Probabilistically-Atomic 2-Atomicity: Enabling Almost Strong Consistency in Distributed Storage Systems

Hengfeng Wei, Yu Huang, *Member, IEEE*, and Jian Lu

Abstract—A consistency/latency tradeoff arises as soon as a distributed storage system replicates data. For low latency, distributed storage systems often settle for weak consistency conditions, providing little guarantee on data consistency. In this paper, we propose the notion of *almost strong consistency* as an option for the consistency/latency tradeoff. It provides both deterministically bounded staleness of data versions for reads and probabilistic quantification on the rate of "reading stale data", while achieving low latency. We then investigate almost strong consistency in terms of *probabilistically-atomic 2-atomicity*. Our PA2AM algorithm for the single-writer model completes each read in one communication round-trip, and guarantees that each read obtains the value of within the latest two versions. To quantify the rate of "reading the stale version", we decompose the so-called "old-new inversion" anomaly into long-lived-write concurrency patterns and non-monotonic read-write patterns, and propose a queueing model and a timed balls-into-bins model to analyze them, respectively. The probabilistic analysis not only demonstrates that old-new inversions rarely occur, but also reveals that the read-write pattern dominates in preventing them from occurring. These are then supported by our experiments. To further demonstrate the benefits of probabilistically-atomic 2-atomicity, we also compare it to weak consistency conditions.

Index Terms—Almost strong consistency, probabilistically-atomic 2-atomicity, consistency/latency tradeoff, bounded staleness, quantifying consistency conditions

1 Introduction

Distributed storage systems [1], [2], [3], [4] underlying today's Internet services are expected to be fast, always available, highly scalable, and partition tolerant. To this end, they typically replicate data across machines and even datacenters, at the expense of introducing data inconsistency.

As soon as a storage system replicates data, a tradeoff between consistency and latency arises [5]. This consistency/latency tradeoff arguably has been highly influential in system design as it exists even when there are no network partitions [5]. In distributed storage systems, latency is widely regarded as a critical factor for a large class of applications. For instance, the experiments from Google [6] demonstrate that increasing web search latency 100 to 400 ms reduces the daily number of searches per user by 0.2 to 0.6 percent. Thus, most storage systems are designed for low latency in the first place. They often sacrifice strong consistency and settle for weaker ones, such as eventual consistency [2], [7], per-record timeline consistency [3], and causal consistency [8]. However, such weak consistency conditions usually provide little, or even worse, no guarantee on data consistency. Specifically, they neither make any deterministic guarantee on the staleness of the data returned by *reads* nor provide probabilistic hints on the rate of violations with respect to the desired strong consistency.

In this paper we propose the notion of *almost strong consistency* as an option for the consistency/latency tradeoff. Though low latency is highly desirable in practical systems, there are usually theoretical lower bounds on the achievable latency to assure strong consistency conditions [9], [10], [11], [12]. Here a natural question arises: *"What (strong) consistency condition can be achieved if low latency is a prerequisite?"*. Inspired by this question, almost strong consistency first demands an implementation with low latency, thus circumventing the theoretical lower bounds, but at the cost of data inconsistency. On the other hand, to prevent from sacrificing too much consistency, it requires deterministically bounded staleness of data versions for each *read*. Therefore, the users are confident that out-of-date data is still useful as long as they can tolerate certain staleness. Furthermore, it provides a probabilistic quantification on the low rate of "reading stale data". This ensures that the users are actually accessing up-to-date data most of the time.

We illustrate the idea of almost strong consistency with two scenarios. First, in the taxi transportation system, each taxi periodically reports its location data to the data server. Due to the natural locality of the update and request of location data, the city is partitioned into multiple areas and a data server is deployed in each area. The location data is replicated among all the data servers. This way the users all over the city can request the location data via a mobile application like Uber [13]. Though consistency is a desirable property, the user may be more concerned about how long he has to wait before his query can be served. Thus, the application may

• *The authors are with the State Key Laboratory for Novel Software Technology, Nanjing University, Nanjing, Jiangsu 210023, China. E-mail: {yuhuang, lj}@nju.edu.cn, hengxin0912@gmail.com.*

Manuscript received 29 Sept. 2015; revised 9 July 2016; accepted 26 July 2016. Date of publication 17 Aug. 2016; date of current version 16 Feb. 2017. Recommended for acceptance by A. Gordon-Ross. For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org, and reference the Digital Object Identifier below. Digital Object Identifier no. 10.1109/TC.2016.2601322

trade certain consistency for low latency, as long as the inconsistency is bounded and the application can still access up-to-date data most of the time [14]. Second, we consider a location-based mobile application, called "meet-up", that helps a group of people meet at some place, such as a restaurant, at the appointed time. People involved in these situations are often eager to know where everyone else is right now. Thus they participate, through their mobile phones, in a peer-to-peer network. The location of each phone is replicated at all the phones in the network. Each phone updates its location at its own rate and every phone is able to access the data at any time. Stale location data is acceptable as long as the staleness is deterministically bounded at a low level. Moreover, a high probabilistic guarantee of obtaining the latest data for each *read* would be quite desirable.

In distributed (key-value) storage systems, we investigate the generic notion of almost strong consistency in terms of *probabilistically-atomic 2-atomicity*,[1] with respect to atomicity [15] (a.k.a linearizability [16]). Atomicity is an ideally strong consistency condition, requiring each *read* to return the *latest* data version. Unfortunately, it has been theoretically proved that atomicity generally does not admit *low-latency* implementations, that *complete each read operation in one communication round-trip* [10]. For example, the ABD algorithm [17] for emulating atomic registers requires each *read* to complete in two round-trips. First of all, probabilistically-atomic 2-atomicity, as an option for the consistency/latency tradeoff, circumvents this impossibility result by giving priority to low latency and achieving as strong consistency as possible.

Second, with low latency in the first place, probabilistically-atomic 2-atomicity enforces the 2-atomicity semantics, which is a special case of k-atomicity [18] and guarantees that the value returned by each *read* is of one of the latest two versions. In our transportation system example, the taxi location data can still be useful if the data returned is no more stale than the previous version to the latest one. This is mainly because the location data cannot change abruptly and the taxi frequently updates its location.

Third, probabilistically-atomic 2-atomicity provides a probabilistic quantification on the rate of violations of atomicity, another perspective for expressing how strong consistency is "almost" guaranteed. In our transportation system example, since the user may request the location data of a number of taxies, the inconsistent data may not affect the quality of service experienced by the user, as long as only a small portion of the query return slightly stale data.

As required, our PA2AM algorithm guarantees 2-atomicity and completes each *read* in *one* round-trip. To quantify the rate of atomicity violations incurred in the PA2AM algorithm, we decompose the so-called old-new inversion anomaly [10], [19] into two patterns: Long-lived-write concurrency pattern and non-monotonic read-write pattern. We then propose a stochastic queueing model and a timed balls-into-bins model to analyze the two patterns, respectively. The probabilistic analysis not only demonstrates that old-new inversions rarely occur as expected, but also reveals that the read-write pattern dominates in preventing them from occurring.

1. Our use of "atomicity" concerns correctness of concurrent objects. Do not confuse it with the all-or-none property in transactions.

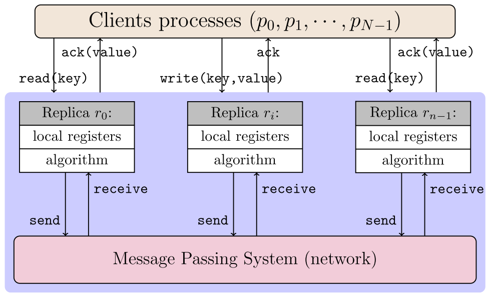


Fig. 1. Distributed (key-value) storage system model.

We have implemented a prototype distributed storage system among mobile phones, which provides 2-atomic data access based on the PA2AM algorithm and atomic data access on the ABD algorithm. The *read* latencies in our PA2AM algorithm have been significantly reduced, compared to those in the ABD algorithm. More importantly, the experimental results have supported our probabilistic analysis. Specifically, the proportion of old-new inversions incurred in the PA2AM algorithm is typically less than 0.01 percent. Furthermore, the proportion of read-write patterns among concurrency patterns (e.g., about 0.01 percent in some setting) is much less than that of concurrency patterns themselves (e.g., more than 50 percent in the same setting).

By comparing probabilistically-atomic 2-atomicity to weak consistency conditions, we find that probabilistically-atomic 2-atomicity brings the best of both worlds: it shares the performance advantage of weak consistency such as eventual consistency, and it has the statistically "almost strong" feature with respect to strong consistency, namely, atomicity. Thus, probabilistically-atomic 2-atomicity would be arguably as valuable an addition to the consistency/latency spectrum.

The paper is organized as follows. Section 2 proposes the generic notion of almost strong consistency and investigates it in terms of probabilistically-atomic 2-atomicity. Section 3 presents the low-latency PA2AM algorithm which satisfies 2-atomicity. Section 4 quantifies the atomicity violations incurred in the PA2AM algorithm. Section 5 presents the prototype storage system and experimental results. Section 6 reviews the related work. Section 7 concludes.

2 Almost Strong Consistency

In this section, we propose the generic notion of almost strong consistency, and instantiate it in terms of probabilistically-atomic 2-atomicity, in distributed storage systems.

2.1 Generic Notion of Almost Strong Consistency

The distributed storage system consists of a fixed number n of *replicas* that communicate through message passing (Fig. 1). Each replica maintains a *subset* of replicated key-value pairs (also referred to as *registers*). That is, we assume the *partial replication* model.

The distributed storage system supports two kinds of *operations* to an arbitrary (but finite) number of N *clients*: *1)* storing a value associated with a key, denoted *write(key, value)*; *and 2)* retrieving a value associated with a key, denoted *value* $\leftarrow$ *read(key)*. (The system model has decoupled the roles of replicas and clients, thus covering both scenarios described in Section 1.) Being replicated, different versions of the same register may co-exist in the distributed storage system. The concept of consistency conditions is then introduced to constrain the possible data versions that are allowed to be returned by each *read*. Particularly, strong consistency requires each *read* to obtain the latest data version according to some sequential order (discussed later in Section 2.2).

Being an option for the consistency/latency tradeoff, the notion of *almost strong consistency* generalizes the traditional strong consistency in three aspects:

1) It demands an implementation with low latency, thus circumventing the theoretical lower bounds on latency, but at the cost of data inconsistency.
2) It provides deterministically bounded staleness of data versions for each *read*; *and*
3) It also provides a probabilistic quantification on the rate of "reading stale data".

We emphasize that the generic notion of almost strong consistency not only specifies the allowable behavior of each *read* as the traditional consistency conditions do, but also is concerned with performance and analytical aspects of protocols.

2.2 Almost Strong Consistency in Terms of Probabilistically-Atomic 2-Atomicity

In distributed (key-value) storage systems, we investigate almost strong consistency in terms of *probabilistically-atomic 2-atomicity*. As preliminaries, we first review atomicity [20], focusing on read/write registers. From the view of clients, each operation is associated with two events: an *invocation* event and a *response* event. For a *read* (on a specific key), the invocation is denoted *read(key)*, and its response has the form *ack(value)*, returning some value to the client. For a *write*, the invocation is denoted *write(key, value)*, and its response is an *ack*, indicating its completion. We consider an asynchronous system, meaning that there is no real global time available to either the clients or the replicas. However, for specification, correctness proof, and offline analysis, we assume an imaginary global clock and all the events are time-stamped with respect to it [15]. Among all the *writes*, we posit, for each register, the existence of a special one which writes the *initial value*, at the very beginning of the imaginary global clock.

An *execution* σ of a distributed storage system is a sequence of invocations and responses. An operation o_1 *precedes* another operation o_2, denoted $o_1 \prec_\sigma o_2$ (or $o_1 \prec o_2$ if σ is clear from the context), if and only if the response of o_1 occurs in σ before the invocation of o_2. Two operations are considered *concurrent* (or called, *overlapping*) if neither of them precedes the other. An execution σ is said to be *well-formed* if each client invokes at most one operation at a time, that is, for each client p_i, $\sigma|i$ (the subsequence of σ restricted on p_i) consists of alternating invocations and matching responses, beginning with an invocation. A well-formed execution σ is *sequential* if for each operation in σ, its invocation is immediately followed by its response.

Intuitively, atomicity requires each operation to appear to take effect instantaneously at some point between its invocation and its response. More precisely,

Definition 2.1. *A distributed storage system satisfies **atomicity** [20] if, for each of its well-formed executions σ, there exists a permutation π of all the operations in σ such that π is sequential and*

- *[real-time requirement] If $o_1 \prec_\sigma o_2$, then o_1 appears before o_2 in π;* and
- *[read-from requirement] Each* read *returns the value written by the most recently preceding* write *in π on the same key.*

A distributed storage system is also said to *emulate* atomic registers if it satisfies atomicity.

As discussed in Section 1, atomicity does not admit low-latency implementations [10]. This impossibility result justifies the semantics of 2-atomicity, which is a special case of k-atomicity [18].

Definition 2.2. *A distributed storage system satisfies **2-atomicity** [18] if, for each of its well-formed executions σ, there exists a permutation π of all the operations in σ such that π is sequential and*

- *[real-time requirement] If $o_1 \prec_\sigma o_2$, then o_1 appears before o_2 in π;* and
- *[weak read-from requirement] Each* read *returns the value written by one of the latest two preceding* writes *in π on the same key.*

Similarly, a distributed storage system is also said to *emulate* 2-atomic registers if it satisfies 2-atomicity.

Probabilistically-atomic 2-atomicity instantiates the generic notion of almost strong consistency as follows.

Definition 2.3. *Probabilistically-atomic 2-atomicity consists of three parts:*

1) *It demands a low-latency implementation, thus circumventing the impossibility result on* read *latency required for atomicity [10], but at the cost of data inconsistency.*
2) *It enforces 2-atomicity, guaranteeing that each* read *obtains the value of within the latest two versions.*
3) *It provides a probabilistic quantification on the rate of actually reading the stale data version.*

Section 3 presents a low-latency implementation for 2-atomicity, meeting the first two conditions of probabilistically-atomic 2-atomicity. Section 4 quantifies its atomicity violations, meeting the third condition.

3 Achieving Probabilistically-Atomic 2-Atomicity

We present the PA2AM algorithm, a low-latency implementation for 2-atomicity, that meets the first two requirements of probabilistically-atomic 2-atomicity. Specifically, the PA2AM algorithm emulates 2-atomic, *single-writer* multi-reader registers, and completes each *read* in *one* round-trip. We highlight that the PA2AM algorithm and its probabilistic analysis in Section 4 are only for the single-writer model, and we discuss the multi-writer model in Section 7.

Despite its simplicity, (atomic,) single-writer registers are of both practical and theoretical interests. In practice, they are particularly suitable for a kind of applications, where the shared data has its natural "owner". The system state as a whole is composed of multiple single-writer registers. Each register is associated with a single user, the only one who can write it. All users can read all registers. Typical scenarios include the taxi transportation system and the "meet-up" application described in Section 1. Theoretically, atomic, single-writer registers are computationally equivalent to atomic, multi-writer registers [20]. Furthermore, single-writer registers are powerful enough to solve some classical synchronization problems [20] in which multiple processes communicate by writing their own registers and reading others'.

3.1 The PA2AM Algorithm

We use the asynchronous, crash-stop failure model, in which: *1)* Each communication channel is reliable and FIFO. This can be easily built on top of a basic message-passing network where messages can be delayed, lost, or delivered out of order, but they are not corrupted, e.g., by following the "communicate" procedure in [17]; *and 2)* Any subset of clients and a minority of replicas may crash.

The PA2AM algorithm is adapted straightforward from the ABD algorithm (specifically, the unbounded emulator) in [17] for atomicity. It makes use of versioning, and relies on the *majority communication rule* that requires each *read/write* operation to contact all the replicas and wait for acknowledgments from a majority of them before completing. Specifically,

- *write(key, value):* To write a value on a specific key, the single writer first generates a larger version (e.g., the next local sequence number) than those it has ever used, associates it with the key-value pair, sends the versioned key-value pair to all the replicas for this key, and waits for acknowledgments from a majority of them.
- *read(key):* To read from a specific key, the reader first queries and collects a set of versioned key-value pairs from a majority of the replicas for this key, from which it chooses the one with the largest version to return.

Each replica replaces its key-value pair whenever another one with a larger version from a *write* is received. It responds to the queries from *reads* with the versioned key-value pair it currently holds.

The pseudo-code for *read/write* operations and the replicas appears in Algorithm 1. Clearly, we have

Proposition 3.1. *The PA2AM algorithm completes each* read *operation in one communication round-trip.*

The PA2AM algorithm differs from the ABD algorithm [17] only in the *read* procedure: Each *read* of the PA2AM algorithm does *not* spend a second round-trip propagating the returned value (along with its version) to a majority of the replicas. The second round-trip of *read* in [17] (often referred to as the "write back" phase) is required to avoid old-new inversions, where two non-overlapping *reads*, both overlapping a *write*, obtain out-of-order values [10], [19]. As proved in the following section, the PA2AM algorithm, intentionally ignoring the write back phase, indeed achieves the emulation of 2-atomic, single-writer registers. As far as we know, this is a new contribution.

Algorithm 1. The PA2AM Algorithm Emulating 2-Atomic, Single-Writer Registers

```
1: procedure WRITE(key,value)                          ▷ for the writer
2:     increment (local) version for this key
3:     pfor each replica $s$ for key                     ▷ pfor: parallel for
4:         send [UPDATE, $key, value, version$] to $s$
5:     wait for [ACK]s from a majority of them
1: procedure READ key                                  ▷ for each reader
2:     vals ← $\emptyset$
3:     pfor each replica $s$ for key
4:         send [QUERY, $key$] to $s$
5:         v ← $[k, val, ver]$ from $s$
6:         vals ← vals $\cup$ {v}
7:     until a majority of them respond
8:     return val with the largest ver in vals
   ▷ Code for replicas below.
   $[k, val, ver]$ : local versioned key-value pairs
1: upon RECEIVE [QUERY, $key$] from $p_i$
2:     send $[k, val, ver]$ with $k = key$ to client $p_i$
1: upon RECEIVE [UPDATE, $key, value, version$] from $p_i$
2:     pick $[k, val, ver]$ with $k = key$
3:     if $ver < version$ then[2]
4:         $val \leftarrow value$
5:         $ver \leftarrow version$
6:     send [ACK] to client $p_i$
```

3.2 Correctness Proof of the PA2AM Algorithm

We prove that, in the PA2AM algorithm, the value returned by each *read* is of within the latest two versions. It is a case-by-case analysis, concerning the partial order among and the semantics of *read/write* operations.

Theorem 3.1. *The PA2AM algorithm achieves the emulation of 2-atomic, single-writer multi-reader registers, thus providing deterministically bounded (i.e., 2) staleness of data versions for each* read.

Proof. First, 2-atomicity is a local property [16], [21], meaning that an execution is 2-atomic if and only if for each register, the sub-execution of operations on that specific register is 2-atomic.[3] Thus we can assume that all the operations involved in the following correctness proof are performed on the same register.

According to the definition of 2-atomicity, it suffices to identify a permutation π of any execution of the PA2AM algorithm, and to prove that π is sequential and satisfies both the "real-time requirement" and the "weak read-from requirement".

For any execution σ, we construct its permutation π in the following manner:

- All the *write* operations issued by the single writer are totally ordered according to the versions they use.
- The *read* operations are scheduled one by one in order of their *invocation time:* A *read* r that reads

2. The "if-then" statement should be executed atomically when run in the multi-threaded mode.

3. The locality of 2-atomicity can be easily proved by following the proof of the locality of atomicity [16].

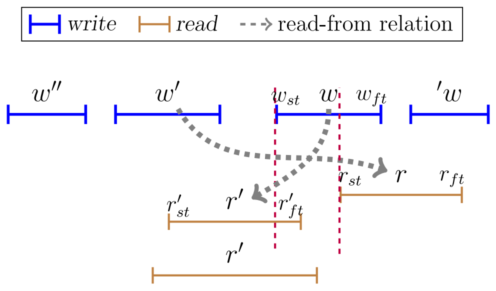


Fig. 2. Old-new inversion. Two non-overlapping *reads* r and r', both overlapping the *write* w, obtain out-of-order values. (*Time goes from left to right.*).

from a *write* w is scheduled immediately after both w and all the *read* operations preceding (i.e., $\prec_\sigma$) r (which have already been scheduled).

Clearly, this permutation π is sequential and satisfies the "real-time requirement" of 2-atomicity. It remains to show that it also satisfies the "weak read-from requirement" for each *read*. This argument is a case-by-case analysis, concerning the partial order among and the semantics of *read/write* operations.

For an operation o, let o_{st} denote its *start time* (i.e., the time of its invocation event), o_{ft} its *finish time* (i.e., the time of its response event), and $[o_{st}, o_{ft}]$ its *time interval* (Fig. 2 for an example). We also write $r = R(w)$ to denote the *"read-from"* relation in which the *read* r reads from the *write* w.

For any *read* operation r, we consider two exhaustive cases according to whether there are concurrent *write* operations with it in the execution σ.

CASE 1: *There is no concurrent write with* r. Due to the majority communication rule, r must read from its most recently preceding *write* w, and hence in π, it is scheduled between w and the next *write*.

CASE 2: *There are concurrent writes with* r, among which the earliest one in time is denoted w. Then for w, $r_{st} \in [w_{st}, w_{ft}]$ holds. There are two sub-cases according to the *write* from which r reads.

CASE 2.1: r *reads from some concurrent write*. In this case, r is scheduled in π between this *write* and its next one, since any *reads* preceding r cannot read from any *writes* later than w.

CASE 2.2: r *reads from its most recently preceding write in* σ *(denoted* w'*)*. Namely, $r = R(w')$. CASE 2.1 and CASE 2.2 are exhaustive since r cannot read from either any earlier *writes* than w' due to the majority communication rule or any *writes* it precedes. We now consider two exhaustive cases about other *read* operations than r (shown in Fig. 2).

CASE 2.2.1: *There is no read* r' *that precedes* r *in* σ *and is concurrent with* w. Formally, $\nexists r' : r'_{ft} \in [w_{st}, r_{st}]$. In π, r is scheduled between w' and its next *write* w.

CASE 2.2.2: *There is some read* r' *that precedes* r *and is concurrent with* w. Formally, $\exists r' : r'_{ft} \in [w_{st}, r_{st}]$. Distinguish two exhaustive cases.

CASE 2.2.2.1: r' *does not reads from* w. In π, r is scheduled between w' and its next *write* w, since r' cannot read from any *writes* later than w either.

CASE 2.2.2.2: r' *reads from* w. Namely, $r' = R(w)$. In this case, we obtain an old-new inversion (Fig. 2), where two non-overlapping *reads* (i.e., r and r'), both overlapping a *write* (i.e., w), obtain out-of-order values. In π, both r and r' are scheduled between w and its next *write* (i.e., $'w$ in Fig. 2). Consequently, r reads from w' which is its *second* most recently preceding *write* in π, satisfying the "weak read-from requirement" of 2-atomicity.

In conclusion, the permutation π satisfies the "weak read-from requirement" of 2-atomicity in all cases. Moreover, CASE 2.2.2.2 (and thus the old-new inversion anomaly) is the *only* case which leads to the violations of atomicity. □

4 Quantifying the Atomicity Violations

In this section, we quantify the atomicity violations incurred in the PA2AM algorithm. It follows from the correctness proof of Theorem 3.1 that the atomicity violations are exactly characterized by the old-new inversions in CASE 2.2.2.2. Furthermore, the proof has also identified the necessary and sufficient condition for the old-new inversion anomaly.

Definition 4.1. *The **old-new inversion** (ONI) involving a* read r *consists of the* read r, *two* writes w *and* w', *and a second* read r', *such that (Fig. 2)*

1) $r_{st} \in [w_{st}, w_{ft}]$, 2) w' *immediately precedes* w: $w' \prec w$, *and no other* writes *are between* w *and* w', 3) $r'_{ft} \in [w_{st}, r_{st}]$, 4) $r = R(w')$, and 5) $r' = R(w)$.

The five requirements for old-new inversion fall into two categories. The first three requirements involve the partial order $\prec$ on, and thus the *concurrency patterns* among, *read/write* operations. Intuitively, the higher degree of concurrency an execution shows, the more old-new inversions it may produce.

Definition 4.2. *The **long-lived-write concurrency pattern**[4] (CP) involving a* read r *consists of the* read r, *two* writes w *and* w', *and a second* read r', *such that*

1) $r_{st} \in [w_{st}, w_{ft}]$,
2) w' *immediately precedes* w: $w' \prec w$, *and no other* writes *are between* w *and* w', *and*
3) $r'_{ft} \in [w_{st}, r_{st}]$.

The concurrency pattern itself is not sufficient for old-new inversion. Only when the *read/write* semantics in the last two requirements of Definition 4.1 is also satisfied, does an old-new inversion arise. Thus, we define the *read-write pattern* conditioning on a *concurrency pattern* as follows.

Definition 4.3. *Given a concurrency pattern consisting of* $r, r', w,$ and w', *exactly as those in Definition 4.2, the **non-monotonic read-write pattern**[5] (RWP) requires*

4) $r = R(w')$ *and*
5) $r' = R(w)$

In this way, an old-new inversion occurs if and only if the read-write pattern arises given that a corresponding

4. The name highlights the intuition that *long-lived writes* (e.g., w in Fig. 2) spanning multiple *reads* are more likely to induce atomicity violations.

5. The name emphasizes that the *read* operation r violates the *monotonic-read property* [22] which requires *reads* to observe increasingly up-to-date data over time.

TABLE 1
Notations and Formulas

N: number of clients, n: number of replicas, $q \triangleq \lfloor n/2 \rfloor + 1$	Beta function: $B(x,y) = \int_0^1 t^{x-1}(1-t)^{y-1}dt$
λ: issue/arrival rate of operations, μ: service rate of operations	$r \triangleq \frac{(2\lambda+\mu)^2}{2(\mu+\lambda)^2}, s \triangleq \frac{1}{2}\frac{\mu}{\mu+\lambda}, p_0 \triangleq \frac{1}{2}\left(1+(\frac{\lambda}{\mu+\lambda})^2\right), t = \frac{1}{\lambda}$
λ_r, λ_w: rate for one-way message delay of *read/write*, $\alpha = \frac{\lambda_r}{\lambda_w+\lambda_r}$	J_1: $\mathbb{P}\{$replica 1 is the q_{th} one that receives the request of read $r'\}$ [a]

[a] *The formula of J_1 can be found in supplementary Appendix 2.3, which can be found on the Computer Society Digital Library at http://doi.ieeecomputersociety.org/10.1109/TC.2016.2601322; see (2.8).*

concurrency pattern has emerged. A concurrency pattern may contain more than one such r' defined in Definition 4.2, as illustrated in Fig. 2.

Let R' be a random variable denoting the number of r's in a concurrency pattern. Then, a read-write pattern arises if for *some* r', Definition 4.3 is satisfied. Therefore, the probability of old-new inversions conditioning on $\mathrm{R}' = \mathrm{m}$ ($m \geq 1$; m can be as large as the number of all *read* operations) is the product of the probability of the concurrency patterns conditioning on $\mathrm{R}' = \mathrm{m}$ and the probability of the read-write patterns conditioning on $\mathrm{R}' = \mathrm{m}$. By the law of total probability, we obtain

$$\begin{aligned}\mathbb{P}\{\text{atomicity violations}\} &= \mathbb{P}\{\text{ONI}\} \\ &= \sum_{m\geq 1} \mathbb{P}\{\text{CP} \mid \mathrm{R}' = \mathrm{m}\} \times \mathbb{P}\{\text{RWP} \mid \mathrm{R}' = \mathrm{m}\}.\end{aligned} \tag{4.1}$$

In the following two sections, we propose a stochastic queueing model and a timed balls-into-bins model to analyze the concurrency pattern and read-write pattern in (4.1), respectively. The frequently used notations and formulas are summarized in Table 1. The detailed calculations can be found in supplementary Appendices 1 and 2, available in the online supplemental material.

4.1 Quantifying the Rate of Concurrency Patterns

To quantify the rate of concurrency patterns conditioning on $\mathrm{R}' = \mathrm{m}$, we need an analytical model of the *workload* consisting of a sequence of *read/write* operations for each client. For each client, the characteristics of its workload are captured by the rate of operations issued by it and the service time of each operation (i.e., $[o_{st}, o_{ft}]$). We assume a Poisson process with parameter λ for the former one and an exponential distribution with parameter μ for the latter one.[6] The scenario of each client issuing a sequence of *read/write* operations is then encoded into a queueing model. Notice that in this model, we have abstracted away the implementation issues of the replicas and considered them as a storage service as a whole. We are only concerned with the service time of each operation, which is one of the key performance characteristics of the storage service.

We consider N independent, parallel $M/M/1$ queues, all with arrival rate λ and service rate μ. Here, the two M's indicate that both the inter-arrival and the service distribution are exponential (and thus memory-less, or Markovian), and the 1 indicate that there is a single server [25]. For each queue, we use the "first come first served" discipline and assume for simplicity that, if there is any operation in service, no more operations can enter it. The queue Q_0 represents the single writer.

To compute the probability that a concurrency pattern occurs in such a queueing system in the long run, we go through the following three steps.

Step 1: What is the stationary distribution for any two queues?

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:

$$P_0 = \frac{\mu}{\mu+\lambda}, P_1 = \frac{\lambda}{\mu+\lambda}.$$

Let $Y(t) = (X^i(t), Y^j(t))$ be the vector of the numbers of operations in queues Q_i and Q_j. Since any two queues are independent, $Y(t)$ is a continuous-time Markov chain with four states $(0,0), (0,1), (1,0),$ and $(1,1)$. Its stationary distribution $P_{i,j} \triangleq P(Y(\infty) = (i,j)), i,j \in \{0,1\}$ is:

$$P_{0,0} = \frac{\mu^2}{(\mu+\lambda)^2}, P_{0,1} = P_{1,0} = \frac{\mu\lambda}{(\mu+\lambda)^2}, P_{1,1} = \frac{\lambda^2}{(\mu+\lambda)^2}.$$

Step 2: Given a read r in Q_i, what is the probability of the event, denoted E, that it starts during the service period of some write w in Q_0 (formally, $r_{st} \in [w_{st}, w_{ft}]$ in Definition 4.2)?

The probability of E equals the probability that when r arrives at Q_i, it finds Q_i empty (denoted E_i) and as a bystander Q_0 full (denoted E_0). Since events E_i and E_0 are independent, we have

$$\begin{aligned}P(E) &= P(E_i \wedge E_0) = P(E_i) \cdot P(E_0) \\ &= P_0 \cdot P_1 \text{ (by the PASTA property [25])} = \frac{\mu\lambda}{(\mu+\lambda)^2}.\end{aligned}$$

Step 3: Conditioning on Step 2, what is the probability of the event, denoted $E_{N-1,m}$, that there are total m read operations (denoted r') in $N-1$ queues (besides Q_0) which finish during the time period $[w_{st}, r_{st}]$ (formally, $r'_{ft} \in [w_{st}, r_{st}]$ in Definition 4.2)?

First, the length of the time period $[w_{st}, r_{st}]$ is exactly the inter-arrival time of Q_i, which is exponential with rate λ. A probabilistic and combinatorial analysis shows that

$$\begin{aligned}\mathbb{P}\{\text{CP} \mid \mathrm{R}' = \mathrm{m}\} &= \mathbb{P}(E_{N-1,m}) \\ &= \sum_{k=0}^{N-2} \binom{N-1}{k} \binom{m-1}{N-k-2} p_0^k r^{N-k-1} s^m,\end{aligned} \tag{4.2}$$

when $m \geq 1$. For the special case $m = 0$, we have

$$\mathbb{P}\{\text{CP} \mid \mathrm{R}' = 0\} = \mathbb{P}(E_{N-1,0}) = p_0^{N-1}.$$

6. The Poisson process, along with the exponential distribution, has been widely used for modeling the arrival phenomena, like packet arrivals in Internet traffic [23] and request arrivals in storage systems [24].

Summing over m ($m \geq 1$), we get the probability that there exists a concurrency pattern (for some *read* r)

$$\mathbb{P}\{\text{CP}\} = 1 - \mathbb{P}\{\text{CP} \mid \text{R}' = 0\} = 1 - p_0^{N-1}. \qquad (4.3)$$

4.2 Quantifying the Rate of Read-Write Patterns

Given the concurrency patterns, we further quantify the rate of read-write patterns, namely, $r = R(w') \wedge \exists r' : r' = R(w)$, conditioning on $\text{R}' = \text{m}$. Here r' is among the m *read* operations in *Step 3* of Section 4.1. To this end, we explore in detail the majority communication rule used in the PA2AM algorithm. We assume that *1*) no node failure or link failure occurs[7]; *2*) to complete an operation, the client accesses all the n replicas and wait for the first $q \triangleq \lfloor n/2 \rfloor + 1$ acknowledgments from them; *and 3*) each replica processes the *read/write* operations on each individual register in FIFO order, the order it receives them. It follows that:

$$\begin{aligned} &\mathbb{P}\{\text{RWP} \mid \text{R}' = \text{m}\} \\ &\quad = \mathbb{P}\{r = R(w') \wedge \exists r' : r' = R(w)\} \\ &\quad \leq \mathbb{P}\{r \neq R(w) \wedge \exists r' : r' = R(w)\} \\ &\quad = \mathbb{P}\{r \neq R(w)\} \times \Big(1 - \mathbb{P}\{r' \neq R(w) \mid r \neq R(w)\}^m\Big), \end{aligned} \qquad (4.4)$$

where $r \neq R(w)$ (resp. $r' \neq R(w)$) denotes that r (resp. r') does not read from w. The inequality is due to the fact that $r = R(w')$ implies $r \neq R(w)$. We then calculate $\mathbb{P}\{r \neq R(w)\}$ and $\mathbb{P}\{r' \neq R(w) \mid r \neq R(w)\}$.

Which *write* would be read from by some *read* depends on the states of the replicas from which it collects the first q acknowledgments. The states of the replicas further depend on the timing issues in the PA2AM algorithm, such as message delays and the time lag between the events that messages are sent. Taking into account the timing issues, we propose the timed balls-into-bins model for the *read* and *write* procedures in the PA2AM algorithm. Let D_r (resp. D_w) be a continuous random variable denoting the *one-way* message delay for *read* (resp. *write*) operations. Let T be a continuous random variable denoting the time lag between the time when two messages of interest are sent, and t an observed value of T.

In the *timed balls-into-bins model*, there are n bins (corresponding to n replicas). Consider two robots R_1 and R_2 (corresponding to *read* or *write* operations) which can produce multiple balls (corresponding to messages) instantaneously. At time 0, robot R_1 *1*) produces n balls instantaneously; *2*) Immediately these n balls are independently sent to the n bins, one ball per bin; *3*) The delays for the balls going from the robot to its destination bin are independent and identically distributed with the same distribution as D_r or D_w defined above,[8] depending on whether the robot represents a *read* or a *write*.

At time t (defined above), robot R_2 independently does exactly the same thing as robot R_1 does (i.e., *1*), *2*), and *3*) for robot R_1 above).

7. The analysis concerning a failure model would be rather involved; we leave it to future work.

8. For simplicity, we assume through the identical distribution that each replica is equally likely to be part of a majority of replicas from which some operation waits for responses. Replicas could be distinguished by their own parameters in the same assumed probability distribution of delays.

To calculate $\mathbb{P}\{r \neq R(w)\}$, we are concerned with the model in which the robots R_1 and R_2 represent the *write* operation w and the *read* operation r involved in an old-new inversion, respectively. Furthermore, we assume that the random variable D_r (resp. D_w) for *one-way* time delay is exponentially distributed with rate λ_r (resp. λ_w). The time lag T between the events that w and r are issued corresponds to the time period $[w_{st}, r_{st}]$. That is, T is an exponential random variable with rate λ (*Step 3* of Section 4.1). For simplicity, we take the time lag t to be the expectation of T, i.e., $t = \frac{1}{\lambda}$. Finally, we are interested in the time point t' when exactly $q \triangleq \lfloor n/2 \rfloor + 1$ of the n bins have received the balls from R_2 (i.e., r), and denote the set of these q bins by B. In terms of the timed balls-into-bins model, the case of $r \neq R(w)$ corresponds to the event that none of the q bins in B receives a ball from R_1 (i.e., w) before it receives a ball from R_2 (i.e., r), and its probability is

$$\mathbb{P}\{r \neq R(w)\} = e^{-q\lambda_w t} \frac{\alpha^q B(q, \alpha(n-q)+1)}{B(q, n-q+1)}. \qquad (4.5)$$

Given $r \neq R(w)$ and $r' \prec r$, some messages from w are *known* to reach the replicas later than the time r' has collected enough acknowledgments and finished. To calculate $\mathbb{P}\{r' \neq R(w) \mid r \neq R(w)\}$, we consider a slightly generalized timed balls-into-bins model, in which robot R_2 picks some bins uniformly at random (without replacement) and sends a ball to each of them. For simplicity, we assume that there is at most one r' following the concurrency pattern in a single process. Based on this assumption, we have

$$\mathbb{P}\{r' \neq R(w) \mid r \neq R(w)\} = \begin{cases} \frac{J_1}{B(q, n-q+1)} & \text{if } n > 2, \\ 1 & \text{if } n = 2. \end{cases} \qquad (4.6)$$

Substituting (4.5) and (4.6) into (4.4) gives, for $n > 2$, the rate of read-write patterns conditioning on $\text{R}' = \text{m}$

$$\begin{aligned} \mathbb{P}\{\text{RWP} \mid \text{R}' = \text{m}\} \leq e^{-q\lambda_w t} & \frac{\alpha^q B(q, \alpha(n-q)+1)}{B(q, n-q+1)} \\ & \cdot \left(1 - \left(\frac{J_1}{B(q, n-q+1)}\right)^m\right). \end{aligned} \qquad (4.7)$$

For $n = 2$, we have $\mathbb{P}\{\text{RWP} \mid \text{R}' = \text{m}\} = 0$. Actually, in this case, there are no concurrency patterns at all.

4.3 Numerical Results and Discussions

Substituting (4.2) and (4.7) into (4.1), we obtain the rate of violating atomicity; see (3.1) of supplementary Appendix 3, available in the online supplemental material. From the numerical analysis, in which we have chosen $\lambda = \mu = 10s^{-1}$ and $\lambda_r = \lambda_w = 20s^{-1}$ based on the experimental results in Section 5.2, we observe that (see Fig. 3[9]; the detailed numerical results and discussions about the generality of our probabilistic analysis can be found in supplementary Appendix 3, available in the online supplemental material):

Observation 4.1. *Probabilistically, the PA2AM algorithm rarely violates atomicity.*

9. The graphs exhibit nearly regular staircase-like patterns due to the floor function $q \triangleq \lfloor n/2 \rfloor + 1$ in their formulas.

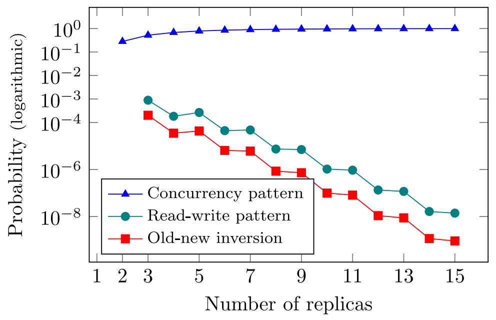


Fig. 3. The probabilities of concurrency patterns, read-write patterns, and old-new inversions ($N = n, \lambda = \mu = 10s^{-1}, \lambda_r = \lambda_w = 20s^{-1}$). ($\log 0$ *is not defined.*).

Observation 4.2. *The read-write patterns dominate in preventing from atomicity violations incurred in the PA2AM algorithm, compared to the concurrency patterns which occur quite often.*

5 Experiments and Evaluations

This section studies the PA2AM algorithm empirically. We have implemented a prototype distributed storage system among mobile phones, which provides both 2-atomic data access based on the PA2AM algorithm and atomic data access on the ABD algorithm (i.e., the unbounded emulator).[10] We compare *read* latencies of both algorithms. We measure the proportions of atomicity violations incurred in the PA2AM algorithm, and compare them to the numerical results. In Section 5.4, we compare probabilistically-atomic 2-atomicity to weak consistency conditions.

5.1 Experimental Design

Our prototype system comprises a collection of Google Nexus5 smartphones (CPU: 2.26 GHz, Heap: 30 MB, Android: 4.4.2), equipped with 72 Mbps wireless LAN. In both algorithms, each phone acts as both a client and a replica, targeted at the peer-to-peer, location-based “meet-up” mobile application in Section 1. As a client, each phone collects its own execution for offline analysis. Clocks on the phones are synchronized with the same desktop computer.

We explore three kinds of parameters: *1*) algorithm parameters: replication factor (i.e., the number of phones) and consistency conditions (i.e., atomicity or 2-atomicity); 2) workload parameters: the number of *read*/*write* operations issued by each client and the issue rate on each client; *and* 3) network parameter: the injected random delay in network communication, modeling the network latency *variances*.

We are concerned with two metrics, for each of which the microbenchmark has been chosen as adverse as possible to the PA2AM algorithm.

Read Latency. We compare *read* latencies of both algorithms by varying replication factors and issue rates of operations. Each client issues operations at a Poisson rate λ (= 5 or 50; 5 is for systems at leisure while 50 at busy) per second. For each λ, the replication factors vary from 2 to 5. Each reader issues 200,000 *read* operations. No additional delays are injected when evaluating read latencies.

10. The “communicate” procedure in [17] has been implemented to build reliable FIFO communication channels.

Atomicity Violations. We quantify the rate of atomicity violations incurred in the PA2AM algorithm by varying both replication factors and network delay variances. The replication factors vary from 2 to 5. For each replication factor, the injected random delays in network communication are uniformly distributed over integers in $[0, d)$ (d can be 0, 10, 20, 50, 100, or 200 ms). Each client issues 200,000 operations, at such a high Poisson rate $\lambda = 50$ per second that the system operates at its full capacity.

In all experiments, operations are performed on a single register, and the single writer issues only *write* operations. This makes the microbenchmarks even more adverse to the PA2AM algorithm.

5.2 Experimental Result 1: Read Latency

We report, for each execution, the minimum, the 25th, the median (marked by a line between box), the 75th, and the 99th quantiles and the average (marked by a star) of its read latencies, using box plots (Fig. 4). The proportion of outliers, those above the 99th quantile in each execution, varies from 0.869167 to 1.0 percent. On average, the average of *the outliers only* is around 3.8 times that of the whole data.

As shown in Fig. 4, the *read* latencies in the PA2AM algorithm have been reduced from those in the ABD algorithm by 20 to 41 percent. The reduced percentages are not necessarily 50 percent,[11] because the *read* latencies are affected by various factors such as network conditions, I/O for logging, garbage collections, and thread and lock contentions.

Fig. 4 also shows that the issue rates of operations have little impact on the average *read* latencies. For in both algorithms, operations proceed independently, without waiting for each other. Nevertheless, an execution issuing operations at rate $\lambda = 50$ tends to trigger garbage collections more frequently and thus incurs a higher variance of the *read* latencies than that at rate $\lambda = 5$.

In our experiments, the executions with two replicas have incurred high *read* latencies because each *read* has to wait for acknowledgments from all two replicas and is thus susceptible to network conditions.

5.3 Experimental Result 2: Atomicity Violations

To measure the proportion of the atomicity violations incurred in the PA2AM algorithm, we count the number of *read* operations (#R) and the occurrences of concurrency patterns (#CP) and read-write patterns (#RWP). Because each concurrency pattern or read-write pattern is associated with some *read* operation r, we are concerned with the following quantities:

$$\begin{aligned} \mathrm{P(CP)} &= \frac{\#\mathrm{CP}}{\#\mathrm{R}}, \mathrm{P(RWP|CP)} = \frac{\#\mathrm{RWP}}{\#\mathrm{CP}}, \text{ and} \\ \mathrm{P(ONI)} &= \mathrm{P(CP)} \cdot \mathrm{P(RWP|CP)} = \frac{\#\mathrm{RWP}}{\#\mathrm{R}}. \end{aligned} \tag{5.1}$$

Equation (5.1) is a practical approximation to (4.1) in theory, without going into the details of conditioning on $\mathrm{R}' = \mathrm{m}$.

11. Given that the PA2AM algorithm completes each *read* in one round-trip instead of two for the ABD algorithm, one might expect that the experimental, quantitative *read* latencies would be reduced by around 50 percent.

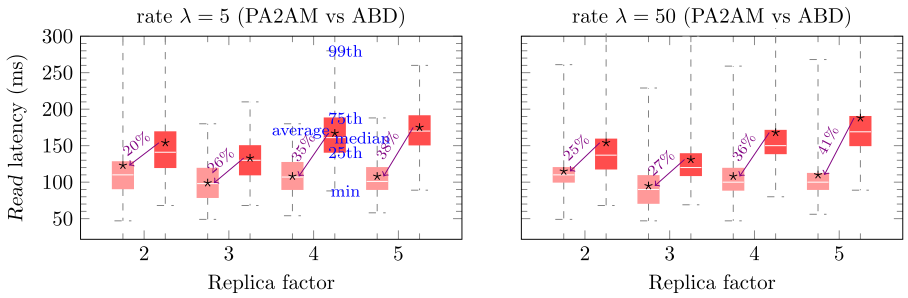


Fig. 4. Comparison of *read* latencies in both the PA2AM algorithm and the ABD algorithm.

TABLE 2
The Numbers and Proportions of Concurrency Patterns and Read-Write Patterns (*Replication Factor = 5)*

d (ms)	# *read* operations	# concurrency patterns	# read-write patterns	P(CP)	P(RWP | CP)	P(ONI)
10	800,000	269,061	47	0.336326	0.000174682	0.00005875
20	800,000	306,274	44	0.382843	0.000143662	0.000055
50	800,000	428,344	44	0.53543	0.000102721	0.000055
100	800,000	549,102	83	0.686378	0.000151156	0.00010375
200	800,000	627,814	100	0.784768	0.000159283	0.000125

TABLE 3
The Numbers and Proportions of Concurrency Patterns and Read-Write Patterns ($d = 0$ ms)

# replicas	# *read* operations	# concurrency patterns	# read-write patterns	P(CP)	P(RWP | CP)	P(ONI)
2	200,000	49,540	0	0.2477	0.	0.
3	400,000	157,350	34	0.393375	0.000216079	0.000085
4	600,000	286,508	6	0.477513	0.000020942	0.00001
5	800,000	429,047	282	0.53630875	0.000657271	0.0003525

The feasibility of such an approximation is justified by the experimental results presented shortly, in the sense that Observations 4.1 and 4.2 drawn from the numerical results based on the equations in theory fit well with the empirical data and (5.1).

Due to the limited space, Tables 2 and 3 summarize part of the experimental results. Table 2 varies the parameter d for network latency variances, while fixing the replication factor to be 5. Table 3 varies the replication factors, while fixing $d = 0$ ms (i.e., without injecting additional delays).

Table 2 shows that higher latency variances produce more concurrency patterns. Table 3 shows that the proportion of concurrency patterns grows as the replication factor increases, as implied by (4.3).

For the number of read-write patterns, the experimental results exhibit three features. First, no read-write patterns arise in only two replicas. This is because both *read* and *write* operations are required to collect acknowledgments from both replicas before completing. Second, there are fewer read-write patterns in the case of four replicas than those in the case of three or five replicas. In the case of four replicas, each *read* needs to collect three acknowledgments from (75 percent of) these four replicas, and gains more opportunities to obtain the latest data version. For three or five replicas, the majorities account for 66.7 and 60 percent, respectively. Third, Table 2 shows that network latency variance also contributes to the occurrences of read-write patterns since it may lead to out-of-order message delivery in the timed balls-into-bins model.

More importantly, the experimental results have supported our probabilistic analysis, especially Observations 4.1 and 4.2. First, the proportion of old-new inversions P(ONI) is quite small, e.g., less than 0.03 percent in most executions. Second, the proportion of read-write patterns among concurrency patterns P(RWP|CP) is much less than that of concurrency patterns P(CP) themselves. Namely, although concurrency patterns appear frequently (e.g., accounting for more than 50 percent in the setting of five replicas and $d = 0$ ms), only a quite small portion of them satisfy the requirements of the read-write pattern to constitute old-new inversions (e.g., about 0.06 percent in the same setting). This way the read-write patterns dominate in preventing from atomicity violations incurred in the PA2AM algorithm.

Table 4 compares the experimental results to the numerical results obtained in Section 4.3, on the probabilities/proportions of old-new inversions. It shows that they are quite close to each other, given that many numbers are of such a small order of magnitude.

5.4 Comparison to Weak Consistency Conditions

To further demonstrate the benefits of probabilistically-atomic 2-atomicity, we compare it to weak consistency conditions, in terms of both *read* latencies and atomicity violations incurred in their maintenance algorithms. We implemented eventual consistency [26] on our prototype distributed storage system. The protocol, denoted RWN, for eventual consistency we adopted is based on the $R + W \leq N$ formula described in [26], where N is the number of replicas of a particular register,[12] R is the number of replicas that a *read* operation needs to

12. We use N to denote the number of replicas in this section.

TABLE 4
Numerical Results and Experimental Results on the Probabilities/Proportions of Old-New Inversions

# replicas	numerical results ($\lambda = 10s^{-1}$)[a] (from Table 2 of Appendix 3, available in the online supplemental material)			experimental results ($\lambda = 50s^{-1}$) [a] (from Table 3)		
	$\mathbb{P}\{\mathrm{CP}\}$	$\mathbb{P}\{\mathrm{RWP} \mid \mathrm{CP}\}$	$\mathbb{P}\{\mathrm{ONI}\}$	P(CP)	P(RWP|CP)	P(ONI)
2	0.28125	0.	0.	0.2477	0.	0.
3	0.518555	0.00088802	0.000203683	0.393375	0.000216079	0.000085
4	0.677307	0.000183791	0.0000352958	0.477513	0.000020942	0.00001
5	0.781222	0.000266569	0.0000437181	0.53630875	0.000657271	0.0003525

[a] *Both $\lambda = 10s^{-1}$ and $\lambda = 50s^{-1}$ mean that the system operates at its full capacity.*

query before completing, and W is the number of replicas from which a *write* operation needs to receive acknowledgments before completing. The *read* and *write* procedures in the RWN protocol are the same with those in the PA2AM algorithm, except that $R = W = \lfloor N/2 \rfloor + 1$ and thus $R + W > N$ in the PA2AM algorithm; this comparability justifies our choice of eventual consistency and the RWN protocol.

5.4.1 Experimental Design

For both metrics of *read* latency and atomicity violations, we vary R, W, and N as follows:

- $N = 3$: $2 + 2 > 3$ for ABD, $2 + 2 > 3$ for PA2AM, $2 + 1 = 3$, $1 + 2 = 3$, and $1 + 1 < 3$ for RWN;
- $N = 5$: $3 + 3 > 5$ for ABD, $3 + 3 > 5$ for PA2AM, $3 + 2 = 5$, $2 + 3 = 5$, $2 + 2 < 5$, $1 + 4 = 5$, $1 + 3 < 5$, and $1 + 2 < 5$ for RWN.

In all experiments, each phone acts as both a client and a replica. Each client issues *reads/writes* at such a high Poisson rate of $\lambda = 50$ per second that the system operates at its full capacity. Each reader issues 200,000 *read* operations, and the single writer issues only *write* operations. All operations are performed on a single register. No additional delays are injected when evaluating either *read* latencies or atomicity violations.

5.4.2 Experimental Result: Read Latency

Fig. 5 compares the *read* latencies under various configurations of R, W, and N, covering the ABD algorithm for atomicity, the PA2AM algorithm for 2-atomicity, and the RWN protocol for eventual consistency. For each configuration, we report the minimum, the 1st, the 25th, the median, the 75th, the 95th, the 99th, and the maximum quantiles and the average (marked by unfilled pentagon) of its *read* latencies. We first observe that the averages of *read* latencies correlate positively with the values of both R and W. More importantly, the latency reduction from the ABD algorithm to the PA2AM algorithm is considerably greater than that from the PA2AM algorithm to the RWN protocol in typical configurations. For example, with five replicas, the latency reduction from $3 + 3 > 5$ (ABD) to $3 + 3 > 5$ (PA2AM) is 40.2 percent (i.e., from 194 to 116 ms), while the latency reduction from $3 + 3 > 5$ (PA2AM) to $2 + 2 < 5$ is 13.8 percent (i.e., from 116 to 100 ms). With three replicas, the latency reduction from $2 + 2 > 3$ (ABD) to $2 + 2 > 3$ (PA2AM) is 28.8 percent (i.e., from 132 to 94 ms), while the latency reduction from $2 + 2 > 3$ (PA2AM) to $1 + 2 = 3$ is 22.3 percent (i.e., from 94 to 73 ms).

5.4.3 Experimental Result: Atomicity Violations

Given an execution of the RWN protocol, we quantify its atomicity violations by counting the number (and computing the proportion) of *reads* in it for each possible staleness k. The staleness of each *read* is calculated with respect to the sequential permutation of this execution constructed in the way described in the correctness proof of Theorem 3.1.

To cover different variants of the RWN protocol, we consider both the original one, explicitly denoted RWN-All here, where each operation contacts *all* replicas and waits for responses from a majority of them and the variant, denoted RWN-Maj, where each operation contacts and waits for responses from *only a majority of replicas* chosen randomly. Intuitively, the effects of anti-entropy mechanisms such as gossip among replicas and read repair [2] on staleness lie in between those of RWN-All and RWN-Maj.

Fig. 6 shows the proportions of k-staleness incurred in the RWN-Maj protocol, under various configurations of $R + W \leq N$. We first observe that the RWN-Maj protocol for eventual consistency does not provide any deterministic, worst-case guarantee on data staleness for *reads*. Most executions shown in Fig. 6 incur staleness of levels $k > 5$. Second, the RWN-Maj protocol incurs atomicity violations much more frequently than the PA2AM algorithm. For example, in the case of $2 + 2 < 5$, about 19.0 percent (i.e., $1.0 - 0.8102775$) of *reads* have obtained stale values. For $1 + 1 < 3$, it is 52.3 percent.

Table 5 summarizes the behavior of the RWN-All protocol concerning atomicity violations, in terms of both the values of k in the worst cases and the proportions of staleness. Intuitively, RWN-All incurs less staleness than RWN-Maj, because

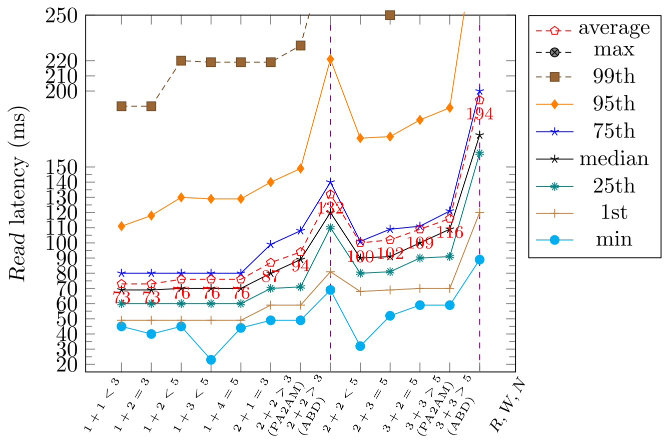


Fig. 5. Comparison of *read* latencies under various configurations of R, W, and N. (*Values above 250 ms are not shown.*)

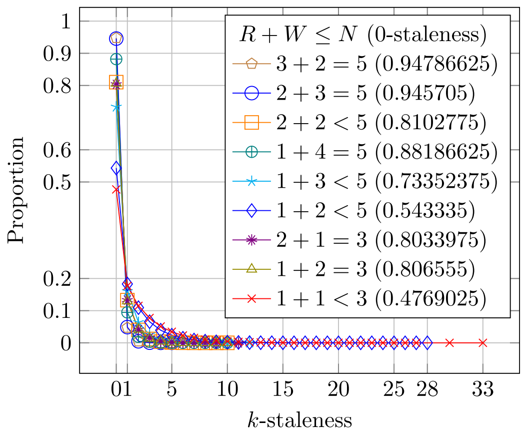


Fig. 6. The proportions of k-staleness incurred in the RWN-Maj protocol under various configurations of R, W, and N.

in the RWN-All protocol, all the operations are effectively being performed on the fastest R or W replicas. However, RWN-All suffers from severer atomicity violations than the PA2AM algorithm. First, all the RWN-All executions shown in Table 5 incur staleness of levels $k \geq 2$, while the PA2AM algorithm incurs only 1-staleness, i.e., the old-new inversions. Moreover, the RWN-All protocol incurs higher rates of staleness than the PA2AM algorithm. For example, with three replicas, 0.0315 percent of (i.e., 126) *reads* in the case of $1 + 2 = 3$ have obtained stale values, while 0.0085 percent of (i.e., 34) *reads* in $2 + 2 > 3$ (PA2AM) obtained stale values.

These experimental results have demonstrated that probabilistically-atomic 2-atomicity, as well as the PA2AM algorithm, brings the best of both worlds: it shares the performance advantage of weak consistency such as eventual consistency, and it has the statistically "almost strong" feature with respect to strong consistency, namely, atomicity. Thus, probabilistically-atomic 2-atomicity would be arguably as valuable an addition to the consistency/latency spectrum.

6 Related Work

We divide the related work into five categories.

Consistency/Latency Tradeoff. Compared to the CAP tradeoff [27] among consistency, availability, and partition tolerance, the consistency/latency tradeoff arguably has been more influential on the designs of distributed storage systems, as it is present at all times during system operation [5]. Practical techniques that allow fine tuning of the consistency/latency tradeoff in storage systems have been investigated [28], [29], [30]. Some other work study the consistency/latency tradeoff from a more theoretical perspective, by establishing lower bounds on the achievable latency to assure strong consistency [9], [11], [12]. In this paper we propose the notion of almost strong consistency as an option for the consistency/latency tradeoff.

Strong/Weak Consistency Conditions. Though many distributed storage systems trade strong consistency for low latency [2], [3], [7], [8], some choose to offer strong consistency. For example, Spanner [31] of Google supports externally-consistent (or equivalently, linearizable [16]) distributed transactions. In the paper [31] the authors have confirmed the complaints from users that Bigtable [1] (another product of Google which supports only eventually-consistent replication across datacenters) can be difficult to use for applications that want strong consistency in the presence of wide-area replication. Windows Azure Storage (WAS) [32] of Microsoft has also been driven by the feedback from many customers who want strong consistency, especially enterprise customers moving their line of business applications to the cloud. Consequently, WAS offers strong consistency so that clients always see the latest value that was written for a data object [22]. Even in Dynamo [2] of Amazon, as well as Apache Cassandra [33], which only claims eventual consistency, offers alternative strongly consistent reads by selecting different read and write quorums [22], [26].

Motivated by the question *"What (strong) consistency condition can be achieved if low latency is a prerequisite?"*, our notion of almost strong consistency requires both deterministically bounded staleness of data versions for each *read* and probabilistic quantification on the low rate of "reading stale data".

Complexity of Emulating Atomic, Single-Writer Registers. The ABD algorithm for atomicity [17], [19] emulates the atomic, single-writer registers in unreliable, asynchronous networks, given that a minority of nodes may fail. It requires each *read* to complete in two round-trips. Dutta et al. [10] proved that it is *impossible* to obtain a *fast* emulation with an arbitrary number of readers, where each *read* (and *write*) completes in *one* round-trip. Georgiou et al. [34] studied the semi-fast emulations (of atomic, single-writer registers) where *most reads* complete in one round-trip. Guerraoui et al. considered the best-cases complexity, assuming synchrony, no or few failures, and absence of operation contention. In this case, fast emulations do exist [35].

We investigate almost strong consistency in terms of probabilistically-atomic 2-atomicity. The PA2AM algorithm emulates 2-atomic, single-writer registers, completing each *read* operation in one round-trip.

Quantifying Consistency Conditions. Consistency conditions can be quantified from four perspectives: data versions, randomness, timeliness, and numerical values [36], [37], [38]. The semantics of k-atomicity [18] allows *reads* to obtain data of stale versions, as long as the staleness is bounded by k. Both random registers [39] and PBS [37] allow one to obtain a probability distribution over the set of stale data versions that may be returned. However, none of them requires deterministic worst-case guarantee on data staleness. Thus it is legal for

TABLE 5
Proportions of Staleness Incurred in the RWN-All Protocol under Various Configurations of R, W, and N

# replicas	replica factor = 3 (400,000 *read* operations)				replica factor = 5 (800,000 *read* operations)						
R, W, N	$1+1<3$	$1+2=3$	$2+1=3$	$2+2>3$ (PA2AM)	$1+2<5$	$1+3<5$	$1+4=5$	$2+2<5$	$2+3=5$	$3+2=5$	$3+3>5$ (PA2AM)
max$k$	6	4	2	**1**	2	2	2	2	2	2	**1**
$\sum_{k\geq1}$-staleness	0.0084125	0.000315	0.0004675	**0.000085**	0.00377875	0.002755	0.00406	0.0027225	0.0020275	0.002255	**0.0003525**

their algorithms to return arbitrarily stale data. As far as we know, our probabilistically-atomic 2-atomicity, as well as almost strong consistency, is the first to integrate deterministically bounded staleness of versions with randomness. Further, the rate of "reading stale data" in the PA2AM algorithm is quantified with respect to atomicity instead of *regularity* (as in [39] and [37]), which is more challenging since we shall deal with *concurrent* operations. Thus, we propose a stochastic queueing model for analyzing the concurrency pattern first and then a timed balls-into-bins model for analyzing the read-write pattern.

Timed consistency conditions [40], [41] require *writes* to be globally visible within a period of time. The Δ-atomicity property [42] allows *reads* to return values that are stale by up to Δ time units. The Γ-atomicity property [43], inspired by Δ-atomicity, is arguably more accurate. TACT [36], a continuous consistency model, integrates the metric on numerical error with staleness.

k-Atomicity. k-atomicity is first proposed in [18]. It also gives a protocol for emulating k-atomic, single-writer registers which, however, completes each *read* in two round-trips. Golab et al. have studied the k-atomicity-verification problem [21], [42], [44], which is to check whether a given execution is k-atomic. Later on, it seems that Taubenfeld [45] have re-defined k-atomic registers and studied them from more theoretical perspectives of computability and complexity.

7 Conclusion and Future Work

In this paper we propose the notion of *almost strong consistency* as an option for the consistency/latency tradeoff. It provides both deterministically bounded staleness of data versions for each *read* and probabilistic quantification on the rate of "reading stale data", while achieving low latency. We investigate almost strong consistency in terms of *probabilistically-atomic 2-atomicity*. The PA2AM algorithm satisfies 2-atomicity, and completes each *read* in *one* round-trip. We quantify the atomicity violations incurred in the PA2AM algorithm, both analytically and experimentally. We also compare probabilistically-atomic 2-atomicity to weak consistency conditions.

We identify two issues for future work. First, we plan to evaluate probabilistically-atomic 2-atomicity on *cloud storage systems* under a variety of benchmarks. In such systems, the registers are replicated among a collection of geographically distributed machines, and a large number of remote clients can be on either physical machines or mobile devices. The major challenge here is to achieve an accurate enough clock synchronization among all the clients due to (possibly) heterogeneous networks, high network latencies, and high latency variances [30], [38], that is crucial for the quantification of the timestamped executions. Second, we turn to k-atomic, *multi-writer* registers ($k \geq 1$). This is challenging because all *writes* are not totally ordered in the multi-writer model. We will first study whether k-atomic, multi-writer registers admit *low-latency* implementations that complete each *read* in one round-trip. Then we will investigate a particular low-latency implementation for *probabilistic* atomic, multi-writer registers, and quantify its behavior with respect to atomicity.

Acknowledgments

This work is supported by the National 973 Program of China (2015CB352202), the National Science Foundation of China (61272047, 91318301, 61321491), the Collaborative Innovation Center of Novel Software Technology and Industrialization, and the program A for Outstanding PhD candidate of Nanjing University. The authors are grateful for the constructive discussions from MathOverflow,[13] on the calculations in Section 4. Finally, the insightful comments of the anonymous reviewers greatly improved this paper. Yu Huang is the corresponding author.

References

[1] F. Chang, et al., "Bigtable: A distributed storage system for structured data," in *Proc. 7th Symp. Operating Syst. Des. Implementation*, 2006, pp. 205–218.
[2] 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.
[3] B. F. Cooper, et al., "PNUTS: Yahoo!'s hosted data serving platform," *Proc. VLDB Endowment*, vol. 1, no. 2, pp. 1277–1288, 2008.
[4] D. Beaver, S. Kumar, H. C. Li, J. Sobel, and P. Vajgel, "Finding a needle in haystack: Facebook's photo storage," in *Proc. 9th USENIX Conf. Operating Syst. Des. Implementation*, 2010, pp. 47–60.
[5] D. Abadi, "Consistency tradeoffs in modern distributed database system design," *IEEE Comput.*, vol. 45, no. 2, pp. 37–42, Feb. 2012.
[6] J. Brutlag, "Speed matters for Google web search," (2009). [Online] Available: https://services.google.com/fh/files/blogs/google_delayexp.pdf, Accessed on: Aug. 7, 2016.
[7] D. B. Terry, M. M. Theimer, K. Petersen, A. J. Demers, M. J. Spreitzer, and C. H. Hauser, "Managing update conflicts in Bayou, a weakly connected replicated storage system," in *Proc. 15th ACM Symp. Operating Syst. Principles*, 1995, pp. 172–182.
[8] 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. Principles*, 2011, pp. 401–416.
[9] M. J. Fischer and N. A. Lynch, "A lower bound for the time to assure interactive consistency," *Inf. Process. Lett.*, vol. 14, no. 4, pp. 183–186, 1982.
[10] P. Dutta, R. Guerraoui, R. R. Levy, and A. Chakraborty, "How fast can a distributed atomic read be?" in *Proc. 23rd Annu. ACM Symp. Principles Distrib. Comput.*, 2004, pp. 236–245.
[11] L. Lamport, "Lower bounds for asynchronous consensus," *Distrib. Comput.*, vol. 19, no. 2, pp. 104–125, 2006.
[12] F. Nawab, V. Arora, D. Agrawal, and A. El Abbadi, "Minimizing commit latency of transactions in geo-replicated data stores," in *Proc. ACM SIGMOD Int. Conf. Manage. Data*, 2015, pp. 1279–1294.
[13] Uber. (2016). [Online] Available: https://www.uber.com/, Accessed on: Aug. 7, 2016.
[14] 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.
[15] L. Lamport, "On interprocess communication (part II: Algorithms)," *Distrib. Comput.*, vol. 1, no. 2, pp. 86–101, 1986.
[16] M. P. HerlihyandJ. M. Wing, "Linearizability: A correctness condition for concurrent objects," *ACM Trans. Programming Languages Syst.*, vol. 12, no. 3, pp. 463–492, 1990.
[17] H. Attiya, A. Bar-Noy, and D. Dolev, "Sharing memory robustly in message-passing systems," *J. ACM*, vol. 42, no. 1, pp. 124–142, 1995.
[18] A. Aiyer, L. Alvisi, and R. A. Bazzi, "On the availability of non-strict quorum systems," in *Proc. 19th Int. Conf. Distrib. Comput.*, 2005, pp. 48–62.
[19] H. Attiya, "Robust simulation of shared memory: 20 years after," *Bulletin EATCS*, vol. 100, pp. 99–113, 2010.
[20] H. Attiya and J. Welch, *Distributed Computing: Fundamentals, Simulations and Advanced Topics*. New York, NY, USA: Wiley, 2004.

13. No. 163869 (http://mathoverflow.net/q/163869/28199) and No. 207800 (http://mathoverflow.net/q/207800/28199).

[21] 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.

[22] D. Terry, "Replicated data consistency explained through baseball," *Commun. ACM*, vol. 56, no. 12, pp. 82–89, 2013.

[23] T. Karagiannis, M. Molle, M. Faloutsos, and A. Broido, "A nonstationary Poisson view of internet traffic," in *Proc. 23rd Annu. Joint Conf. IEEE Comput. Commun. Soc.*, 2004, pp. 1558–1569.

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

[25] S. M. Ross, *Introduction to Probability Models*, 10th ed. San Francisco, CA, USA: Academic Press, 2010.

[26] W. Vogels, "Eventually consistent," *Commun. ACM*, vol. 52, no. 1, pp. 40–44, 2009.

[27] E. Brewer, "Towards robust distributed systems," in *Proc. 19th Annu. ACM Symp. Principles Distrib. Comput.*, 2000, Art. no. 7.

[28] D. Bermbach, M. Klems, S. Tai, and M. Menzel, "Metastorage: A federated cloud storage system to manage consistency-latency tradeoffs," in *Proc. IEEE Int. Conf. Cloud Comput.*, 2011, pp. 452–459.

[29] 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. Principles*, 2013, pp. 309–324.

[30] M. McKenzie, H. Fan, and W. M. 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.

[31] J. C. Corbett, et al., "Spanner: Google's globally-distributed database," in *Proc. 10th USENIX Symp. Operating Syst. Des. Implementation*, 2012, pp. 251–264.

[32] B. Calder, et al., "Windows Azure storage: A highly available cloud storage service with strong consistency," in *Proc. 23rd ACM Symp. Operating Syst. Principles*, 2011, pp. 143–157.

[33] A. Lakshman and P. Malik,"Cassandra: A decentralized structured storage system," *ACM SIGOPS Operating Syst. Rev.*, vol. 44, no. 2, pp. 35–40, 2010.

[34] C. Georgiou, N. C. Nicolaou, and A. A. Shvartsman, "On the robustness of (semi) fast quorum-based implementations of atomic shared memory," in *Proc. 22nd Int. Symp. Distrib. Comput.*, 2008, pp. 289–304.

[35] R. Guerraoui and M. Vukolić, "Refined quorum systems," in *Proc. 26th Annu. ACM Symp. Principles Distrib. Comput.*, 2007, pp. 119–128.

[36] H. Yu and A. Vahdat, "Design and evaluation of a conit-based continuous consistency model for replicated services," *ACM Trans. Comp. Syst.*, vol. 20, no. 3, pp. 239–282, 2002.

[37] P. Bailis, S. Venkataraman, M. J. Franklin, J. M. Hellerstein, and I. Stoica,"Probabilistically bounded staleness for practical partial quorums," *Proc. VLDB Endowment*, vol. 5, no. 8, pp. 776–787, 2012.

[38] D. Bermbach, "Benchmarking eventually consistent distributed storage systems," Ph.D. dissertation, Fakultät für Wirtschaftswissenschaften, Karlsruhe Inst. Technol., Karlsruhe, Germany, 2014.

[39] H. Lee and J. L. Welch, "Randomized registers and iterative algorithms," *Distrib. Comput.*, vol. 17, no. 3, pp. 209–221, 2005.

[40] 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.

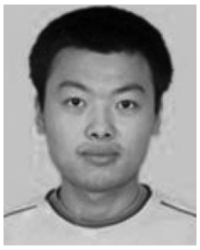
[41] F. J. Torres-Rojas, M. Ahamad, and M. Raynal, "Timed consistency for shared distributed objects," in *Proc. 18th Annu. ACM Symp. Principles Distrib. Comput.*, 1999, pp. 163–172.

[42] W. Golab, X. Li, and M. A. Shah, "Analyzing consistency properties for fun and profit," in *Proc. 30th Annu. ACM SIGACT-SIGOPS Symp. Principles Distrib. Comput.*, 2011, pp. 197–206.

[43] 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.

[44] W. Golab, X. S. Li, A. López-Ortiz, and N. Nishimura, "Computing weak consistency in polynomial time: [Extended abstract]," in *Proc. ACM Symp. Principles Distrib. Comput.*, 2015, pp. 395–404.

[45] G. Taubenfeld, "Weak read/write registers," in *Proc. Int. Conf. Distrib. Comput. Netw.*, 2013, pp. 423–427.



Hengfeng Wei received the BSc degree in computer science from Nanjing University, China, in 2009, where he is working toward the PhD degree in computer science. His research interests include distributed computing theory and formal specification and verification.

Yu Huang received the BSc and PhD degrees in computer science from the University of Science and Technology, China. He is currently an associate professor in the Department of Computer Science and Technology, Nanjing University. His research interests include distributed computing theory, formal specification and verification, and pervasive context-aware computing. He is a member of the IEEE and the China Computer Federation.

Jian Lu received the BSc, MSc, and PhD degrees in computer science from Nanjing University, China. He is currently a professor in the Department of Computer Science and Technology and the director of the State Key Laboratory for Novel Software Technology, Nanjing University. His research interests include software methodologies, software automation, software agents, and middleware systems.