The Complexity of Robust Atomic Storage

Dan Dobre
TU Darmstadt
Darmstadt, Germany
dan@cs.tu-darmstadt.de

Rachid Guerraoui
EPFL
Lausanne, Switzerland
rachid.guerraoui@epfl.ch

Matthias Majuntke
TU Darmstadt
Darmstadt, Germany
majuntke@cs.tu-darmstadt.de

Neeraj Suri
TU Darmstadt
Darmstadt, Germany
suri@cs.tu-darmstadt.de

Marko Vukolić
EURECOM
Sophia-Antipolis, France
marko.vukolic@eurecom.fr

ABSTRACT

We study the time-complexity of robust atomic read/write storage from fault-prone storage components in asynchronous message-passing systems. Robustness here means wait-free tolerating the largest possible number t of Byzantine storage component failures (optimal resilience) without relying on data authentication. We show that no single-writer multiple-reader (SWMR) robust atomic storage implementation exists if (a) read operations complete in less than *four* communication round-trips (rounds), and (b) the time-complexity of write operations is constant. More precisely, we present two lower bounds. The first is a read lower bound stating that *three* rounds of communication are necessary to read from a SWMR robust atomic storage. The second is a write lower bound, showing that $\Omega(log(t))$ write rounds are necessary to read in three rounds from such a storage. Applied to known results, our lower bounds close a fundamental gap: we show that time-optimal robust atomic storage can be obtained using well-known transformations from regular to atomic storage and existing time-optimal regular storage implementations.

Categories and Subject Descriptors

C.2.4 [**Computer-Communication Networks**]: Distributed Systems; D.4.1 [**Operating Systems**]: Process Management—*concurrency, multiprocessing / multiprogramming / multitasking, synchronization*; D.4.5 [**Operating Systems**]: Reliability—*Fault-tolerance*

General Terms

Algorithms, Performance, Reliability, Theory

Keywords

Lower bounds, Storage emulations, Arbitrary failures, Optimal resilience, Time-complexity

PODC'11, June 6–8, 2011, San Jose, California, USA.

1. INTRODUCTION

1.1 Background

Variable sharing is critical to modern distributed and concurrent computing. The *atomic* read/write register abstraction [18] is essential to sharing information in distributed systems; it abstracts away the complexity incurred by concurrent access to shared data by providing processes an illusion of sequential access to data. This abstraction is also referred to as atomic *storage*, for its importance as a building-block in practical distributed storage and file systems (see e.g., [24, 25]). Besides, its read/write API, despite being very simple, is today the heart of modern "cloud" key-value storage APIs (e.g., [5]).

In this paper, we study atomic storage implementations in asynchronous message-passing systems in which a set of reader and writer processes (*clients*) share data leveraging a set of storage *object* processes. We consider fault-tolerant, *robust* [3] storage implementations characterized by: a) wait-freedom [17], i.e., the fact that read/write operations invoked by correct clients always eventually return, and b) ensuring correctness despite the largest possible number t of object failures (optimal resilience). We allow for the most general type of failures, arbitrary, also called Byzantine [19] failures[1], without assuming authenticated (also called self-verifying [23]) data to limit the adversary (by relying on e.g., digital signatures).

In this model, we ask a fundamental question: what is the optimal worst-case complexity of robust atomic storage implementations? Our complexity metric is an important one: time-complexity, or *latency*, measured in number of *communication round-trips* (or simply *rounds*) between a client and objects. The relevance of the question we ask extends beyond theoretical. Namely, with the growth in storage outsourcing driven by the advent of cloud computing, the arbitrary failure model becomes increasingly relevant in absence of the full trust in the cloud [6]. In addition, the number of interactions with the remote cloud storage needed to access the data, maps to our latency metric and is often directly associated with the monetary cost; this obviously increases further the practical relevance of the question we ask.

[1] In the Byzantine failure model, optimal resilience corresponds to using $3t + 1$ objects to tolerate t failures [23].

Perhaps surprisingly and despite the wealth of literature exploring latency-optimal storage implementations, this question has not been answered. It is known that the worst-case latency of *writing* into robust storage is at least 2 rounds [1]. In this paper, we show that the optimal worst-case latency of *reading* from *scalable* robust atomic storage is 4 (four) rounds. Here, the notion of scalability captures two basic criteria: a) support for any number of readers, and b) constant write-latency. Our results close a fundamental gap, showing that latency-optimal scalable and robust atomic storage, combining 2-round writes and 4-round reads, can be achieved (in the case of single-writer multi-reader (SWMR) storage) using standard transformations from weaker, regular [18] registers to the atomic ones [4, 20].

Our contribution goes through proving two lower bounds. To help fully appreciate our contributions, we first discuss how the scope of this paper fits into related work.

1.2 Related work

Several papers have explored the time-complexity metric in the context of a read/write register abstraction. A seminal crash-tolerant robust atomic SWMR register implementation assuming a majority of correct processes was presented in [3]. In [3], all write operations complete in a single round; on the other hand, read operations always take two rounds between a client and objects.

The problem of modifying [3] to enable single round reads was explored in [9], which showed that such *fast* atomic implementations are possible albeit they come with the price of limited number of readers and suboptimal resilience. Moreover, the reader in [9] needs to write (i.e., modify the objects' state) as dictated by the lower bound of [12] which showed that every atomic read must write into at least t objects. [10] extends the result of [9] to the Byzantine failure model assuming authenticated (i.e., digitally signed) data and established the impossibility of fast crash-tolerant multi-writer multi-reader (MWMR) atomic register implementations. This result is in line with classical MWMR implementations such as [22] that have read/write latency of at least 2 rounds. The limitation on the number of readers of [9], was relaxed in [13], where a crash-tolerant robust SWMR atomic register implementation was presented, in which most of the reads complete in a single round, yet a fraction of reads is permitted to be slow and complete in 2 rounds.

In the Byzantine context, optimizing latency is particularly interesting when data is assumed to be unauthenticated, which we also assume here. [1] showed that any Byzantine-tolerant storage employing at most $4t$ storage objects has at least some write operation complete in 2 rounds. Moreover, [1] showed a tight lower bound of $t + 1$ rounds from reading from robust SWMR safe [18] storage, with the constraint that readers are precluded from writing. However, allowing readers to write helps improve latency as shown in [15], through a 2-round tight lower bound on reading from robust SWMR *regular* [18] storage. This bound was circumvented in [8], assuming secret values used to detect concurrent operations, where reads are expedited to complete in a single round. However, none of these papers dealt with optimal worst-case latency of reading from robust *atomic* storage, which is precisely the scope of our paper.

On the other hand, few papers have explored the *best-case* complexity of Byzantine-tolerant optimally resilient atomic storage. Here, "best-case" encompasses synchrony, no or few object failures and the absence of read/write concurrency. In this context, [14] presented the first robust atomic storage implementation in which both reads and writes are fast in the best-case (i.e., complete in a single round-trip). Furthermore, [16] considered robust atomic storage implementations with the possibility of having fast reads and writes gracefully degrade to 2 or 3 rounds, depending on the size of the available quorum of correct objects. Unlike these papers, we are interested here with the unconditional, *worst-case* latency of atomic storage.

Finally, the worst-case read latency in existing Byzantine-tolerant robust atomic storage implementations for unauthenticated data (e.g., [2, 14, 16, 23]) is either unbounded or $\Omega(t)$ rounds at best [2].

1.3 Contributions

We present two lower bounds (impossibility results) on time-complexity of reading from robust atomic storage for unauthenticated data, implemented from storage objects prone to Byzantine faults. Together, our lower bounds imply that *there is no* scalable robust atomic storage implementation in the Byzantine unauthenticated model in which all reads complete in less than 4 rounds.

- The first lower bound, referred to as the *read lower bound*, demonstrates the impossibility of reading from robust SWMR atomic storage in two rounds. More precisely, we show that if the number of storage objects S is at most $4t$ and if the number of readers R is greater than 3, then no SWMR atomic implementation may have all reads complete in two rounds.

  Our proof scheme resembles that of [9] and relies on sequentially appending reads on a write operation, while progressively deleting the steps of a write and preceding read operations, exploiting asynchrony and possible failures. This deletion ultimately allows reusing readers and reaching an impossibility with as few as $R = 4$ readers. As none of these appended operations are concurrent under step contention, the impossibility also holds in the stronger data model of [8], in which the adversary is unable to simulate step contention among operations, making use of secret values.

- Our second lower bound, referred to as the *write lower bound*, shows that if read operations are required to complete in three communication rounds, then the number of write rounds k is $\Omega(log(t))$. More precisely, we show that if the number of storage objects is at most $3t + \lfloor t/t_k \rfloor$ and $R \geq k$, then no implementation of a SWMR atomic storage may have all reads complete in three rounds and all writes in $k \leq \lfloor \log(\lceil \frac{3t_k+1}{2} \rceil) \rfloor$ rounds. In a sense, our lower bound generalizes the write lower bound of [1], which proves our result for the special case of $k = 1$.

> While using a similar approach, the write lower bound proof is much more involved and differs from our read lower bound proof in several key aspects. Due to the additional third read round, read steps cannot be entirely deleted, which prohibits the reuse of readers. Consequently, the number of supported readers R and the number of write rounds k are related ($R \geq k$). Furthermore, the proof relies on a set of malicious objects that forges critical steps of the write and of prior reads with respect to subsequent reads. This set grows with the number of appended reads, relating the number of faulty objects t and the number of readers (which is at least k). At the heart of the proof we use a recurrent formula that relates t and k, similar to a Fibonacci sequence, which describes the exact relation between the two parameters. In its closed form, the formula transforms to the log function ($k = \Omega(log(t))$).

The rest of the paper is organized as follows. In Section 2 we give our model and definitions. Sections 3 and gives the proof of our read lower bound. [2] Section 4 gives the proof of our write lower bound. Section 5 concludes the paper by discussing modular implementations that match our lower bounds.

2. MODEL

2.1 Basics

The distributed system we consider consists of three *disjoint* sets of processes: a set *objects* of size S containing processes $\{s_1, ..., s_S\}$ and representing the base register elements; a singleton writer containing a single process $\{w\}$; and a set *readers* of size R containing processes $r_1, ..., r_R$. The set clients is the union of the sets writer and readers. We assume that every client may communicate with any process by message passing using point-to-point reliable communication channels. However, objects cannot communicate among each other, nor send messages to clients other than in reply to clients' messages.

Here we define only the notions we use in our proofs; model details can be found in [20]. A distributed algorithm A is a collection of automata [21], where automaton A_p is assigned to process p. Computation proceeds in steps of A; each step is denoted by a pair of process id and a set of messages received in that step $\langle p, M \rangle$ (M might be $\emptyset$). A run is an infinite sequence of steps of A. A partial run is a finite prefix of some run. A (partial) run r extends some partial run pr if pr is a *prefix* of r. At the end of a partial run, all messages that are sent but not yet received are said to be *in transit*. In any run, any client can fail by crashing and up to t objects may be malicious faulty, exhibiting arbitrary behavior. The non-faulty objects are also called correct. An algorithm that assumes $S = 3t + 1$ is said to be optimally resilient.

2.2 Atomic Storage

A register abstraction is a read/write data structure. It provides two operations: write(v), which stores v in the register, and read(), which returns the value from the register. We assume that each client invokes at most one operation at a time (i.e., does not invoke the next operation until it receives the response for the current operation). Only readers invoke read operations and only the writer invokes write operations. We further assume that the initial value of a register is a special value $\perp$, which is not a valid input value for a write operation. We say that an operation *op* is complete in a (partial) run if the run contains a response step for *op*. In any run, we say that a complete operation op_1 precedes operation op_2 (or op_2 succeeds op_1) if the response step of *op*1 precedes the invocation step of op_2 in that run. If neither op_1 nor op_2 precedes the other, then the operations are said to be concurrent.

An algorithm implements a register if every run of the algorithm satisfies *wait-freedom* and *atomicity* properties. Wait-freedom states that if a process invokes an operation, then eventually, unless that process crashes, the operation completes (even if all other client processes have crashed). Here we give a definition of atomicity for the single-writer registers. In the single-writer setting, the writes in a run have a natural ordering which corresponds to their physical order. Denote by wr_k the k^{th} write in a run ($k \geq 1$), and by val_k the value written by the k^{th} write. Let $val_0 = \perp$. We say that a partial run satisfies atomicity if the following properties hold: (1) if a read returns x then there is k such that $val_k = x$, (2) if a read rd is complete and it succeeds some write wr_k ($k \geq 1$), then rd returns val_l such that $l \geq k$, (3) if a read rd returns val_k ($k \geq 1$), then wr_k either precedes rd or is concurrent with rd, and (4) if some read rd_1 returns val_k ($k \geq 0$) and a read rd_2 that succeeds rd_1 returns val_l, then $l \geq k$.

Time-complexity. We measure the time-complexity of an atomic register implementation in terms of communication round-trips (or simply rounds). A round is defined similar to [9, 11, 13, 22]:

Definition 1. *Client c performs a communication round during operation* op *if the following conditions hold:*

1. *The client c sends messages to all objects. (This is without loss of generality because we can model the fact that messages are not sent to certain objects by having these objects not change their state or reply.)*
2. *Objects, on receiving such a message, reply to the client before receiving any other messages (as dictated by our model).*
3. *When the invoking client receives a sufficient number of such replies, the round (rnd) terminates, and the operation* op *either completes or moves to the next round.*

Note that, since any number of clients can crash, we can construct partial runs in which no client receives any message from any other client. In our proofs in Section 3 and 4 we focus, without loss of generality, on such partial runs.

Since up to t objects might be faulty, ideally, in every round *rnd* the invoking client can only wait for reply messages from

[2] An extension to the model of [26] using distinct thresholds for malicious and crash objects' faults can be found in our full paper [7].

correct objects (at least $S - t$). In fact, we require that if in a partial run pr, a round rnd terminates without the reply from some object s_i, then either (a) s_i is faulty or (b) there is partial run pr' indistinguishable from pr, and in which s_i is faulty.

Each round attempts to invoke operations on *all* objects. If on some correct object s_i there is a pending invocation (of an earlier round), then the new invocation awaits the completion of the pending one. Note that this is equivalent to the round model of [1].

3. THE READ LOWER BOUND

In this section we prove the following proposition.

PROPOSITION 1. *: If $S \leq 4t$ and $R > 3$, then no* read *implementation I of a multi-reader (SWMR) atomic register exists that completes in two rounds.*

3.1 Overview

The idea behind the proof is to start with a complete write that writes 1 into the storage, after which a complete read is appended. By atomicity, the read returns 1. Then, further reads by distinct readers are appended one after the other such that the last appended read returns 1. At the same time, steps of the write and the previous reads are progressively deleted. After appending the fourth read, the final round of the write is deleted from the storage. Moreover, similar to a circular buffer, all steps of the first read are erased, and the read ca be "recycled". By atomicity, the last appended read returns 1. The next iteration starts by reusing the first read, which in turn frees the second read. The proof proceeds through a sequence of such iterations. In each iteration, the last appended read frees the first appended read, and deletes another round of the write. After the last iteration, all steps of the write are deleted, meaning that no write is invoked. However, the last appended read returns 1, violating atomicity.

Preliminaries. In the proof w denotes the writer, r_i for $1 \leq i \leq 4$ denote the readers, and s_i for $1 \leq i \leq S$ denote objects. Suppose by contradiction that $R = 4$ and there is an atomic register implementation I that uses at most $4t$ objects, such that in every partial run of I every *read* operation completes in two rounds.

We partition the set *objects* into four disjoint subsets (which we call *blocks*), denoted B_1,B_2,B_3 and B_4. Blocks B_1, B_2 and B_3 are of size exactly $t \geq 1$ and the size of B_4 is at least 1 and at most t. We refer to the initial state of every correct block B_j as σ_0^j. For simplicity we simply write σ_0, where the block name is implicit.

We say that a round rnd of an operation op *skips* a set of blocks BS in a partial run, (where $BS \subseteq \{B_1, \ldots, B_4\}$), if (1) no object in any block $BL \in BS$ receives any message in round rnd from op in that partial run; (2) all other objects receive all messages in round rnd from op and reply to the messages, and (3) in case round rnd is terminated, the invoking client has received all these reply messages or, in case rnd is not terminated, all these reply messages are in transit. We say that an operation op skips a set of blocks BS in a partial run if every round of op skips BS.

To show a contradiction, we construct a partial run of the implementation I that violates atomicity: a partial run of I in which no value is ever written and some read returns 1.

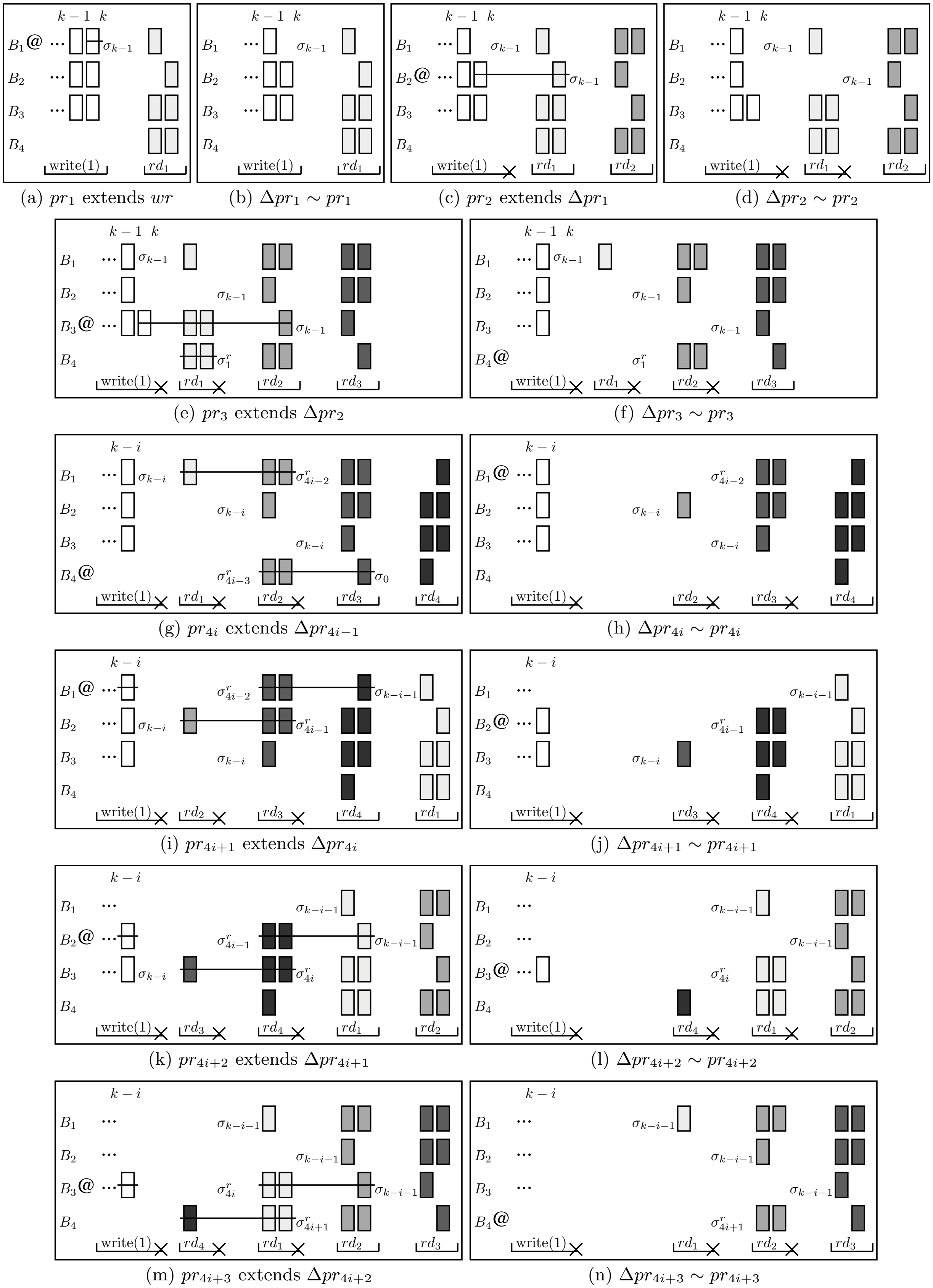
Partial writes. Throughout the proof there is only one write operation write(1) by w that writes value 1. Consider a partial run wr in which w completes write(1) on the register and let k be the number of rounds invoked by w in wr. We denote the state of every correct block B_j after it has replied to the messages of the write during round 1 to i where $1 \leq i \leq k$ as σ_i, where j is again implicit. The write operation skips blocks B_4. We define a series of partial runs containing an incomplete write(1) invocation, each being a prefix of wr. For $1 \leq i \leq k$ and $1 \leq j \leq 4$, we define wr_j^i as the partial run in which (1) rounds 1 to $i-1$ are terminated and skip B_4; (2) round i is not terminated and skips all blocks $\{B_l \mid 1 \leq l \leq j-1\} \cup \{B_4\}$, and (3) all objects are correct. We make two observations: (1) partial run wr_1^k differs from wr only at w and (2) partial run wr_4^1 differs from a run in which write(1) is never invoked only at w.

Block diagrams. We illustrate the proof in Figure 1 (a)-(n). We depict a round rnd of an operation op through a set of rectangles arranged in a single column. In the column corresponding to some round rnd of op we draw a rectangle in a given row, if all objects in the corresponding block BL have received the message from the client in round rnd of op and have sent reply messages, i.e., if round rnd of op does not skip BL. We write "@" in the row corresponding to BL iff BL is malicious.

Appending reads. Partial run pr_1 extends wr by appending a complete read rd_1 by r_1 that skips B_2 in round one and B_1 in round two (see Figure 1 (a)). Note that when the second round is started, there is a pending first round invocation on B_2. Therefore in the second round, rd_1 waits for both first *and* second round replies from B_2. For ease of presentation, the late first round replies are not illustrated.

In pr_1, all objects in block B_1 are malicious, and forge their state to σ_{k-1} before replying to rd_1. By atomicity rd_1 returns 1. Observe that r_1 cannot distinguish pr_1 from some partial run Δpr_1 that extends wr_2^k by appending rd_1, and where all objects are correct (see Figure 1 (b)). Note that Δpr_1 is obtained from pr_1 by deleting the crossed steps.

Partial run pr_2 extends Δpr_1 by appending a complete read rd_2 by r_2 that skips B_3 and B_2 in round one and two respectively (see Figure 1 (c)). In pr_2, all objects in block B_2 are malicious, and forge their state to σ_{k-1} before replying to rd_2. By atomicity rd_2 returns 1. Observe that r_2 cannot distinguish pr_2 from some partial run Δpr_2, that extends wr_3^k by appending an incomplete rd_1 and a complete rd_2, and where all objects are correct (Figure 1 (d)). Δpr_2 is obtained from pr_2 by deleting the crossed steps.


$k-1$ k
B_1 @ ⋯ σ_{k-1}
B_2 ⋯
B_3 ⋯
B_4
write(1) rd_1
B_2 @ σ_{k-1}
write(1) rd_1 rd_2
B_3 @ σ_1^r
write(1) rd_1 rd_2 rd_3
B_4 @
$k-i$
σ_{k-i} σ_{4i-2}^r σ_{4i-3}^r σ_0
write(1) rd_1 rd_2 rd_3 rd_4
σ_{k-i-1} σ_{4i-1}^r
write(1) rd_2 rd_3 rd_4 rd_1
σ_{4i}^r
write(1) rd_3 rd_4 rd_1 rd_2
σ_{4i+1}^r
write(1) rd_4 rd_1 rd_2 rd_3

(a) pr_1 extends wr

(b) $\Delta pr_1 \sim pr_1$

(c) pr_2 extends Δpr_1

(d) $\Delta pr_2 \sim pr_2$

(e) pr_3 extends Δpr_2

(f) $\Delta pr_3 \sim pr_3$

(g) pr_{4i} extends Δpr_{4i-1}

(h) $\Delta pr_{4i} \sim pr_{4i}$

(i) pr_{4i+1} extends Δpr_{4i}

(j) $\Delta pr_{4i+1} \sim pr_{4i+1}$

(k) pr_{4i+2} extends Δpr_{4i+1}

(l) $\Delta pr_{4i+2} \sim pr_{4i+2}$

(m) pr_{4i+3} extends Δpr_{4i+2}

(n) $\Delta pr_{4i+3} \sim pr_{4i+3}$

Figure 1: Illustration of the runs used in the proof of Proposition 1 ($1 \leq i \leq k-1$)

Partial run pr_3 extends Δpr_2 by appending a complete read rd_3 by r_3 that skips B_4 in round one and B_3 in round two (Figure 1 (e)). In pr_3, all objects in block B_3 are malicious, and forge their state to σ_{k-1} before replying to rd_3. By atomicity rd_3 returns 1. Let σ_1^r denote the state of the objects in block B_4 in run pr_3 before replying to rd_2. Observe that r_3 cannot distinguish pr_3 from some partial run Δpr_3, that extends wr_4^k by appending incomplete reads rd_1 and rd_2 and a complete read rd_3 and in which (1) all objects in B_4 are malicious and (2) they forge their state to σ_1^r before replying to rd_2 (Figure 1 (f)).

Note that in pr_3, rd_3 completes the second round based on replies from all correct objects, and similarly in Δpr_3, the first round misses replies only from faulty objects. Since r_3 cannot distinguish pr_3 and Δpr_3, it cannot wait for additional replies (in any of the two runs).

Partial run pr_4 (illustrated in Figure 1 (g)) extends Δpr_3 by appending a complete read rd_4 by r_4 that skips B_1 in round one and B_4 in round two. In pr_4, all objects in block B_4 are malicious and forge their state (1) to σ_1^r before replying to rd_2 and (2) to σ_0 before replying to rd_4. By atomicity rd_4 returns 1. Let σ_2^r denote the state of the objects in block B_1 before replying to rd_3. Observe that r_4 cannot distinguish pr_4 from some partial run Δpr_4, that extends wr_1^{k-1} by appending incomplete reads rd_2, rd_3 and a complete read rd_4, and in which (1) all objects in B_1 are malicious and (2) they forge their state to σ_2^r before replying to rd_3 (Figure 1 (h)). Note that in partial run pr_4, rd_4 receives second round replies from all correct objects. Similarly in Δpr_4, rd_4 receives first round replies from all objects except the faulty ones. Since r_4 cannot distinguish pr_4 and Δpr_4, rd_4 cannot wait for additional replies without violating termination.

After appending rd_4 and constructing Δpr_4 by deleting all steps from pr_4 which are not visible to rd_4, we notice that we have erased all steps in column k of write(1) as well as, deleted all steps of rd_1. Thus, we can recycle r_1 by appending rd_1 again and start deleting the steps in column $k-1$.

Starting from Δpr_4 we iteratively define the following partial runs for $1 \leq i \leq k-1$ and $1 \leq j \leq 4$ (see Figure 1 (g)-(n)). Partial run pr_{4i+j} extends Δpr_{4i+j-1} by appending rd_j. In pr_{4i+j}, all objects in block B_j are malicious and they forge their state (1) to $\sigma^r_{4i+(j-3)}$ before replying to rd_{j-2}[3] and (2) to $\sigma_{((j \mod 4)/j)(k-i-1)}$ before replying to rd_j. Let $\sigma^r_{4i+(j-2)}$ denote the state of the objects in block $B_{(j \mod 4)+1}$ before replying to rd_{j-1}. Observe that r_j cannot distinguish pr_{4i+j} from some partial run Δpr_{4i+j}, that extends $wr^{k-i}_{(j \mod 4)+1}$ by appending incomplete reads rd_{j-2} and rd_{j-1} and a complete read rd_j, and in which (1) all objects in $B_{(j \mod 4)+1}$ are malicious, and (2) they forge their state to $\sigma^r_{4i+(j-2)}$ before replying to rd_{j-1} (Figure 1 (h),(j),(l),(n)). In run Δpr_{4i+j} and pr_{4i+j}, rd_j receives first and second round replies from all correct objects respectively. Since r_j cannot distinguish Δpr_{4i+j} and pr_{4i+j}, rd_j cannot wait for additional replies without blocking.

Read rd_4 in Δpr_4 returns 1. Since pr_5 extends Δpr_4 by appending rd_1, by atomicity, rd_1 in pr_5 returns 1. However, as r_1 cannot distinguish pr_5 from Δpr_5, rd_1 in Δpr_5 returns 1. In general, since pr_{4i+j} extends Δpr_{4i+j-1} by appending rd_j (for $1 \leq i \leq k-1$ and $1 \leq j \leq 4$), and r_j cannot distinguish pr_{4i+j} from Δpr_{4i+j}, it follows by induction that rd_j in Δpr_{4i+j} returns 1. In particular, rd_3 reads 1 in Δpr_{4k-1}. By our construction, Δpr_{4k-1} extends wr_4^1 and wr_4^1 is indistinguishable from a run in which write(1) is never invoked. Hence, rd_3 returns 1 even if no write is invoked, violating atomicity. □

4. THE WRITE LOWER BOUND

In this section we prove the following proposition.

Proposition 2. *: If $S \leq 3t + \lfloor t/t_k \rfloor$ and every read completes in three rounds then no* write *implementation I of a multi-reader atomic register exists that completes in $\min\{R, \lfloor \log(\lceil (3t_k+1)/2 \rceil) \rfloor\}$ rounds.*

We first prove the following key lemma. In the effort of making its involved proof easier to follow we first proceed through a careful proof setup that we found worthwhile. To further help follow the proof, we also visualize runs we use in the proof in Figure 2.

Lemma 1. *Let $k \geq 1$, $t_{-1} = t_0 = 0$ and $t_k = t_{k-1} + 2t_{k-2} + 1$. There is no implementation I of a k-reader atomic storage with $3t_k + 1$ objects and t_k faults such that the* write *completes in k rounds and the* read *completes in three rounds.*

Preliminaries. Recall that w denotes the writer, r_i for $1 \leq i \leq k$ denote the readers, and s_i for $1 \leq i \leq S$ denote the objects. The initial value of the register is $\perp$. In the proof, there is only one write operation write(1) by w that writes value 1. We know from [1] that the lemma is true for $k = 1$; hence, we assume $k \geq 2$. Suppose by contradiction that there is an implementation I that uses at most $3t_k + 1$ objects, such that in every partial run of I every write (resp., read) completes in k (resp. 3) rounds.

We partition the set *objects* into $2k+2$ distinct blocks, $B_0, \ldots, B_{k+1}$ and $C_1, \ldots, C_k$ such that $|\bigcup_{j=0}^{k+1} B_j| = 2t_k + 1$ and $|\bigcup_{j=1}^{k} C_j| = t_k$. Block B_0 contains a single object. For $1 \leq l \leq k$, the size of B_l is $t_l - t_{l-2}$ and the size of B_{k+1} is $2t_k + 1 - |\bigcup_{j=0}^{k} B_j| = t_k - t_{k-1}$. For $1 \leq l \leq k-1$, the size of C_l is $t_{l-1} - t_{l-2}$ and the size of C_k is $t_k - |\bigcup_{j=1}^{k-1} C_j| = t_k - t_{k-2}$. It is important to note that C_1 is empty. Towards a uniform presentation of the result, we will refer to C_1 wherever appropriate. Also, we use the abbreviation $BL_{i,j}$ to denote the set $\{BL_i, BL_j\}$, for some $BL \in \{B, C\}$.

We also define three sets of blocks called *superblocks*: the "malicious" superblock $\mathcal{M}_l$, the "parity" superblock $\mathcal{P}_l$ and the "correct" superblock $\mathcal{C}_l$. Superblock $\mathcal{M}_l$ contains all blocks with index at most l. Formally, for $-1 \leq l \leq k-1$ we define $\mathcal{M}_l := \{B_j \mid 0 \leq j \leq l\} \cup \{C_j \mid 1 \leq j \leq l\}$. For

[3] Please note that when we write rd_{j-c}, we always mean $rd_{4-((c-j) \mod 4)}$.

instance, $\mathcal{M}_{-1} = \emptyset$ and $\mathcal{M}_2 = \{B_0, B_1, C_1, C_2\}$. Superblock $\mathcal{P}_l$ contains all blocks B_j with index $j \geq l \geq 1$ such that j and l have the same parity. More formally, for $1 \leq l \leq k$, we define $\mathcal{P}_l := \{B_j \mid l \leq j \leq k+1 \ \wedge \ j \equiv (l \mod 2)\}$. For instance, if k is even then $\mathcal{P}_1 = \{B_1, B_3, \ldots, B_{k-1}, B_{k+1}\}$ and $\mathcal{P}_2 = \{B_2, B_4, \ldots, B_{k-2}, B_k\}$. Finally, superblock $\mathcal{C}_l := \{C_j \mid l \leq j \leq k\}$.

Given the size of the individual blocks, we can determine the cardinality of the union of all elements of a superblock. Namely, if $\mathcal{S} \in \{\mathcal{M}_l, \mathcal{P}_l, \mathcal{C}_l\}$, then we define the union of its elements as $\bigcup \mathcal{S} = \{s \in BL \mid BL \in \mathcal{S}\}$. Having in mind that $t_k = t_{k-1} + 2t_{k-2} + 1$ (Def.) and $t_{-1} = t_0 = 0$, we have:

$$|\bigcup \mathcal{M}_l| = t_l + 2t_{l-1} + 1 \overset{(Def.)}{=} t_{l+1} \qquad \text{for } 0 \leq l \leq k-1 \quad (1)$$

$$|\bigcup \mathcal{P}_l| = t_k - t_{l-2} \quad \text{for } \ 1 \leq l \leq k+1 \qquad (2)$$

$$|\bigcup \mathcal{C}_l| = t_k - t_{l-2} \quad \text{for } \ 1 \leq l \leq k \qquad (3)$$

Block diagrams. Figure 2 illustrates the proof for $R = k = 4$. Reader r_i invokes read rd_l, $1 \leq l \leq k$. In the column corresponding to some round *rnd* of *op* we draw a rectangle in a given row, iff round *rnd* of *op* does not skip[4] the corresponding block BL. We write "@" in the row of BL iff BL is malicious.

Read patterns. We first characterize a *complete* read rd_l for $1 \leq l \leq k-1$. A complete rd_l skips (1) $\mathcal{M}_{l-2} \cup \mathcal{P}_{l+1}$ in round one and two, and (2) $\mathcal{M}_{l-2} \cup \mathcal{C}_{l+1}$ in round three. Read rd_k skips $\mathcal{M}_{k-2} \cup \mathcal{P}_{k+1}$. Observe that by equations (1), (2) and (3), a read skips exactly t_k objects in each round.

Consider the example in Figure 2. Complete reads rd_1, rd_2 and rd_3 skip (respectively): (1) $\{B_{2,4}\}$, $\{B_0\} \cup \{B_{3,5}\}$ and $\{B_{0,1}\} \cup \{B_4\}$ in rounds one and two, and (2) $\{C_{2,3,4}\}$, $\{B_0\} \cup \{C_{3,4}\}$ and $\{B_{0,1}\} \cup \{C_4\}$ in round three. Read rd_4 skips $\{B_{0,1,2}, C_2\} \cup \{B_5\}$.

We further define three types of *incomplete* reads *inc1*, *inc2* and *inc3*, depending on the read's progress. For $1 \leq l \leq k$, read rd_l is of type *inc1* if the first round is not terminated and skips all blocks *except* $\mathcal{P}_l$. For $1 \leq l \leq k-1$, read rd_l is of type (1) *inc2* if the first round is terminated, and the second round is not terminated and skips all blocks *except* $\mathcal{C}_l$, and (2) *inc3* if the second round is terminated and the third round is not terminated and skips $\mathcal{M}_{l-2} \cup \mathcal{C}_{l+1} \cup \mathcal{P}_{l+1}$.

Consider our example in Figure 2 (c) that illustrates partial run Δpr_2 (after deleting the crossed out steps). Observe that (1) rd_2 is incomplete of type *inc3* (its third round skips $\{B_0\} \cup \{C_{3,4}\} \cup \{B_{3,5}\}$), (2) rd_1 is incomplete of type *inc2* (its second round skips all blocks except $\{C_{2,3,4}\}$) and (3) rd_3 (resp., rd_4) is incomplete of type *inc1*; its first round skips all blocks except $\{B_{3,5}\}$ (resp., $\{B_4\}$).

Towards a contradiction, we construct a partial run of the atomic register implementation I that violates atomicity. More specifically, we exhibit a partial run in which some read returns a value that was never written.

Initialization. Consider a partial run pr_{init} in which (1) all blocks are correct and (2) pr_{init} extends the empty run by appending incomplete reads rd_l by r_l of type *inc1*, for $1 \leq l \leq k$, one after the other. In pr_{init}, there is no write operation. We refer to the state of each correct block $BL \in \mathcal{P}_l$ after replying to rd_l as σ_0^l. Thus, the state of B_l at the end of pr_{init} corresponds to σ_0^l for $1 \leq l \leq k$. Further, B_{k+1} is in state σ_0^{k-1}. To see why, note that B_{k-1} and B_{k+1} have the same parity and there are only k reads.

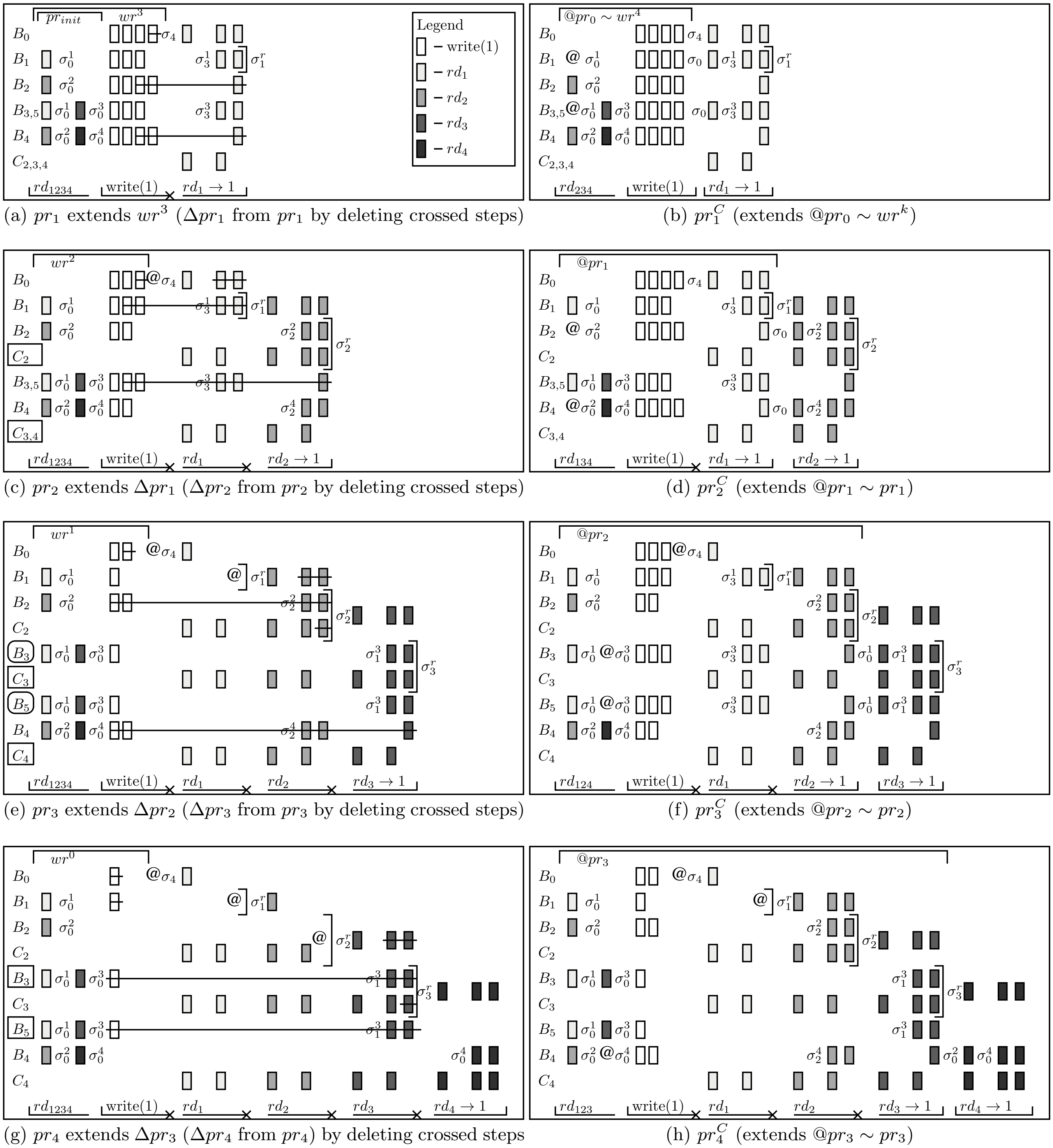
Consider our example Figure 2 (a). At the end of pr_{init}, block B_1 (resp., B_2; $B_{3,5}$; B_4) replied to rd_1 (resp., rd_2; rd_1 and rd_3; rd_2 and rd_4); thus, at the end of the run its state is σ_0^1 (resp., σ_0^2; σ_0^3; σ_0^4).

Partial writes. We extend pr_{init} to a partial run wr^k by appending a complete write(1) that completes in k rounds and skips superblock $\mathcal{C}_1$. Moreover, we define a series of partial runs each being a prefix of wr^k. For $1 \leq i \leq k$, let wr^{k-i} be the partial run which extends pr_{init} by appending an incomplete write(1) such that (i) round 1 to $k-i$ are terminated and (ii) round $k-i+1$ is not terminated and skips $\mathcal{C}_1$ and all B_j's such that $j > 0$ and i and j have the same parity, i.e., $\mathcal{C}_1 \cup \mathcal{P}_{2-(i \mod 2)}$ (Fig. 2 (a) and (c)). We refer to the state of the blocks $B_l \in \mathcal{P}_{2-(i \mod 2)}$ at the end of wr^{k-i} as σ_{k-i}^l for $1 \leq l \leq k$. If $B_{k+1} \in \mathcal{P}_{2-(i \mod 2)}$, then we refer to its state at the end of wr^{k-i} as σ_{k-i}^{k-1}. Note here that σ_{k-i}^l results from σ_0^l by appending $k-i$ rounds of the write. When the context is clear, for simplicity we refer to these states using the implicit notation σ_{k-1}^*. Finally, we refer to the state of B_0 at the end of runs wr^k and wr^{k-1} as σ_k.

We refer to our example in Figure 2 (a),(c),(e) and (g) for illustrations of the runs wr^3 to wr^0 and the corresponding states. For instance Figure 2 (a), illustrates wr^3 as an extension of pr_{init}. The states of the blocks B_0, B_1 and $B_{3,5}$ at the end of wr^3 are σ_4 (4 rounds of write), σ_3^1 and σ_3^3 (3 rounds of write).

Appending Reads. Partial run pr_1 extends wr^{k-1} by appending the missing steps of a complete read rd_1. In pr_1 all objects are correct and thus rd_1 receives replies from $S - t_k$ correct objects. After receiving the third round replies, rd_1 completes and returns value x. We now show that $x = 1$. We define a partial run $@pr_0$, (Fig. 2(b)) which is *identical* to wr^k except that in $@pr_0$ (1) no read by r_1 occurs and (2) superblock $\mathcal{P}_1$ is malicious and mimics the occurrence of rd_1 by forging its initial state to σ_0^1. By equation (1), the malicious objects in $@pr_0$ amount to t_k. Partial run pr_1^C (Fig. 2(b)) is defined as an extension of $@pr_0$ by appending a complete read rd_1. Read rd_1 cannot distinguish pr_1^C from pr_1 because $\mathcal{P}_1$, which is malicious in pr_1^C, mimics pr_1. Specifically, $\mathcal{P}_1$ forges its state to σ_0 before replying to rd_1's first round, and then to σ_{k-1}^* before replying to rd_1's second

[4] The definition of *skipping* extends here from Sec. 3.



(a) pr_1 extends wr^3 (Δpr_1 from pr_1 by deleting crossed steps)

(b) pr_1^C (extends $@pr_0 \sim wr^k$)

(c) pr_2 extends Δpr_1 (Δpr_2 from pr_2 by deleting crossed steps)

(d) pr_2^C (extends $@pr_1 \sim pr_1$)

(e) pr_3 extends Δpr_2 (Δpr_3 from pr_3 by deleting crossed steps)

(f) pr_3^C (extends $@pr_2 \sim pr_2$)

(g) pr_4 extends Δpr_3 (Δpr_4 from pr_4) by deleting crossed steps

(h) pr_4^C (extends $@pr_3 \sim pr_3$)

Figure 2: Instance of the proof with $k = 4$.

round. In pr_1^C, by atomicity rd_1 returns 1. Since pr_1^C and pr_1 are indistinguishable to reader r_1, $x = 1$.

Next, we define partial run Δpr_1 obtained from pr_1 by deleting the steps of the read and the write as illustrated in Figure 2 (a). More specifically, Δpr_1 extends wr^{k-2} by appending the missing steps of an incomplete read rd_1 of type $inc3$, after which rd_1 crashes. In Δpr_1, $\mathcal{M}_0 = \{B_0\}$ is malicious and forges its state to σ_k before replying to rd_1. Observe that at the end of pr_1 and Δpr_1, every correct block is in the same state, except $\mathcal{P}_2$. We refer to the state of B_1 at the end of Δpr_1 as σ_1^r.

Starting from Δpr_1 we iteratively define the following partial runs for $2 \leq l \leq k$ (see Fig. 2). Partial run pr_l extends Δpr_{l-1} by appending the missing steps of a complete read rd_l. In pr_l, superblock $\mathcal{M}_{l-2}$ is malicious and all other blocks are correct. Since rd_l does not receive any messages from $\mathcal{M}_{l-2}$, it completes only on the basis of replies from correct objects (at least $S - t_k$ by equation (1)). At the end of pr_l, rd_l completes and returns value x. To show that $x = 1$, we define a partial run $@pr_{l-1}$ which is identical to pr_{l-1} except that in $@pr_{l-1}$ (1) there is no read by r_l and (2) and (in addition to $\mathcal{M}_{l-3}$), superblock $\mathcal{P}_l$ is malicious and forges its state to σ_0^l, simulating the occurrence of rd_l as in pr_{l-1}. The count of malicious objects in $@pr_{l-1}$ is exactly t_k. To see why, notice that by equation (1) and (2) the malicious objects in $@pr_{l-1}$ amount to $|\bigcup \mathcal{P}_l| + |\bigcup \mathcal{M}_{l-3}| = t_k - t_{l-2} + t_{l-2} = t_k$.

Then, partial run pr_l^C extends $@pr_{l-1}$ by appending rd_l. Note that rd_l cannot distinguish pr_l^C from pr_l because superblock $\mathcal{P}_l$, which is malicious in pr_l^C, mimics pr_l. In particular, $\mathcal{P}_l$ forges its state to σ_0 before replying to rd_l's first round and then to σ_{k-l}^* before replying to rd_l's second round. By atomicity, rd_l returns 1 in pr_l^C. Since pr_l^C and pr_l are indistinguishable to reader r_l, $x = 1$.

Next, we define partial run Δpr_l. For $2 \leq l < k$, Δpr_l is obtained from pr_l by deleting steps of rd_l, rd_{l-1} and the write (see Fig. 2 (c) and (e)). In Δpr_l, superblock $\mathcal{M}_{l-1}$ is malicious, all other block are correct, and blocks $\{B_{l-1}, C_{l-1}\} \in \mathcal{M}_{l-1}$ forge their state to σ_j^r before replying to rd_l.[5] In more detail, Δpr_l extends wr^{k-l-1} by appending the missing steps (1) of incomplete reads $rd_1, \ldots, rd_{l-1}$ of type *inc2*, and (2) of an incomplete rd_l of type *inc3*. B_0 forges its state to σ_k before replying to rd_1 and for $1 \leq j \leq l-1$, $\{B_j, C_j\}$ forge their state to σ_j^r before replying to rd_{j+1}. Observe that at the end of pr_l and Δpr_l, every correct block is in the same state, except $\mathcal{P}_{l+1}$. We refer to the state of $\{B_l, C_l\}$ at the end of Δpr_1 as σ_l^r.

Finally, partial run Δpr_k is obtained analogously from pr_k, except that in Δpr_k, (a) no write is invoked and (b) read rd_k is complete and skips $\mathcal{M}_{k-2} \cup \mathcal{P}_{k+1}$ (see Fig. 2 (g) for $k = 4$). In particular, in Δpr_k, $\mathcal{M}_{k-1}$ is malicious and blocks $\{B_{k-1}, C_{k-1}\} \in \mathcal{M}_{k-1}$ forge their state to σ_{k-1}^r before replying to rd_k. By equation (1) the malicious objects amount to $|\bigcup \mathcal{M}_{k-1}| = t_k$. Partial runs pr_k and Δpr_k differ only at $\mathcal{P}_{k+1}$, and rd_k completes without receiving any message from $\mathcal{P}_{k+1}$. Thus, rd_k cannot distinguish Δpr_k from pr_k and returns 1 in Δpr_k, a contradiction, as no write was invoked. □

LEMMA 2. *: If $S \leq 3t + 1$ and every read completes in three rounds then no* write *implementation I of a multi-reader (SWMR) atomic register exists that completes in* $\min\{R, \lfloor \log(\lceil (3t+1)/2 \rceil) \rfloor\}$ *rounds.*

PROOF. Let $k = \min\{R, \lfloor \log(\lceil (3t+1)/2 \rceil) \rfloor\}$, i.e., $R \geq k$ and $k \leq \lfloor \log(\lceil (3t+1)/2 \rceil) \rfloor$. By Lemma 1, there exists no optimally resilient k-reader atomic register implementation with $t_k = t_{k-1} + 2t_{k-2} + 1$ faulty objects, where the read completes in three rounds and the write completes in k rounds. Observe that this is valid even with $R \geq k$ readers and $t \geq t_k$ faults. Writing t_k in closed form results in $t_k = \frac{1}{6}(2^{k+2} - (-1)^k - 3)$. Thus, we have that $t \geq \frac{1}{6}(2^{k+2} - (-1)^k - 3)$. Solving for k results in $k \leq \lfloor \log(\lceil (3t+1)/2 \rceil) \rfloor$. □

Finally, we generalize our result to a resilience of $3t + \lfloor t/t_k \rfloor$ for $t \geq t_k$, proving Proposition 2.

PROOF. Without loss of generality we can assume that $t \geq t_k$ because every implementation is subject to the resilience lower bound of $S \geq 3t + 1$. The observation is that if we multiply each of the blocks in the proof of Lemma 1 with a constant c, then the result holds for $S' = cS = 3ct_k + c$ objects and ct_k faults. By carefully choosing $c = t/t_k$, we obtain a lower bound proof for $S' = 3t + \lfloor t/t_k \rfloor$ and t faults. □

5. CONCLUSION

In this paper, we show that no single-writer multiple-reader (SWMR) robust atomic storage implementation exists if (a) read operations complete in less than *four* communication round-trips (rounds), and (b) the time-complexity of write operations is constant.

However, we observe that a matching implementation can simply be obtained by a) reusing the SWMR regular storage implementation of [15] which features the worst-case time complexity of 2 rounds for both reads and writes, and b) transforming it to the SWMR atomic implementations using a standard SWMR regular – SWMR atomic transformation technique [4, 20].[6] This yields a sought SWMR atomic implementation in which write operations complete in 2 rounds whereas reads complete in 4 rounds.

Furthermore, in the stronger authentication model that allows for secret values [8], regular storage of [15] can be replaced in the above transformation with the corresponding time-optimal regular implementation [8], yielding a 2-round write 3-round read atomic storage, which is optimal in this model. In both models, multi-writer atomic storage can be implemented by applying the standard transformations further [4, 20].

In summary, we present two lower bounds. The first is a read lower bound stating that *three* rounds of communication are necessary to read from a SWMR robust atomic storage. The second is a write lower bound, showing that $\Omega(log(t))$ write rounds are necessary to read in three rounds from such a storage. Our results close a fundamental gap: we show that time-optimal, 2-round write 4-round read (resp. 3-round read in the secret value model) robust atomic storage can be obtained using well-known transformations from regular to atomic storage and existing time-optimal regular storage implementations.

[5] The states are different and are indexed by the object's id, which for simplicity of presentation is made implicit.

[6] In short, this transformation employs $R + 1$ regular registers, one dedicated to the writer and R additional ones, one per reader, in which a given reader writes back the read value.

6. REFERENCES

[1] Ittai Abraham, Gregory Chockler, Idit Keidar, and Dahlia Malkhi. Byzantine disk paxos: optimal resilience with byzantine shared memory. *Distributed Computing*, 18(5):387–408, 2006.

[2] Amitanand S. Aiyer, Lorenzo Alvisi, and Rida A. Bazzi. Bounded wait-free implementation of optimally resilient byzantine storage without (unproven) cryptographic assumptions. In *Proceedings of the 21st International Symposium on Distributed Computing*, pages 7–19, September 2007.

[3] Hagit Attiya, Amotz Bar-Noy, and Danny Dolev. Sharing memory robustly in message-passing systems. *Journal of the ACM*, 42(1):124–142, 1995.

[4] Hagit Attiya and Jennifer Welch. *Distributed Computing. Fundamentals, Simulations, and Advanced Topics*. McGraw-Hill, 1998.

[5] AWS Simple Storage Service. http://aws.amazon.com/s3/.

[6] Christian Cachin, Idit Keidar, and Alexander Shraer. Trusting the cloud. *SIGACT News*, 40(2):81–86, 2009.

[7] Dan Dobre, Rachid Guerraoui, Matthias Majuntke, Neeraj Suri, and Marko Vukolić. The Complexity of Robust Atomic Storage. Technical Report TR-TUD-DEEDS-06-01-2010, 2010.

[8] Dan Dobre, Matthias Majuntke, Marco Serafini, and Neeraj Suri. Efficient robust storage using secret tokens. In *Proceedings of the 11th International Symposium on Stabilization, Safety, and Security of Distributed Systems*, pages 269–283, 2009.

[9] Partha Dutta, Rachid Guerraoui, Ron R. Levy, and Arindam Chakraborty. How fast can a distributed atomic read be? In *Proceedings of the 23rd annual ACM symposium on Principles of distributed computing*, pages 236–245, July 2004.

[10] Partha Dutta, Rachid Guerraoui, Ron R. Levy, and Marko Vukolic. Fast access to distributed atomic memory. *SIAM J. Comput.*, 39(8):3752–3783, 2010.

[11] Burkhard Englert, Chryssis Georgiou, Peter M. Musial, Nicolas C. Nicolaou, and Alexander A. Shvartsman. On the efficiency of atomic multi-reader, multi-writer distributed memory. In *Proceedings of the 13th International Conference on Principles of Distributed Systems*, pages 240–254, 2009.

[12] Rui Fan and Nancy Lynch. Efficient replication of large data objects. In *Proceedings of the 17th International Symposium on Distributed Computing*, pages 75–91, October 2003.

[13] Chryssis Georgiou, Nicolas C. Nicolaou, and Alexander A. Shvartsman. Fault-tolerant semifast implementations of atomic read/write registers. *J. Parallel Distrib. Comput.*, 69(1):62–79, 2009.

[14] Rachid Guerraoui, Ron R. Levy, and Marko Vukolić. Lucky read/write access to robust atomic storage. In *Proceedings of the International Conference on Dependable Systems and Networks*, pages 125–136, 2006.

[15] Rachid Guerraoui and Marko Vukolić. How fast can a very robust read be? In *Proceedings of the twenty-fifth annual ACM symposium on Principles of distributed computing*, pages 248–257, New York, NY, USA, 2006. ACM.

[16] Rachid Guerraoui and Marko Vukolić. Refined quorum systems. In *Proceedings of the twenty-sixth annual ACM symposium on Principles of distributed computing*, pages 119–128, 2007.

[17] Maurice Herlihy. Wait-free synchronization. *ACM Transactions on Programming Languages and Systems*, 13(1):124–149, January 1991.

[18] Leslie Lamport. On interprocess communication. *Distributed computing*, 1(1):77–101, May 1986.

[19] Leslie Lamport, Robert E. Shostak, and Marshall C. Pease. The Byzantine generals problem. *ACM Transactions on Programming Languages and Systems*, 4(3):382–401, 1982.

[20] Nancy A. Lynch. *Distributed Algorithms*. Morgan-Kaufmann, 1996.

[21] Nancy A. Lynch and Mark R.Tuttle. An introduction to input/output automata. *CWI Quarterly*, 2(3):219–246, 1989.

[22] Nancy A. Lynch and Alexander A. Shvartsman. Rambo: A reconfigurable atomic memory service for dynamic networks. In *Proceedings of the 16th International Conference on Distributed Computing*, pages 173–190, London, UK, 2002. Springer-Verlag.

[23] Jean-Philippe Martin, Lorenzo Alvisi, and Michael Dahlin. Minimal Byzantine storage. In *Proceedings of the 16th International Conference on Distributed Computing*, pages 311–325, October 2002.

[24] Yasushi Saito, Svend Frolund, Alistair Veitch, Arif Merchant, and Susan Spence. Fab: building distributed enterprise disk arrays from commodity components. *SIGOPS Oper. Syst. Rev.*, 38(5):48–58, 2004.

[25] Frank Schmuck and Roger Haskin. GPFS: A shared-disk file system for large computing clusters. In *Proceedings of the 1st USENIX Conference on File and Storage Technologies*, pages 231–244, Berkeley, CA, USA, 2002. USENIX Association.

[26] Philip M. Thambidurai and You-Keun Park. Interactive consistency with multiple failure modes. In *Symposium on Reliable Distributed Systems*, pages 93–100, 1988.