities (Grant 14380063), and in part by the Collaborative Innovation Center of Novel Software Technology and Industrialization. The insightful comments of the anonymous reviewers greatly improved this article.

References

[1] G. DeCandia *et al.*, "Dynamo: Amazon's highly available key-value store," in *Proc. 21st ACM SIGOPS Symp. Operating Syst. Principles*, 2007, pp. 205–220. [Online]. Available: http://doi.acm.org/10.1145/1294261.1294281

[2] B. F. Cooper *et al.*, "PNUTS: Yahoo!'s hosted data serving platform," *Proc. VLDB Endowment*, vol. 1, no. 2, pp. 1277–1288, Aug. 2008. [Online]. Available: http://dx.doi.org/10.14778/1454159.1454167

[3] A. Lakshman and P. Malik, "Cassandra: A decentralized structured storage system," *SIGOPS Operating Syst. Rev.*, vol. 44, no. 2, pp. 35–40, Apr. 2010. [Online]. Available: http://doi.acm.org/10.1145/1773912.1773922

[4] S. Burckhardt, A. Gotsman, H. Yang, and M. Zawirski, "Replicated data types: Specification, verification, optimality," in *Proc. 41st ACM SIGPLAN-SIGACT Symp. Princip. Program. Lang.*, 2014, pp. 271–284. [Online]. Available: http://doi.acm.org/10.1145/2535838.2535848

[5] S. J. Park and J. K. Ousterhout, "Exploiting commutativity for practical fast replication," in *16th USENIX Symp. Networked Syst. Des. and Implementation, NSDI 2019, Boston, MA, February 26–28, 2019.*, 2019, pp. 47–64.

[6] E. A. Brewer, "Towards robust distributed systems (abstract)," in *Proc. 19th Annu. ACM Symp. Princip. Distrib. Comput.*, 2000, Art. no. 7.

[7] S. Gilbert and N. A. Lynch, "Perspectives on the CAP theorem," *Comput.*, vol. 45, no. 2, pp. 30–36, Feb. 2012.

[8] K. Sivaramakrishnan, G. Kaki, and S. Jagannathan, "Declarative programming over eventually consistent data stores," in *Proc. 36th ACM SIGPLAN Conf. Program. Lang. Des. Implementation*, 2015, pp. 413–424. [Online]. Available: http://doi.acm.org/10.1145/2737924.2737981

[9] W. Lloyd, M. J. Freedman, M. Kaminsky, and D. G. Andersen, "Stronger semantics for low-latency geo-replicated storage," in *Proc. 10th USENIX Conf. Netw. Syst. Des. Implementation*, 2013, pp. 313–328. [Online]. Available: http://dl.acm.org/citation.cfm?id=2482626.2482657

[10] C. Li, D. Porto, A. Clement, J. Gehrke, N. Preguiça, and R. Rodrigues, "Making geo-replicated systems fast as possible, consistent when necessary," in *Proc. 10th USENIX Conf. Operating Syst. Des. Implementation*, 2012, pp. 265–278. [Online]. Available: http://dl.acm.org/citation.cfm?id=2387880.2387906

[11] S. Burckhardt, "Principles of eventual consistency," *Found. Trends Program. Lang.*, vol. 1, no. 1–2, pp. 1–150, Oct. 2014. [Online]. Available: https://doi.org/10.1561/2500000011

[12] D. B. Terry, A. J. Demers, K. Petersen, M. J. Spreitzer, M. M. Theimer, and B. B. Welch, "Session guarantees for weakly consistent replicated data," in *Proc. 3rd Int. Conf. Parallel Distrib. Inf. Syst.*, 1994, pp. 140–150.

[13] 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. 23rd ACM Symp. Operating Syst. Princ.*, 2011, pp. 401–416. [Online]. Available: https://doi.org/10.1145/2043556.2043593

[14] L. Lamport, "On interprocess communication. Part I: Basic formalism," *Distrib. Comput.*, vol. 1, no. 2, pp. 77–85, 1986. [Online]. Available: http://dx.doi.org/10.1007/BF01786227

[15] M. P. Herlihy and J. M. Wing, "Linearizability: A correctness condition for concurrent objects," *ACM Trans. Program. Lang. Syst.*, vol. 12, pp. 463–492, Jul. 1990. [Online]. Available: http://doi.acm.org/10.1145/78969.78972

[16] H. Wei, Y. Huang, and J. Lu, "Probabilistically-Atomic 2-Atomicity: Enabling almost strong consistency in distributed storage systems," *IEEE Trans. Comput.*, vol. 66, no. 3, pp. 502–514, Mar. 2017. [Online]. Available: https://doi.org/10.1109/TC.2016.2601322

[17] P. Dutta, R. Guerraoui, R. R. Levy, and M. Vukolić, "Fast access to distributed atomic memory," *SIAM J. Comput.*, vol. 39, no. 8, pp. 3752–3783, Dec. 2010. [Online]. Available: http://dx.doi.org/10.1137/090757010

[18] D. J. Abadi, "Consistency tradeoffs in modern distributed database system design: Cap is only part of the story," *Comput.*, vol. 45, no. 2, pp. 37–42, Feb. 2012.

[19] Y. Huang, J. Cao, B. Jin, X. Tao, J. Lu, and Y. Feng, "Flexible cache consistency maintenance over wireless ad hoc networks," *IEEE Trans. Parallel Distrib. Syst.*, vol. 21, no. 8, pp. 1150–1161, Aug. 2010. [Online]. Available: https://doi.org/10.1109/TPDS.2009.168

[20] H. Attiya, S. Burckhardt, A. Gotsman, A. Morrison, H. Yang, and M. Zawirski, "Specification and complexity of collaborative text editing," in *Proc. ACM Symp. Princ. Distrib. Comput.*, 2016, pp. 259–268. [Online]. Available: http://doi.acm.org/10.1145/2933057.2933090

[21] N. Vidot, M. Cart, J. Ferrié, and M. Suleiman, "Copies convergence in a distributed real-time collaborative environment," in *Proc. ACM Conf. Comput. Supported Cooperative Work*, 2000, pp. 171–180.

[22] [Online]. Available: https://docs.microsoft.com/en-us/azure/cosmos-db/introduction

[23] [Online]. Available: https://azure.microsoft.com/en-us/support/legal/sla/cosmos-db/v1_3

[24] R. Jagadeesan and J. Riely, "Between linearizability and quiescent consistency," in *Automata, Languages, and Programming*, J. Esparza, P. Fraigniaud, T. Husfeldt, and E. Koutsoupias, Eds. Berlin, Germany: Springer, 2014, pp. 220–231.

[25] W. Golab, J. Hurwitz, and X. S. Li, "On the k-atomicity-verification problem," in *Proc. IEEE 33rd Int. Conf. Distrib. Comput. Syst.*, 2013, pp. 591–600. [Online]. Available: http://dx.doi.org/10.1109/ICDCS.2013.45

[26] M. Naor and A. Wool, "The load, capacity, and availability of quorum systems," *SIAM J. Comput.*, vol. 27, no. 2, pp. 423–447, Apr. 1998. [Online]. Available: http://dx.doi.org/10.1137/S0097539795281232

[27] M. Vukolic, *Quorum Systems With Applications to Storage and Consensus*. San Rafael, CA, USA: Morgan & Claypool Publishers, 2012.

[28] N. A. Lynch and A. A. Shvartsman, "Robust emulation of shared memory using dynamic quorum-acknowledged broadcasts," in *Proc. IEEE 27th Int. Symp. Fault Tolerant Comput.*, 1997, pp. 272–281.

[29] L. Ouyang, Y. Huang, H. Wei, and J. Lu, "Appendix to "achieving probabilistic atomicity with well-bounded staleness and low read latency in distributed datastores," Ins. Comput. Software, Nanjing Univ., Tech. Rep., 2020. [Online]. Available: https://github.com/Lingzhi-Ouyang/Almost-Strong-Consistency-Cassandra/b%lob/master/document/Appendix.pdf

[30] G. L. Peterson, "Myths about the mutual exclusion problem," *Inf. Process. Lett.*, vol. 12, pp. 115–116, 1981.

[31] C. Shao, J. L. Welch, E. Pierce, and H. Lee, "Multiwriter consistency conditions for shared memory registers," *SIAM J. Comput.*, vol. 40, no. 1, pp. 28–62, Jan. 2011. [Online]. Available: http://dx.doi.org/10.1137/07071158X

[32] [Online]. Available: http://cassandra.apache.org

[33] B. F. Cooper, A. Silberstein, E. Tam, R. Ramakrishnan, and R. Sears, "Benchmarking cloud serving systems with YCSB," in *Proc. 1st ACM Symp. Cloud Comput.*, 2010, pp. 143–154. [Online]. Available: http://doi.acm.org/10.1145/1807128.1807152

[34] [Online]. Available: https://github.com/brianfrankcooper/YCSB

[35] W. Golab, X. SteveLi, A. Lopez-Ortiz, and N. Nishimura, "Computing k-atomicity in polynomial time," *SIAM J. Comput.*, vol. 47, no. 2, pp. 420–455, 2018. [Online]. Available: https://doi.org/10.1137/16M1056389

[36] R. Pitchumani, S. Frank, and E. L. Miller, "Realistic request arrival generation in storage benchmarks," in *Proc. 31st Symp. Mass Storage Syst. Technol.*, 2015, pp. 1–10.

[37] T. Karagiannis, M. Molle, M. Faloutsos, and A. Broido, "A nonstationary poisson view of internet traffic," in *Proc. IEEE INFOCOM*, 2004, vol. 3, pp. 1558–1569.

[38] S. M. Ross, *Introduction to Probability Models*. New York, NY, USA: Academic Press, 2014.

[39] H. Attiya, A. Bar-Noy, and D. Dolev, "Sharing memory robustly in message-passing systems," *J. ACM*, vol. 42, no. 1, pp. 124–142, Jan. 1995. [Online]. Available: http://doi.acm.org/10.1145/200836.200869
[40] [Online]. Available: https://www.alibabacloud.com
[41] P. B. Gibbons and E. Korach, "Testing shared memories," *SIAM J. Comput.*, vol. 26, no. 4, pp. 1208–1244, 1997.
[42] L. Lamport, "How to make a multiprocessor computer that correctly executes multiprocess programs," *IEEE Trans. Comput.*, vol. C-28, no. 9, pp. 690–691, Sep. 1979. [Online]. Available: https://doi.org/10.1109/TC.1979.1675439
[43] J. C. Corbett *et al.*, "Spanner: Google's globally-distributed database," in *Proc. USENIX Symp. Operating Syst. Des. Implementation*, 2012, pp. 251–264. [Online]. Available: http://dl.acm.org/citation.cfm?id=2387880.2387905
[44] F. Chang *et al.*, "Bigtable: A distributed storage system for structured data," *ACM Trans. Comput. Syst.*, vol. 26, no. 2, pp. 4:1–4:26, Jun. 2008. [Online]. Available: http://doi.acm.org/10.1145/1365815.1365816
[45] B. Calder *et al.*, "Windows azure storage: A highly available cloud storage service with strong consistency," in *Proc. 23rd ACM Symp. Operating Syst. Princ.*, 2011, pp. 143–157.
[46] D. Terry, "Replicated data consistency explained through baseball," *Commun. ACM*, vol. 56, no. 12, pp. 82–89, 2013.
[47] A. Medeiros, "Zookeeper's atomic broadcast protocol: Theory and practice," Tech. Rep., 2012.
[48] F. P. Junqueira, B. C. Reed, and M. Serafini, "Zab: High-performance broadcast for primary-backup systems," in *Proc. IEEE/IFIP 41st Int. Conf. Depend. Syst. Netw.*, 2011, pp. 245–256.
[49] J. Baker *et al.*, "Megastore: Providing scalable, highly available storage for interactive services," in *Proc. Conf. Innovative Data Syst. Res.*, 2011, pp. 223–234.
[50] C. Li, J. A. Leitão, A. Clement, N. Preguiça, R. Rodrigues, and V. Vafeiadis, "Automating the choice of consistency levels in replicated systems," in *Proc. USENIX Conf. USENIX Annu. Tech. Conf.*, 2014, pp. 281–292. [Online]. Available: http://dl.acm.org/citation.cfm?id=2643634.2643664
[51] H. Yu and A. Vahdat, "Design and evaluation of a conit-based continuous consistency model for replicated services," *ACM Trans. Comput. Syst.*, vol. 20, no. 3, pp. 239–282, 2002.
[52] P. Bailis, S. Venkataraman, M. J. Franklin, J. M. Hellerstein, and I. Stoica, "Probabilistically bounded staleness for practical partial quorums," 2012, *arXiv :1204.6082*.
[53] D. Bermbach, "Benchmarking eventually consistent distributed storage systems," Ph.D. dissertation, Karlsruhe Ins. of Tech., Karlsruhe, Germany, 2014.
[54] A. Aiyer, L. Alvisi, and R. A. Bazzi, "On the availability of non-strict quorum systems," in *Proc. Int. Symp. Distrib. Comput.*, 2005, pp. 48–62.
[55] W. Golab, X. Li, and M. A. Shah, "Analyzing consistency properties for fun and profit," in *Proc. 30th Annu. ACM SIGACT-SIGOPS Symp. Princ. Distrib. Comput.*, 2011, pp. 197–206.
[56] W. Golab, J. Hurwitz, and X. Li, "On the k-atomicity-verification problem," in *Proc. IEEE 33rd Int. Conf. Distrib. Comput. Syst.*, 2013, pp. 591–600.
[57] W. Golab, X. Li, A. López-Ortiz, and N. Nishimura, "Computing weak consistency in polynomial time," in *Proc. ACM Symp. Princ. Distrib. Comput.*, 2015, pp. 395–404.
[58] G. Taubenfeld, "Weak read/write registers," in *Proc. Int. Conf. Distrib. Comput. Netw.*, 2013, pp. 423–427.
[59] A. Singla, U. Ramachandran, and J. Hodgins, "Temporal notions of synchronization and consistency in Beehive," in *Proc. 9th Annu. ACM Symp. Parallel Algorithms Architectures*, 1997, pp. 211–220.
[60] F. J. Torres-Rojas, M. Ahamad, and M. Raynal, "Timed consistency for shared distributed objects," in *Proc. 18th Annu. ACM Symp. Princ. Distrib. Comput.*, 1999, pp. 163–172.
[61] W. Golab, M. R. Rahman, A. AuYoung, K. Keeton, and I. Gupta, "Client-centric benchmarking of eventual consistency for cloud storage systems," in *Proc. IEEE 34th Int. Conf. Distrib. Comput. Syst.*, 2014, pp. 493–502.
[62] H. Lee and J. L. Welch, "Randomized registers and iterative algorithms," *Distrib. Comput.*, vol. 17, no. 3, pp. 209–221, 2005.
[63] C. Meiklejohn, "Riak PG:distributed process groups on dynamo-style distributed storage," in *Proc. 12th ACM Sigplan Workshop Erlang*, 2013, pp. 27–32.
[64] R. Sumbaly, J. Kreps, L. Gao, A. Feinberg, C. Soman, and S. Shah, "Serving large-scale batch computed data with project voldemort," in *Proc. 10th USENIX Conf. File Storage Technol.*, 2012, pp. 18–18.
[65] D. Bermbach, M. Klems, S. Tai, and M. Menzel, "MetaStorage: A federated cloud storage system to manage consistency-latency tradeoffs," in *Proc. IEEE 4th Int. Conf. Cloud Comput.*, 2011, pp. 452–459.
[66] 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. 24th ACM Symp. Operating Syst. Princ.*, 2013, pp. 309–324.
[67] M. McKenzie, H. Fan, and W. Golab, "Fine-tuning the consistency-latency trade-off in quorum-replicated distributed storage systems," in *Proc. IEEE Int. Conf. Big Data*, 2015, pp. 1708–1717.
[68] S. Sidhanta, W. Golab, S. Mukhopadhyay, and S. Basu, "OptCon: An adaptable SLA-aware consistency tuning framework for quorum-based stores," in *Proc. 16th IEEE/ACM Int. Symp. Cluster Cloud Grid Comput.*, 2016, pp. 388–397.

Lingzhi Ouyang received the BSc degree in geological engineering from Nanjing University, China, in 2017. He is currently working toward the PhD degree in computer science in the same university. His research interests include distributed algorithms and system reliability.

Yu Huang received the BSc and PhD degrees in computer science from the University of Science and Technology of China, in 2002 and 2007, respectively. He is currently a professor with the Department of Computer Science and Technology at Nanjing University. His research interests include distributed algorithms, distributed systems, formal methods, and system reliability. He is a member of China Computer Federation.

Hengfeng Wei received the BSc and PhD degrees in computer science and technology from Nanjing University, in 2009 and 2016, respectively. He is currently an assistant professor with the Department of Computer Science and Technology at Nanjing University. His research interests include distributed computing and formal methods.

Jian Lu received the PhD degree in computer science and technology from Nanjing University, China. He is a full professor with the Department of Computer Science and Technology and director of the State Key Laboratory for Novel Software Technology at Nanjing University. He has served as a vice chairman of the China Computer Federation, since 2011. His research interests include software methodologies, automated software engineering, software agents, and middleware systems.