On the Efficiency of Atomic Multi-Reader, Multi-Writer Distributed Memory

(Regular Submission)

Burkhard Englert [*]
benglert@csulb.edu

Chryssis Georgiou [†]
chryssis@cs.ucy.ac.cy

Peter M. Musial [‡§]
peter.musial@uprrp.edu

Nicolas Nicolaou [¶]
nicolas@engr.uconn.edu

Alexander A. Shvartsman[¶‖]
aas@cse.uconn.edu

Abstract

This paper considers quorum-replicated, multi-writer, multi-reader (MWMR) implementations of survivable atomic registers in a distributed message-passing system with processors prone to failures. Previous implementations in such settings invariably required two rounds of communication between readers/writers and replica owners. Hence the question arises whether it is possible to have single round read and/or write operations in this setting.

As a first step, we present an algorithm, called CWFR, that allows the classic two round write operations, while supporting *single round* read operations. Since multiple write operations may be concurrent with a read operation, this algorithm involves an iterative (local) discovery of the latest completed write operation. This algorithm precipitates the question of whether fast (single round) writes may co-exist with fast reads. We thus devise a second algorithm, called SFW, that exploits a new technique called *server side ordering* (SSO), which –unlike previous approaches– places partial responsibility for the ordering of write operations on the replica owners (the servers). With SSO, *fast* write operations are introduced for the very first time in the MWMR setting. While this is possible, we show that under certain conditions the MWMR model imposes inherent limitations on any quorum-based fast write implementation of a *safe read/write register* and potentially even restricts the number of writer participants in the system. In this case our second algorithm achieves near optimal efficiency. Both algorithms are proved to preserve atomicity in all permissible executions.

Keywords: Atomic memory, multi-reader/multi-writer, quorums, message passing, upper/lower bounds

Contact Author: Nicolas Nicolaou
Computer Science and Engineering, Unit 2155
University of Connecticut, Storrs, CT 06268, USA

[*] Computer Engineering and Computer Science, California State University Long Beach
[†] Department of Computer Science, University of Cyprus
[‡] Department of Computer Science, University of Puerto Rico - Rio Piedras, USA
[§] Part of this work was performed while the author was affiliated with: Comp. Sci. Dept., Naval Postgraduate School, USA.
[¶] Computer Science and Engineering, University of Connecticut
[‖] Work supported in part by the NSF awards 0702670, 0121277, and 0311368.

1 Introduction

Data survivability is essential in distributed systems. Replication is broadly used to sustain critical data in networked settings prone to failures, and a variety of distributed storage systems have been designed to replicate and maintain data residing at distinct network locations or servers. Together with replication come the problems of maintaining consistency among the replicas and of efficiency of access to data, all in the presence of failures.

A long string of research has been addressing the consistency challenge by devising efficient, wait-free, atomic (linearizable [21]) read/write sharable objects in message-passing systems (e.g., [1, 2, 3, 5, 7, 14, 15, 19, 20, 23, 26]). An atomic read/write object, or register [22], provides the semantics of a sequentially accessed single object. The underlying implementations replicate the object at several failure-prone servers and allow concurrent reading and writing by failure-prone clients. The *efficiency* of read or write operations is measured in terms of the number of communication rounds between clients and servers.

In their pioneering work, Attiya et al. [3] presented an implementation of an atomic Single-Writer, Multiple-Reader (SWMR) object in the message-passing model. The registers are replicated at the servers where any minority of them may crash. The write protocol involves a single communication round-trip between the writer and the servers, while the read protocol requires two communication rounds, where in the second round the readers essentially perform a write of the value obtained in the first round. Each round includes communication with a majority of servers. Building on [3], Lynch and Shvartsman [26, 23], presented an atomic reconfigurable implementation if Multiple-Writer, Multiple-Reader (MWMR) objects, with majorities generalized to quorums, where both read and write operations take two communication rounds (in the absence of reconfiguration).

A common theme in such solutions is that read operations have two phases, where the second phase in essence performs a write. This contributed to a folklore belief that "atomic reads must write". This belief is reexamined by Dutta et al. [7], who present a wait-free atomic SWMR register implementation where *both* read and write operations take a *single* communication round. This *fast* access (i.e., single round) is traded for a restriction on the number of readers in terms of a bound involving the number of servers and the number of server crashes. To allow a large number of readers, a *semifast* implementation was proposed [15] that pays for the unbounded number of readers with at most a single *slow* read operation (i.e., two-round) for every write operation. It was also shown that it is not possible to have fast or semifast implementations for MWMR registers.

Thus the following questions arises: *Under what conditions may one obtain efficient atomic read/write register implementations in the MWMR model?* Answering this question is the objective of this work.

Prior Work. In the SWMR model, Attiya et al. [3] achieve consistency by exploiting the intersecting sets of majorities in combination with $\langle timestamp, value \rangle$ pairs, comprised of a logical clock and the associated replica value. A write operation increments the writer's local timestamp and delivers the new timestamp-value pair to a majority of servers, taking one round. A read operation obtains timestamp-value pairs from some majority, then propagates the pair corresponding to the highest timestamp to some majority of servers, thus taking two rounds.

The majority-based approach in [3] is readily generalized to quorum-based approaches (e.g., [26, 8, 23, 9, 18]). In this context, a *quorum system* [16, 29, 11, 28, 27] is a collection of subsets of server identifiers with pairwise non-empty intersections. The work of [9] shows that the read operations must write to as many replica servers as the maximum number of failures allowed. A dynamic atomic memory implementation using reconfigurable quorums is given in [23] (with several practical refinements in [17, 12, 13, 4]), where the sets of servers can arbitrarily change over time as processes join and leave the system. Retargeting this work to ad-hoc mobile networks, Dolev *et al.* [6] formulated the GeoQuorums approach. There (and in [4]), some reads involve a single communication round when it is confirmed that the corresponding write operation has completed.

Dutta et al. [7] present the first *fast* atomic SWMR implementation where all operations take a *single* communication round. They show that fast behavior is achievable only when the number of reader processes R is inferior to $\frac{S}{t} - 2$, where S the number of servers, t of whom may crash. They also showed that fast implementations in the MWMR model are impossible in the presence of a single server failure. Georgiou et al. [15] introduced the notion of *virtual nodes* that enables an unbounded number of readers. They define the notion of *semifast* implementations were only a single read operation per write needs to be "slow" (take two rounds). Their algorithm requires that the number of virtual nodes V is inferior to $\frac{S}{t} - 2$; this does not prevent multiple readers

as long as at least one virtual node exists. They also show that semifast MWMR implementations are impossible.

Other works, e.g., [1, 19, 20, 14], pursue bounds on the efficiency of distributed storage in a variety of organizational and failure models. For example, [19, 1], explore conditions under which two round operations are required by safe and regular SWMR registers.

Recently quorum-based approaches were further explored in the context of efficient atomic registers [20, 14]. Guerraoui and Vukolić [20] defined the notion of *Refined Quorum Systems* (RQS), where quorums are classified in three categories, according to their intersection size with other quorums. The authors characterize these properties and develop an efficient Byzantine-resilient SWMR atomic object implementation and a solution to the consensus problem. In synchronous failure-free runs their implementation is fast. Georgiou et al. [14] specified the properties that a general quorum system must possess in order to achieve single round operations in the presence of crashes and asynchrony. They showed that fast and semifast quorum-based SWMR implementations are possible iff a common intersection exists among all quorums, hence a single point of failure exists in such solutions (i.e., any server in the common intersection), making such implementations not robust. To trade efficiency for improved fault-tolerance, *weak-semifast* implementations are introduced in [14] that require at least one single slow read per write operation, and where all writes are fast. In addition, they present a client-side prediction tool called *Quorum Views* that enables fast read operations in general quorum-based implementations even under read/write concurrency. Simulation results demonstrated the effectiveness of this approach, showing that a small fraction of read operations need to be slow under realistic scenarios. A question that naturally follows is whether it is possible, and under what conditions, to have weak-semifast atomic MWMR register implementations.

Contributions. Our goal is to identify the conditions under which it is possible to obtain atomic register implementations that allow single round write and read operations in the Multi-Writer, Multi-Reader (MWMR) model. We adopt previous techniques to improve the efficiency of read operations and we incorporate new techniques that enable single communication round, i.e., fast, write operations. Our contributions are as follows.

1. We present algorithm CWFR that employs the *Quorum Views* technique of [14] in an iterative manner for discovering the latest completed write operation among concurrent writes. This enables fast read operations. Indeed this algorithm shows that fast read operations are possible in the MWMR setting.
2. To enable *fast write* operations we introduce a new technique called *Server Side Ordering (SSO)* that assigns to the server processes the responsibility of maintaining and incrementing logical timestamps, that are used by both readers and writers and helps to ensure atomicity. Previous algorithms, including algorithm CWFR, placed this responsibility on the writer's side. (In the presence of asynchrony and failures, SSO alone does not suffice to guarantee atomicity: using SSO by itself may result in the generation of non unique timestamps for each write operation.)
3. We developed a quorum-based implementation for atomic MWMR registers, called SFW, that (a) employs the SSO technique by having the servers assign logical timestamps to writes. and (b) ensures uniqueness of timestamps by combining them with writer-generated (locally) logical timestamps. This hybrid approach guarantees uniqueness of tags among the read and write participants for every written value and allows the writers and readers to reason about the state of the system. To the best of our knowledge, this is the *first* MWMR atomic register implementation that provides the possibility of fast reads *and* writes.
4. Lastly, we develop a framework for reasoning about impossibility and lower bounds for MWMR implementations. An *n-wise* quorum system is such where any n quorums have a common non-empty intersection. We call two operations *consecutive* if they are complete, not concurrent, and originate at two distinct processes. Two operations are *quorum shifting* if they are consecutive and the two originating processes receive replies from two distinct quorums during these operations. We prove lower bounds on the number of consecutive, quorum shifting fast write operations that an execution of a safe register implementation may contain. We show that a safe register implementation is impossible in an n-wise quorum system, where not all quorums have a common intersection, if any execution contains more than $n - 1$ consecutive, quorum shifting single round write operations. This ultimately implies that in an implementation with only fast writes there cannot be more than $n - 1$ writers. Algorithm SFW is nearly optimal since it approaches this bound as it yields executions with up to $n/2$ consecutive fast write operations, while maintaining atomicity.

Document Structure. In Section 2 we present our model and definitions. In Section 3 we describe the algorithm CWFR, which contains slow write and fast read operations. Our second algorithm, SFW, is presented in Section 4. The inherent limitations of MWMR model and the conditions under which it is possible to obtain fast write operations, are presented in Section 5. We conclude with discussion in Section 6.

2 Model and Definitions

We consider the asynchronous message-passing model. There are three distinct finite sets of processors: a set of readers $\mathcal{R}$, a set of writers $\mathcal{W}$, and a set of $\mathcal{S}$ servers. The identifiers of all processors are unique and comparable. Communication among the processors is accomplished via reliable communication channels. Servers are arranged into intersecting sets, or *quorums*, that together form a quorum system $\mathbb{Q}$. For a set of quorums $\mathcal{A} \subseteq \mathbb{Q}$ we denote the intersection of the quorums in $\mathcal{A}$ by $I_{\mathcal{A}} = \bigcap_{Q \in \mathcal{A}} Q$. We define specializations of quorum systems based on the number of quorums that together have a non-empty intersection.

Definition 2.1 *A quorum system $\mathbb{Q}$ is called an **n-wise quorum system** if for any $\mathcal{A} \subseteq \mathbb{Q}$, s.t. $|\mathcal{A}| = n$ we have $I_{\mathcal{A}} \neq \emptyset$ holds. We call n the **intersection degree** of $\mathbb{Q}$.*

In a common quorum system any two quorums intersect, and so any quorum system is a *2-wise* (pairwise) quorum system. At the other extreme, a $|\mathbb{Q}|$*-wise* quorum system has a common intersection among all quorums. From the definition it follows that an *n-wise* quorum system is also a *k-wise* quorum system, for $2 \leq k \leq n$. We will organize the servers into n-$wise$ quorum systems known to all the participants as needed.

Algorithms presented in this work are specified in terms of *I/O automata* [25, 24], where an algorithm A is a composition of automata A_i, each assigned to some process i. Each A_i is defined in terms of a set of states $states(A_i)$ that includes the initial state σ_0, a signature $sig(A_i)$ that specifies input, output, and internal actions (external signature consists of only input and output actions), and *transitions*, that for each action ν gives the triple $\langle \sigma, \nu, \sigma' \rangle$ defining the transition of A_i from state σ to state σ'. Such a triple is also called a *step*. An *execution fragment* ϕ of A_i is a finite or an infinite sequence $\sigma_0, \nu_1, \sigma_1, \nu_2, \ldots, \nu_r, \sigma_r, \ldots$ of alternating states and actions, such that every $\sigma_k, \nu_{k+1}, \sigma_{k+1}$ is a step of A_i. If an execution fragment begins with an initial state of A_i then it is called an *execution*. We say that an execution fragment ϕ' of A_i, *extends* a finite execution fragment ϕ of A_i if the first state of ϕ' is the last state of ϕ. The concatenation of ϕ and ϕ' is the result of the extension of ϕ by ϕ' where the duplicate occurrence of the last state of ϕ is eliminated, yielding an execution fragment of A_i.

A process i *crashes* in an execution ϕ if it contains a step $\langle \sigma_k, fail_i, \sigma_{k+1} \rangle$ as the last step of A_i. A process i is *faulty* in an execution ϕ if i crashes in ϕ; otherwise i is *correct*. A quorum $Q \in \mathbb{Q}$ is non-faulty if $\forall i \in Q$, i is correct; otherwise Q is faulty. We assume that at least one quorum in $\mathbb{Q}$ is non-faulty in any execution.

Atomicity. We aim to implement atomic read/write memory, where each object is replicated at servers. Each object has a unique name, x from some set X, and object values v come from some set V_x; initially each x is set to a distinguished value v_0 ($\in V_x$). Reader p requests a read operation ρ on an object x using action $\mathsf{read}_{x,p}$. Similarly a write operation is requested using action $\mathsf{write}(*)_{x,p}$ at writer p. The steps corresponding to such actions are called *invocation* steps. An operation terminates with the corresponding $\mathsf{read\text{-}ack}(*)_{x,p}$ or $\mathsf{write\text{-}ack}_{x,p}$ action; these steps are called *response* steps. An operation π is *incomplete* in an execution when the invocation step of π does not have the associated response step; otherwise we say that π is *complete*. We assume that requests made by read and write processes are *well-formed*: a process does not request a new operation until it receives the response for a previously invoked operation.

In an execution, we say that an operation (read or write) π_1 *precedes* another operation π_2, or π_2 *succeeds* π_1, if the response step for π_1 precedes in real time the invocation step of π_2; this is denoted by $\pi_1 \rightarrow \pi_2$. Two operations are *concurrent* if neither precedes the other.

Correctness of an implementation of an atomic read/write object is defined in terms of the *atomicity* and *termination* properties. Assuming the failure model discussed earlier, the termination property requires that any operation invoked by a correct process eventually completes. Atomicity is defined as follows [24]. For any

execution of a memory service, if all the read and the write operations that are invoked complete, then the read and write operations can be partially ordered by an ordering $\prec$, so that the following properties are satisfied:

P1. The partial order is consistent with the external order of invocation and responses, that is, there do not exist operations π_1 and π_2, such that π_1 completes before π_2 starts, yet $\pi_2 \prec \pi_1$.
P2. All write operations are totally ordered and every read operation is ordered with respect to all the writes.
P3. Every read operation ordered after any writes returns the value of the last write preceding it in the partial order, and any read operation ordered before all writes returns the initial value of the object.

In the sequel we assume a single register memory system. By composing multiple single register implementations, one may obtain a complete atomic memory [24]. Thus, we omit further mention of object names.

Efficiency and Fastness. We measure the efficiency of an atomic register implementation in terms of *communication round-trips* (or simply rounds). A round is defined as follows [7, 15, 14]:

Definition 2.2 *Process p performs a communication round during operation π if all of the following hold:*
1. p sends request messages that are a part of π to a set of processes,
2. any process q that receives a request message from p for operation π, replies without delay.
3. when process p receives enough replies it terminates the round (either completing π or starting new round).

Operation π is *fast* [7] if it completes after its first communication round; an implementation is fast if in each execution all operations are fast. *Semifast* implementations as defined in [15] allow some read operations to perform two communication rounds. Briefly, an implementation is semifast if the following properties are satisfied: (a) writes are fast, (b) reads complete in one or two rounds, (c) only a single complete read operation is slow (two round) per write operation, and (d) there exists an execution that contains at least one write and one read operation and all operations are fast. Finally, *weak-semifast* implementations [14] satisfy properties (a), (b), and (d), but eliminate the property (c), allowing multiple slow read operations per write. As shown in [7, 15] no MWMR implementation of atomic memory can be fast or semifast. So we focus our attention on implementations where both reads and writes maybe slow. We use quorum systems and tags to maintain, and impose an ordering on, the values written to the register replicas. We say that a quorum $Q \in \mathbb{Q}$, *replies* to a process p for an operation π during a round, if $\forall s \in Q$, s receives messages during the round and replies to these messages, and p receives all of the replies.

Given that any subset of readers or writers may crash, the termination of an operation cannot depend on the progress of any other operation. Furthermore we guarantee termination only if servers' replies within a round of some operation do not depend on receipt of any message sent by other processes. Thus we can construct executions where only the messages from the invoking processes to the servers, and from the servers to the invoking processes are delivered. Lastly, to guarantee termination under the assumed failure model, no operation can wait for more than a singe quorum to reply within the processing of a single round.

3 Algorithm CwFr

We begin the study of the efficiency of atomic MWMR register implementations with algorithm CwFr. The algorithm employs two techniques, (i) the classic query and propagate technique (two round) for write operations, and (ii) analysis of Quorum Views [14] for potentially fast (single round) read operations. This allows us to maintain two-round communication complexity of write operations, while enabling fast read operations.

The algorithm uses $\langle tag, value \rangle$ pairs, to impose ordering on the values written to the register. A tag is a tuple of the form $\langle ts, wid \rangle \in \mathbb{N} \times \mathcal{W}$, where ts is the timestamp and wid is a writer id. Two tags are ordered lexicographically, first by the timestamp, then by the writer id. Initially tags are set to $\langle 0, \min(\mathcal{W}) \rangle$ for all processes. We use $maxTag$ to denote the maximum tag that a client process observes in the replies it receives from a quorum of servers for the operation that it invoked.

The *quorum views* technique [14], provides sufficient information about the distribution of the latest tag in the accessed quorum and allows read operations to decide locally whether a second round is needed. Quorum views are defined as follows in terms of tags:

Definition 3.1 *Assume that for a read or write operation π invoked by process p, each server s from some quorum $Q \in \mathbb{Q}$ replies to p with the tag $s.tag$. Let $maxTag = \max_{s \in Q}(s.tag)$. We say that p observes one of the following* ***quorum views*** *for Q:*

$qView(1)$*:* $\forall s \in Q : s.tag = maxTag$,
$qView(2)$*:* $\forall Q' \in \mathbb{Q} : Q \neq Q' \wedge \exists A \subseteq Q \cap Q'$, *s.t.* $A \neq \emptyset$ *and* $\forall s \in A : s.tag < maxTag$,
$qView(3)$*:* $\exists s' \in Q : s'.tag < maxTag$ *and* $\exists Q' \in \mathbb{Q}$ *s.t.* $Q \neq Q' \wedge \forall s \in Q \cap Q' : s.tag = maxTag$

Summarizing the above definition, $qView(1)$ requires that all servers in some quorum reply with the same tag. $qView(3)$ reveals that some servers in the quorum contain an older value, but there exists an intersection where all of its servers contain the new value. Finally $qView(2)$ is the negation of the other two views, revealing a quorum where the new value is neither distributed to the full quorum or distributed fully in any of its intersections.

We now give a high level description of our implementation. For paucity of space, the Input/Output Automata specification and the proofs of correctness are given in Appendix A.

Servers. The servers play a passive role. They receive read or write requests, update their object replica accordingly, and reply to the process that originated the request. In additional detail, upon receipt of any message, the server compares its local tag with the tag included in the message. If the tag of the message is higher than its local tag, the server adopts the higher tag along with its corresponding value. Once this is done the server replies to the invoking process.

Writers. The write protocol has two rounds. During the first round the writer discovers the maximum tag among the servers: It sends read messages to all servers and waits for a quorum to reply. Once it receives replies from a complete quorum, it discovers the maximum tag among the replies. The writer increments this maximum timestamp and proceeds to the second round, where it propagates the new tag along with the value to be written. Once the writer receives replies from a complete quorum, the write completes.

Readers. The read protocol is more involved. When a reader invokes a read operation, it sends a read message to all servers and waits for some quorum to reply. Once a quorum replies, the reader determines the $maxTag$ and stores it locally. Then the reader analyzes the distribution of the tag information within the responding quorum Q in an attempt to determine the latest, potentially complete, write operation. This is accomplished by determining the quorum view conditions. Detecting conditions of $qView(1)$ and $qView(3)$ are straightforward. When condition for $qView(1)$ is detected, the read completes and the value associated with the discovered $maxTag$ is returned. In the case of $qView(3)$ the reader continues into the second round, advertising the latest tag ($maxTag$) and its associated value. Analysis of $qView(2)$ involves discovery of the earliest completed write operation, and this is done iteratively by (locally) removing the servers from Q that replied with the largest tags. After each iteration the reader determines the next largest tag in the remaining server set, and then re-examines the quorum views in the next iteration. This process eventually leads to either $qView(1)$ or $qView(3)$ being observed. If $qView(1)$ is observed, then the read completes in a single round with the maximum tag among the servers that remain in Q and returns the associated value. If $qView(3)$ is observed, then the reader proceeds to the second round as above, and upon completion it returns the value associated with the maximum tag discovered among the original respondents in Q.

Let us discuss the idea behind our technique. Observe that under our failure model, any write operation can expect a response from at least one full quorum. Moreover a write ω distributes its tag τ_ω to some quorum, say Q_i, before completing.Thus when a read operation ρ, s.t. $\omega \rightarrow \rho$, receive replies from some quorum Q_j, then it will observe one of the following tag distributions: (a) if $Q_j = Q_i$, then $\forall s \in Q_j, \tau_s = \tau_\omega$ ($qView(1)$), or (b) if $Q_j \neq Q_i$, then $\forall s \in Q_i \cap Q_j, \tau_s = \tau_\omega$ ($qView(3)$). Hence, if ρ observes a distribution as in $qView(1)$ then it follows that a write operation completed and received replies from the same quorum that replied to ρ. Alternatively, if only an intersection contains a uniform tag (i.e., the case of $qView(3)$) then there is a possibility

that some write completed in an intersecting quorum (in this example Q_i). The read operation is fast in $qView(1)$ since it is determinable that the write potentially completed. The read proceeds to the second round in $qView(3)$, since the completion of the write is indeterminable and it is necessary to ensure that any subsequent operation will observe that tag. If none of the previous quorum views hold (and thus $qView(2)$ holds), then it must be the case that the write that yielded the maximum tag is not yet completed. Hence we try to discover the latest potentially completed write by removing all the servers with the highest tag from Q_j and repeating the analysis. If at some iteration, $qView(1)$ holds on the remaining tag values, then a potentially completed write (that was overwritten by greater values in the rest of the servers) is discovered and that tag is returned. If no iteration is interrupted because of $qView(1)$, then eventually $qView(3)$ will be observed, in the worst case, when a single server will remain in some intersection of Q_j. Since a second round cannot be avoided in this case, we take the opportunity to propagate the largest tag observed in Q_j. At the end of the second round that tag will be written to at least as single complete quorum and thus the reader can safely return it.

Theorem 3.2 *Algorithm* CWFR *implements a MWMR atomic read/write register.*

Proof. [Sketch] We first show that the iterative application of the quorum views reveals the smallest tag potentially associated with a completed write operation, and that no other read operation will perceive a different smallest tag. With this we are able to claim consistency between the read operations and show that if a read operation ρ_1 returns a tag τ and a read operation ρ_2 succeeds ρ_1 (i.e. $\rho_1 \rightarrow \rho_2$) and returns a tag τ', then $\tau' \geq \tau$. Furthermore it allows us to show that if a write operation ω precedes a read operation ρ (i.e., $\omega \rightarrow \rho$) then ρ returns tag τ higher or equal to the tag associated with the write operation. Lastly the fact that the write operations always performs two rounds, implies the uniqueness of the tags associated with write operations and we show that if there are two write operations s.t. $\omega_1 \rightarrow \omega_2$ and tag τ_1 is associated with ω_1 and τ_2 with ω_2 then it must be the case that $\tau_2 > \tau_1$. This reasoning is accompanied by the proof of tag monotonicity, completing the proof (contained in the optional appendix). □

4 SSO-based Algorithm

In this section we present algorithm SFW that utilizes a technique, called SSO, to introduce fast read *and* write operations. In traditional MWMR atomic register implementations (including algorithm CWFR), the writer is solely responsible for incrementing the tag that imposes the ordering on the values of the register. With the new technique, and our hybrid approach, this task is now also assigned to the servers, hence the name *Server Side Ordering*. Algorithm SFW involves two predicates. One for the write protocol and one for the read protocol. Both protocols evaluate the distribution of a tag within the quorum that replies to a write/read operation respectively. A formal specification in the form of IOA and the detail proof of correctness, can be found in Appendix B.2.

The algorithm uses $\langle tag, value \rangle$ pairs, to impose ordering on the values written to the register. In contrast with the traditional approach where the tag is a two field tuple, this algorithm requires the tag to be a triple. In particular the tag is of the form $\langle ts, wid, wc \rangle \in \mathbb{N} \times \mathcal{W} \times \mathbb{N}$, where the fields ts and wid are used as in common tags and represent the timestamp and writer identifier respectively. Field wc represents the write operation counter and facilitates the ability to distinguish between write operations. Initially the tag is set to $\langle 0, \min(\mathcal{W}), 0 \rangle$ in every process. In contrast to ts, wc is incremented by the writer before invokes a write operation and it denotes the sequence number of that write. Recall that by our key technique, the tags (and particularly the timestamps in the tags) are incremented by the server processes. Thus if a tag was a tuple of the form $\langle ts, wid \rangle$, then two server processes s_i and s_j may associate two different tags $\langle ts_i, w \rangle$ and $\langle ts_j, w \rangle$ to a single write operation. Any operation however that witness such tags cannot distinguish whether the tags refer to a single or different write operations from w. By introducing the new term the tags will become $\langle ts_i, w, wc \rangle$ and $\langle ts_j, w, wc \rangle$, and thus any operation establishes that the same write operation was assigned two different timestamps. The triples can be compared alphanumerically. In particular we say that $tag_1 > tag_2$ if $tag_1.ts > tag_2.ts$, or $tag_1.ts = tag_2.ts \wedge tag_1.wid > tag_2.wid$, or $tag_1.ts = tag_2.ts \wedge tag_1.wid = tag_2.wid \wedge tag_1.wc > tag_2.wc$.

Server. The server maintains the register replica and acts depending on the message it receives. The local state of a server process s, is defined by three local variables: (1) the tag_s variable which is the local tag stored in the server, (2) the $confirmed_s$ variable which stores the largest tag known by s that has been returned by some reader or writer process and, (3) the $inprogress_s$ set which includes all the latest tags assigned by s to write requests. Each server s waits to receive a read or write message from operation initiated at some process p. Where this message contains: (a) the type of the message $msgType$, (b) the last tag returned by p ($msgtag$), (c) the value to be written if p invokes a write operation or the latest value returned by p if p invokes a read operation, (d) a counter $msgwc$ that specifies the sequence number of this operation if p invokes a write or is equal to $msgtag.wc$ if p invokes a read, and (e) a counter to help the server distinguish between new and stale messages from p. Upon receipt of any type of message, s updates its local and confirmed tags if they are smaller than the tag enclosed in the received message. In particular if $msgtag > tag_s$ (resp. $msgtag > confirmed_s$) then s assigns $tag_s = msgtag$ (resp. $confirmed_s = msgtag$). In addition to the above updates, if s receives a WRITE message from p, then s generates a new tag $newt = \langle tag_s.ts + 1, p, msgwc \rangle$, by incrementing the timestamp included in its local timestamp by 1 and assigning the new timestamp to the write operation from p. Note that the new tag generated is greater than both tag_s and $msgtag$. The server then pairs the new tag to the value included in the write message and changes its local tag to $tag_s = newt$. Then s adds $newt$ in the $inprogress_s$ set, and removes any tag maintained previously in that set for any write operation from p. Once it completes its local update, s acknowledges every message received by sending its $inprogress_s$ set and $confirmed_s$ variable to the requesting process.

Writer. To uniquely identify all write operations, a writer w maintains a local variable wc that is incremented each time w invokes a write operation. Essentially that variable counts the number of write operations performed by w and every such write can be identified by the tuple $\langle w, wc \rangle$, by any process in the system. To perform a write operation $\omega = \langle w, wc \rangle$, w sends messages to all of the servers and waits for a quorum of these, Q, to reply. Once enough replies arrive (each server's $inprogress$ set and $confirmed$ variable), w collects all of the tags assigned to ω by each server in Q. Then it applies a predicate on the collected tags. In few words the predicate is used to checks if any of the collected tags appear in some intersection of Q with at most $\frac{n}{2} - 1$ (see proof sketch below why this is sufficient) other quorums, where n the intersection degree of the deployed quorum system. If there exists such a tag τ then the writer adopts τ as the tag of the value it tried to write; otherwise the writer adopts the maximum among the collected tags in the replied quorum. The writer proceeds in a second communication round to propagate the tag assigned to the written values if: (a) the predicate holds but the tag is only propagated in an intersection of Q with more than $\frac{n}{2} - 2$ other quorums, or (b) the predicate does not hold. In any other case the write operation is fast and completes in a single communication round. More formally the writer predicate is the following, where $|A|$ is rounded down to the nearest integer:

> **Writer predicate for a write ω (PW):** $\exists\, \tau, A, MS$ where: $\tau \in \{\langle ., \omega \rangle : \langle ., \omega \rangle \in inprogress_s(\omega) \ \wedge \ s \in Q\}$, $A \subseteq \mathbb{Q}, 0 \leq |A| \leq \frac{n}{2} - 1$, and $MS = \{s : s \in Q \ \wedge \ \tau \in inprogress_s(\omega)\}$, s.t. either $|A| \neq 0$ and $I_A \cap Q \subseteq MS$ or $|A| = 0$ and $Q = MS$.

Reader. The main difference between reader and writer protocols is that the reader has to examine each tag assigned to *all* of the write operations contained in $inprogress$ sets of the servers that replied. (In contrast, writer examines only the tags assigned only to its own write operation.)

The reader proceeds by sending messages to all the servers and waits for some quorum of these to reply. Soon as enough replies arrive, it computes the maximum confirmed tag $maxConf$, and populates the set inP with all tags from $inprgoress$ set reported by each of the replying servers. Then the reader chooses the largest tag $maxT$ found in inP and checks if: (a) $maxConf \geq maxT$, or (b) whether $maxT$ satisfies a reader predicate (defined below). If neither condition is valid, then $maxT$ tag is removed from inP and $maxT$ is assigned the next largest tag in inP, then the two checks are repeated. If inP becomes empty, then $maxConf$ is returned along with its associated value. If (a) holds, then some tag that has already been returned by some process is higher than any remaining tag in inP. In this case reader returns $maxConf$ and its assigned value. If (b) holds,

then reader returns the tag and the associated value that satisfies its predicate. The reader is requires to ensure that the tag is propagated in an intersection between the replied quorum and at most $\frac{n}{2}-2$ other quorums, where n the intersection degree of the quorum system. A read operation is slow and performs a second communication round if: (1) the predicate holds but the tag is propagated in an intersection between Q and exactly $\frac{n}{2}-2$ other quorums, or (2) the reader decides to return $maxConf$, but this was received from no complete intersection or an intersection between Q and $n-1$ other quorums. The tag and the associated value that will be returned by the read operation are propagated to some quorum of servers during the second communication round. More formally the reader predicate is, where $|B|$ is rounded down to the nearest integer:

> **Reader predicate for a read ρ (PR):** $\exists\, \tau, B, MS$, where: $\max(\tau) \in \bigcup_{s \in Q_i} inprogress_s(\rho)$, $B \subseteq \mathbb{Q}, 0 \leq |B| \leq \frac{n}{2}-2$, and $MS = \{s : s \in Q_i \ \wedge\ \tau \in inprogress_s(\rho)\}$, s.t. either $|B| \neq 0$ and $I_B \cap Q_i \subseteq MS$ or $|B| = 0$ and $Q_i = MS$.

We provide a sketch of correctness proof for algorithm SFW. (Omitted details can be found in Appendix B.2.)

Theorem 4.1 *Algorithm* SFW *implements a MWMR atomic read/write register.*

Proof. [Sketch] The key challenge is to show that every reader and writer process decide on a single unique tag for each write operation, despite the fact that servers may assign different tags to that same write operation. To this end, we first show that in an n-wise quorum system, if some process p obtains replies from the servers of some quorum, then p may witness only a single tag per write operation to be distributed in a k-wise intersection, for $k < \frac{n+1}{2}$.

Writer's perspective: Based on the above observation, we show that only a single (unique) tag may satisfy the write predicate (**PW**). Observe that if there is a tag τ that satisfies **PW**, then it follows that τ is distributed in an intersection of at most $\frac{n}{2}$ quorums (i.e. $\frac{n}{2}$-wise intersection, including the replying quorum); otherwise, if no tag satisfies **PW** then the write operation is associated with the unique maximum tag received by the writer.

Reader's perspective: The goal is to show that if a read ρ returns a tag τ for a write ω, then τ was also the tag assigned to ω by the writer that invoked ω. Observe that a read operation returns a tag τ for a write ω in two cases: (a) τ satisfied the reader predicate **PR**, or (b) τ was equal to the max confirmed tag. In the first case the predicate ensures that τ was distributed in an intersection of at most $\frac{n}{2}-1$ quorums (including the replying quorum). Thus the writer should have observed the tag in at least an $\frac{n}{2}$-wise intersection, and hence τ would satisfy the writer's predicate as well. Furthermore, τ should be the only tag that satisfies the two predicates since it is distributed in an intersection that consists of less than $\frac{n+1}{2}$ quorums. If the reader returns τ because of case (b) then it follows that τ was confirmed by either a reader that was about to return τ because it satisfied its predicate, or by the writer that decided to associate τ with its write operation ω. Either way this was a unique tag and thus returning the confirmed tag maintains its uniqueness.

Using the proof of uniqueness of a tag assigned to a write operation, we proceed to show that the atomic properties are satisfied. In particular we show the following: (1) the monotonicity of the tag in all participants, (2) that if a write operation proceeds a read operation then the read returns a tag greatest or equal to the one associated to the write operation, (3) If a write $\omega_1 \rightarrow \omega_2$ then ω_2 is associated with a higher tag than the tag associated to ω_1, and (4) If there are two read operation s.t. $\rho_1 \rightarrow \rho_2$ then ρ_2 decides and returns a value associated with a higher or equal tag than the one returned by ρ_1. □

5 Write Optimality

We now investigate the conditions under which it is possible for an execution of a MWMR register implementation to contain only fast write operations. In particular we show that by exploiting an $n\text{-}wise$ quorum system, it is possible to have executions with only fast write operations iff a certain number of "consecutive" write operations are contained in the execution. In extend, we show that this result imposes bounds on the number of writer participants in the system. For space limitations some proofs appear in Appendix C.

We consider all operations that alter the tag value at some set of servers to be write operations. In an execution, an operation π invoked by process p is said to *contact* a subset of servers $\mathcal{G} \subseteq \mathcal{S}$, denoted by $cont_p(\mathcal{G}, \pi)$, if for every server $s \in \mathcal{G}$: (a) s receives the messages sent by p within π, (b) s replies to p, and (c) p receives the reply from s. If $cont_p(\mathcal{G}, \pi)$ occurs and additionally no other server (i.e., $s \notin \mathcal{G}$) receives any message from p within π then we say that π *strictly contacts* $\mathcal{G}$, and is denoted by $scnt_p(\mathcal{G}, \pi)$. Next we give two important definitions.

Definition 5.1 *Two operations π_1, π_2 are **consecutive** in an execution if: (i) they are invoked from processes p_1 and p_2, s.t. $p_1 \neq p_2$, (ii) they are complete, and (iii) $\pi_1 \rightarrow \pi_2$ or $\pi_2 \rightarrow \pi_1$ (they are not concurrent).*

In lieu to the above definition, a *safe register* constitutes the weakest consistency guarantee in the chain, and is defined [22] as property **S1**: *Any read operation that is not concurrent to any write operation returns the value written by the last preceding write operation.*

Definition 5.2 *A set of operations Π in an execution is called **quorum shifting** if $\forall \pi_1, \pi_2 \in \Pi$ strictly contact quorums $Q', Q'' \in \mathbb{Q}$ respectively, then π_1 and π_2 are consecutive and $Q' \neq Q''$.*

Given the two definitions above, we now show the ensuing lemma.

Lemma 5.3 *A read operation that succeeds a set of fast write operations Π, may retrieve the latest written value only from the servers that received messages from all the write operations in Π.*

Given an *n-wise* quorum system we show that if there are $n-1$ consecutive, quorum shifting fast write operations in an execution then safe register implementations are possible.

Lemma 5.4 *Any execution fragment ϕ of a safe register implementation that uses an n-wise quorum system $\mathbb{Q}$ s.t. $2 \leq n < |\mathbb{Q}|$, contains at most $n-1$ consecutive, quorum shifting, fast write operations for any number of writers $W \geq 2$.*

We now show that safe register implementations are not possible if we extend any execution that contains $n-1$ consecutive writes, with one more consecutive, quorum shifting write operation. It suffices to assume a very basic system consisting of two writers w_1 and w_2, and one reader r. Thus our results hold for at least two writers.

Theorem 5.5 *No execution fragment ϕ of a safe register implementation that exploits an n-wise quorum system $\mathbb{Q}$ s.t. $2 \leq n < |\mathbb{Q}|$, can contain more than $n-1$ consecutive, quorum shifting, fast write operations for any number of writers $W \geq 2$.*

Proof. Let $\mathbb{Q}$ be an *n-wise* quorum system, for $2 \leq n < |\mathbb{Q}|$. From Lemma 5.4 we obtain that an implementation exploiting an *n-wise* quorum system may contain $n-1$ consecutive, quorum shifting fast write operations and still preserve property S1. Thesis of this proof follows from the contradiction, where we assume that an implementation can include n consecutive fast writes and still satisfy property S1.

Let $\mathbb{Q}$ be an $(k+2)$*-wise* system and let ξ_k be an execution of the safe register implementation that exploits $\mathbb{Q}$. Suppose the execution follows the construction in Lemma 5.4. It follows that ξ_k contains $k+1$ consecutive, quorum shifting, fast writes. Moreover by the induction we know that ξ_k satisfies safe register property if extended by a read operation. Let us now extend ξ_k with a write $\omega(k+2)$ from writer $w_{(k+1 \mod 2)+1}$ with $scnt_{w_{(k+1 \mod 2)+1}}(Q_{k+2}, \omega(k+2))$, and a read operation ρ from r with $scnt_r(Q_j, \rho)$. Notice that since $n < |\mathbb{Q}|$ then $k+2 < |\mathbb{Q}|$ and thus there exists a quorum $Q \in \mathbb{Q}$ such that $\left(\bigcap_{i=1}^{k+2} Q_i\right) \cap Q = \emptyset$. Let $Q_j \in \mathbb{Q}$ be such quorum and w.l.o.g let us assume that $Q_j = Q_{k+3}$. We denote the obtained execution by $\Delta(\xi_k)$. Below we can see the last three operations in the execution sequence of $\Delta(\xi_k)$:

a) a complete fast write operation $\omega(k+1)$ by $w_{(k \mod 2)+1}$ with $scnt_{w_{(k \mod 2)+1}}(Q_{k+1}, \omega_{k+1})$,

b) a complete fast write operation $\omega(k+2)$ by $w_{(k+1 \mod 2)+1}$ with $scnt_{w_{(k+1 \mod 2)+1}}(Q_{k+2}, \omega(k+2))$,

c) and a complete read operation ρ by r with $scnt_r(Q_{k+3}, \rho)$.

Notice that by the above construction reader r has to return the value written by $\omega(k+2)$ to preserve property S1. Furthermore since we assumed a $k+2$-wise quorum then ρ, according to Lemma 5.3, observes the value written by $\omega(k+1)$ as the latest from the servers in $\left(\bigcap_{i=1}^{k} Q_i\right) \cap Q_{k+1} \cap Q_{k+3}$ and the value written by $\omega(k+2)$ as

the latest from the servers in $\left(\bigcap_{i=1}^{k} Q_i\right) \cap Q_{k+2} \cap Q_{k+3}$. We should note here that servers in both sets receive messages from all write operations in the set $\{\omega(1), \ldots, \omega(k)\}$. The first set however receives messages from $\omega(k+1)$ but not from $\omega(k+2)$ and vice versa.

Consider now the execution fragment $\Delta(\xi_k)'$ where the two write operations are switched. More precisely we obtain ξ_k' by extending ξ_{k-1} with the write operation $\omega(k+2)$ by $w_{(k+2 \mod 2)+1}$ instead of $\omega(k+1)$. Then we obtain $\Delta(\xi_k)'$ by extending ξ_k' with the write operation $\omega(k+1)$ by $w_{(k+1 \mod 2)+1}$, and the read operation ρ from r. In more detail, the last three operations that appear, and the quorums they contact are as follows:

a) a complete fast write operation $\omega(k+2)$ by $w_{(k+1 \mod 2)+1}$ with $scnt_{w_(k+1 \mod 2)+1}(Q_k+2, \omega_{k+2})$,

b) a complete fast write operation $\omega(k+1)$ by $w_{(k \mod 2)+1}$ with $scnt_{w_{(k \mod 2)+1}}(Q_{k+1}, \omega(k+1))$,

c) and a complete read operation ρ by r with $scnt_r(Q_{k+3}, \rho)$.

Observe that executions $\Delta(\xi_k)$ and $\Delta(\xi_k)'$ differ only at the writers and the servers in $\bigcap_{i=1}^{k+2} Q_i$. Any other server and the reader cannot distinguish between the two executions. In particular the reader does not receive any messages from any server in $\bigcap_{i=1}^{k+2} Q_i$, since $\left(\bigcap_{i=1}^{k+2} Q_i\right) \cap Q_3 = \emptyset$. Moreover the reader observes the value written by ω_1 and ω_2 as latest values from the servers in $\left(\bigcap_{i=1}^{k} Q_i\right) \cap Q_{k+1} \cap Q_{k+3}$ and $\left(\bigcap_{i=1}^{k}\right) \cap Q_{k+2} \cap Q_{k+3}$ respectively. Since those are the same servers that replied with the same values to ρ in $\Delta(\xi_k)$ then r cannot distinguish $\Delta(\xi_k)'$ from $\Delta(\xi_k)$ and thus has to return $\omega(k+2)$ in $\Delta(\xi_k)'$ as well. This however violates property S1 since in $\Delta(\xi_k)'$ the two write operations are consecutive and the latest completed write operation is $\omega(k+1)$. Hence the read operation had to return $\omega(k+1)$ in $\Delta(\xi_k)'$ to preserve property S1, contradicting our findings. □

Remark 5.6 *By close investigation of the predicates of Algorithm* SFW*, one can see that* SFW *approaches the bound of Theorem 5.5, as it produces executions that contain up to $n/2$ fast consecutive write operations, while maintaining atomic consistency. Obtaining a tighter upper bound is subject of future work.*

Note that Theorem 5.5 is not valid in the following two cases: (i) Only a single writer exists in the system, (ii) There is a common intersection among all the quorums in the quorum system. In the first case the sole writer imposes the ordering on the tags introduced in the system and in the second case that ordering is imposed by the common servers that need to be contacted by every operation. It follows by the same theorem that it is impossible to have more than $n-1$ *consecutive* fast write operations then it is also prohibited to have more than $n-1$ *concurrent* fast write operations. Since no communication between the writers is assumed and achieving agreement in an asynchronous distributed system with a single failure (on the set of concurrent writes) is impossible, by [10], then we can obtain the following corollary:

Corollary 5.7 *No MWMR implementation of a safe register, that exploits an n-wise quorum system $\mathbb{Q}$ s.t. $2 \leq n < |\mathbb{Q}|$ and contains only fast writes is possible, if $|\mathcal{W}| > n-1$.*

Moreover assuming that readers also may alter the value of the register, and thus write, then the following theorem holds:

Theorem 5.8 *No MWMR implementation of a safe register, that exploits an n-wise quorum system $\mathbb{Q}$ s.t. $2 \leq n < |\mathbb{Q}|$ and contains only fast operations is possible, if $|\mathcal{W} \cup \mathcal{R}| > n-1$.*

Recall that [7] proved the impossibility of implementations where both writes and reads are fast in the MWMR model, while Theorem 5.8 complements that result by presenting the exact participation conditions under which such implementations could have been possible. They also showed that in the case of a single writer (i.e. $|\mathcal{W}| = 1$), a bound $|\mathcal{R}| < \frac{|\mathcal{S}|}{f} - 2$ is imposed on the number of readers, where f is the total number of allowed server failures. The authors assumed that $f \leq |\mathcal{S}|/2$, and they adopted the technique of communicating with $|\mathcal{S}| - f$ servers for each operation. This technique however depicts a quorum system where every member has a size of $|\mathcal{S}| - f$. The following lemma presents the intersection degree of such a system.

Lemma 5.9 *The intersection degree of a quorum system $\mathbb{Q}$ where $\forall Q_i \in \mathbb{Q}$, $|Q_i| = |\mathcal{S}| - f$ is equal to $\frac{|\mathcal{S}|}{f} - 1$.*

Note that by Lemma 5.9 and Theorem 5.8, the system in [7] could only accommodate:

$$|\mathcal{W} \cup \mathcal{R}| \leq (\frac{|\mathcal{S}|}{f} - 1) - 1 \Rightarrow 1 + |\mathcal{R}| \leq \frac{|\mathcal{S}|}{f} - 2 \Rightarrow |\mathcal{R}| \leq \frac{|\mathcal{S}|}{f} - 3$$

and thus their bound follows. This leads us to the following remark.

Remark 5.10 *Fast implementations, such as the one presented in [7], follow our proved restrictions on the number of participants in the service.*

6 Conclusion

In this paper we presented two algorithms for atomic MWMR registers that allow for fast (single round) read and write operations in the message-passing model with asynchrony and crashes. The first algorithm (CWFR) adopts the traditional two communication round protocol for a write operation, while incorporates the idea of quorum views to enable single communication round reads. The second algorithm (SFW) exploits a new technique, called server side ordering (SSO), that, depending on the intersection degree of a quorum system, allows in some cases for both reads and writes to complete in a single communication round. To the best of our knowledge, algorithm SFW, is the first implementation that achieves single round write operations in the MWMR setting.

Our future work will aim to establish the precise relationship between two algorithms in terms of their performance. At the first glance, algorithm SFW outperforms algorithm CWFR in quorum systems with high intersection degree. The relative performance of the two algorithms becomes unclear when the intersection degree of the quorum system drops below 4. In this case, all write operations in algorithm SFW are slow as well, and thus only the speed (rounds) of read operations will determine a "winner." Simulation results derived from the implementations of the algorithms presented here may facilitate the comparison. Such simulations may also yield a sensible comparison of the performance of these algorithms in the MWMR setting with other algorithms in the SWMR setting.

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. Preliminary version appeared in PODC 2004.

[2] Marcos Aguilera, Idit Keidar, Dahlia Malkhi, and Alexander Shraer. Dynamic atomic storage without consensus. In *Proceedings of the twenty-eight annual ACM symposium on Principles of distributed computing (PODC09)*, to appear, 2009.

[3] H. Attiya, A. Bar-Noy, and D. Dolev. Sharing memory robustly in message passing systems. *Journal of the ACM*, 42(1):124–142, 1996.

[4] Gregory Chockler, Seth Gilbert, Vincent Gramoli, Peter M. Musial, and Alex A. Shvartsman. Reconfigurable distributed storage for dynamic networks. *J. Parallel Distrib. Comput.*, 69(1):100–116, 2009.

[5] Gregory Chockler, Idit Keidar, Rachid Guerraoui, and Marko Vukolic. Reliable distributed storage. *IEEE Computer*, 2008.

[6] S. Dolev, S. Gilbert, N. Lynch, A. Shvartsman, and J. Welch. GeoQuorums: Implementing atomic memory in mobile ad hoc networks. *Distributed Computing*, 18(2):125–155, 2005.

[7] Partha Dutta, Rachid Guerraoui, Ron R. Levy, and Arindam Chakraborty. How fast can a distributed atomic read be? In *Proceedings of the 23rd ACM symposium on Principles of Distributed Computing (PODC)*, pages 236–245, 2004.

[8] Burkhard Englert and Alexander A. Shvartsman. Graceful quorum reconfiguration in a robust emulation of shared memory. In *Proceedings of International Conference on Distributed Computing Systems (ICDCS)*, pages 454–463, 2000.

[9] Rui Fan and Nancy Lynch. Efficient replication of large data objects. In *Proceeding of the 17th International Symposium on Distributed Computing (DISC)*, pages 75–91, 2003.

[10] Michael J. Fischer, Nancy A. Lynch, and Michael S. Paterson. Impossibility of distributed consensus with one faulty process. *J. ACM*, 32(2):374–382, 1985.

[11] Hector Garcia-Molina and Daniel Barbara. How to assign votes in a distributed system. *Journal of the ACM*, 32(4):841–860, 1985.

[12] Chryssis Georgiou, Peter M. Musial, and Alex A. Shvartsman. Long-lived RAMBO: Trading knowledge for communication. *Theoretical Computer Science*, 383(1):59–85, 2007.

[13] Chryssis Georgiou, Peter M. Musial, and Alexander A. Shvartsman. Developing a consistent domain-oriented distributed object service. *IEEE Transactions of Parallel and Distributed Systems, to appear*, 2009. Preliminary version appeared in NCA 2005.

[14] Chryssis Georgiou, Nicolas Nicolaou, and Alexander A. Shvartsman. On the robustness of (semi)fast quorum-based implementations of atomic shared memory. In *Proceedings of the 22nd International Symposium on Distributed Computing (DISC)*, pages 289–304, 2008.

[15] Chryssis Georgiou, Nicolas Nicolaou, and Alexander A. Shvartsman. Fault-tolerant semifast implementations for atomic read/write registers. *Journal of Parallel and Distributed Computing*, 69(1):62–79, 2009. Preliminary version appeared in SPAA 2006.

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

[17] S. Gilbert, N. Lynch, and A.A. Shvartsman. RAMBO II: Rapidly reconfigurable atomic memory for dynamic networks. In *Proceedings of International Conference on Dependable Systems and Networks (DSN)*, pages 259–268, 2003.

[18] Vincent Gramoli, Emmanuelle Anceaume, and Antonino Virgillito. SQUARE: scalable quorum-based atomic memory with local reconfiguration. In *Proceedings of the 2007 ACM symposium on Applied computing (SAC)*, pages 574–579, 2007.

[19] Rachid Guerraoui and Marko Vukolić. How fast can a very robust read be? In *Proceedings of the 25th ACM symposium on Principles of Distributed Computing (PODC)*, pages 248–257, 2006.

[20] Rachid Guerraoui and Marko Vukolić. Refined quorum systems. In *Proceedings of the 26th ACM Symposium on Principles of Distributed Computing (PODC)*, pages 119–128, 2007.

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

[22] Leslie Lamport. On interprocess communication, parts I and II. *Distributed Computing*, 1(2):77–101, 1986.

[23] N. Lynch and A.A. Shvartsman. RAMBO: A reconfigurable atomic memory service for dynamic networks. In *Proceedings of 16th International Symposium on Distributed Computing (DISC)*, pages 173–190, 2002.

[24] N.A. Lynch. *Distributed Algorithms*. Morgan Kaufmann Publishers, 1996.

[25] Nancy Lynch and Mark Tuttle. An introduction to input/output automata. *CWI-Quarterly*, pages 219–246, 1989.

[26] Nancy A. Lynch and Alexander A. Shvartsman. Robust emulation of shared memory using dynamic quorum-acknowledged broadcasts. In *Proceedings of Symposium on Fault-Tolerant Computing*, pages 272–281, 1997.

[27] Dahlia Malkhi and Michael Reiter. Byzantine quorum systems. *Distributed Computing*, 11:203–213, 1998.

[28] D. Peleg and A. Wool. Crumbling walls: A class of high availability quorum systems. In *Proceedings of 14th ACM Symposium on Principles of Distributed Computing (PODC)*, pages 120–129, 1995.

[29] Robert H. Thomas. A majority consensus approach to concurrency control for multiple copy databases. *ACM Trans. Database Syst.*, 4(2):180–209, 1979.

A Algorithm CWFR

A.1 Formal Specification of CWFR Algorithm

Algorithm CWFR consist of four kinds of automata: writer $Writer_p$, reader $Reader_p$, server $Server_s$, and channels $Channel_{p,s}$ and $Channel_{s,p}$, for $p \in \mathcal{W} \cup \mathcal{R}$ and $s \in \mathcal{S}$. Figures 1-3 present the IOA specification of the server, writer and reader automata of algorithm CWFR, respectively.

Signature:

Input:
$\mathsf{rcv}(m)_{p,s_i},\ m \in M,\ s_i \in \mathcal{S},\ p \in \mathcal{R} \cup \mathcal{W}$
fail_{s_i}

Output:
$\mathsf{send}(m)_{s_i,p},\ m \in M,\ s_i \in \mathcal{S},\ p \in \mathcal{R} \cup \mathcal{W}$

State:

$tag = \langle ts, label \rangle \in T \times \mathcal{W}$, initially $\{0, \min(\mathcal{W})\}$
$value \in V$, initially $\perp$
$Counter(p) \in \mathbb{N}^+,\ p \in \mathcal{R} \cup \mathcal{W}$, initially 0

$msgType \in \{$WRITEACK,READACK,INFOACK$\}$
$status \in \{idle, active\}$, initially $idle$
$failed$, a Boolean initially **false**

Transitions:

Input $\mathsf{rcv}(\langle msgT, t, val, C \rangle)_{p,s_i}$
Effect:
  if $\neg failed$ then
    if $status = idle$ and $C > Counter(p)$ then
      $status \leftarrow active$
      $Counter(p) \leftarrow C$
      if $tag < t$ then
        $(tag.ts, tag.label, value) \leftarrow (t.ts, t.label, val)$

Output $\mathsf{send}(\langle msgT, t, val, C \rangle)_{s_i,p}$
Precondition:
  $\neg failed$
  $status = active$
  $p \in \mathcal{R} \cup \{w\}$
  $\langle msgT, t, val, C \rangle =$
    $\langle msgType, tag, value, Counter(p) \rangle$
Effect:
  $status \leftarrow idle$

Input fail_{s_i}
Effect:
  $failed \leftarrow true$

Figure 1: Algorithm CWFR, $Server_{s_i}$: Signature, State and Transitions

A.2 Correctness of Algorithm CWFR

We proceed to show the correctness (safety) of algorithm CWFR, that is, to show that the algorithm satisfies the atomicity properties presented in Section 2. Let write-fix(π) (resp. read-fix(π)) denote the fix point of a write (resp. read) operation π, where the status becomes equal to done. In other words the write-fix(π) operation happens when write-phase2-fix event occurs. On the other hand read-fix(π) of a fast read operation happens when read-qview-eval event occurs, while if the read operation is slow then its fix point is reached when read-phase2-fix occurs. For the rest of the section we use $\tau_p(\pi)$ to denote the tag of a read/write operation π from a process p, after the read-fix(π) or write-fix(π) event of π respectively. For the server automaton $\tau_s(\pi)$ denotes the value of the tag variable at server s after the send action at s for π. Let for any process p, τ_p denote the value of the local tag variable at p. For a read/write operation π we denote by start-maxTag(π) the maximum tag at the read/write event of π and by witnessed-maxTag(π) (analogously witnessed-minTag(π)) the maximum (resp. minimum) tag witnessed at the read-phase1-fix (resp. write-phase1-fix) event if π is a read (resp. write) operation. Lastly given tag τ and a set of servers Q that replied to some operation π, let $M_{Q,\tau} = \{s : s \in Q \wedge \tau_s(\pi) > \tau\}$ be the set of servers in Q that replied with a tag greater than τ.

We first provide an alternative definition to atomicity, to express the three atomicity properties based on the tags returned. Notice that for ease of analysis we split property P1 in two properties that capture the relation between reads and writes separately. So the following must hold for every finite or infinite execution ξ of our implementation:

1. For each process p the tag variable is alphanumerically monotonically nondecreasing and it contains a non-negative timestamp.
2. If the read event of a read operation ρ from reader r succeeds the write-fix(ω) event of a write operation ω in ξ then, $\tau_r(\rho) \geq \tau_w(\omega)$.

Signature:

Input:
$\mathsf{write}(v)_{w_i},\ v \in V, w_i \in \mathcal{W}$
$\mathsf{rcv}(m)_{s_j,w_i},\ m \in M,\ s_j \in \mathcal{S}, w_i \in \mathcal{W}$
$\mathsf{fail}_{w_i}, w_i \in \mathcal{W}$

Output:
$\mathsf{send}(m)_{w_i,s_j},\ m \in M,\ s_j \in \mathcal{S}, w_i \in \mathcal{W}$
$\mathsf{write\text{-}ack}_{w_i}, w_i \in \mathcal{W}$

Internal:
$\mathsf{write\text{-}phase1\text{-}fix}_{w_i}, w_i \in \mathcal{W}$
$\mathsf{write\text{-}phase2\text{-}fix}_{w_i}, w_i \in \mathcal{W}$

State:

$tag = \langle ts, wid \rangle \in T \times \mathcal{W}$, initially $\{0, w_i\}$
$value \in V$, initially $\perp$
$phase \in \{1, 2\}$, initially 1
$pvalue \in V$, initially $\perp$
$wCounter \in \mathbb{N}^+$, initially 0

$status \in \{idle, active, done\}$, initially $idle$
$srvAck \subseteq M \times \mathcal{S}$, initially $\emptyset$
$maxAck \subseteq M \times \mathcal{S}$, initially $\emptyset$
$failed$, a Boolean initially **false**

Transitions:

Input $\mathsf{write}(v)_{w_i}$
Effect:
  if $\neg failed$ then
    if $status = idle$ then
      $status \leftarrow active$
      $srvAck \leftarrow \emptyset$
      $phase \leftarrow 1$
      $pvalue \leftarrow value$
      $value \leftarrow v$
      $wCounter \leftarrow wCounter + 1$

Input $\mathsf{rcv}(\langle msgT, t, val, C \rangle)_{s_j,w_i}$
Effect:
  if $\neg failed$ then
    if $status = active$ and $wCounter = C$ then
      $srvAck \leftarrow srvAck \cup \{s_j, \langle msgT, t, val, C \rangle\}$

Output $\mathsf{send}(\langle msgT, t, val, C \rangle)_{w_i,s_j}$
Precondition:
  $status = active$
  $\neg failed$
  $\big[(phase = 1 \wedge \langle msgT, t, val, C \rangle =$
    $\langle \text{READ}, tag, pvalue, wCounter \rangle) \vee$
  $(phase = 2 \wedge \langle msgT, t, val, C \rangle =$
    $\langle \text{WRITE}, tag, value, wCounter \rangle)\big]$
Effect:
  none

Output $\mathsf{write\text{-}ack}_w$
Precondition:
  $status = done$
  $\neg failed$
Effect:
  $status \leftarrow idle$

Internal $\mathsf{write\text{-}phase1\text{-}fix}_{w_i}$
Precondition:
  $\neg failed$
  $status = active$
  $phase = 1$
  $\exists Q \in \mathbb{Q} : Q \subseteq \{s : (s, m) \in srvAck\}$
Effect:
  $maxTs \leftarrow max_{s \in Q \wedge (s,m) \in srvAck}(m.t)$
  $tag = \langle maxTs + 1, w_i \rangle$
  $phase \leftarrow 2$
  $srvAck \leftarrow \emptyset$
  $wCounter \leftarrow wCounter + 1$

Internal $\mathsf{write\text{-}phase2\text{-}fix}_{w_i}$
Precondition:
  $\neg failed$
  $status = active$
  $phase = 2$
  $\exists Q \in \mathbb{Q} : Q \subseteq \{s : (s, m) \in srvAck\}$
Effect:
  $status \leftarrow done$

Input fail_w
Effect:
  $failed \leftarrow true$

Figure 2: Algorithm CWFR, $Writer_{w_i}$: Signature, State and Transitions

Signature:

Input:
$\mathsf{read}_{r_i},\ r_i \in \mathcal{R}$
$\mathsf{rcv}(m)_{s_j,r_i},\ m \in M,\ r_i \in \mathcal{R},\ s_j \in \mathcal{S}$
$\mathsf{fail}_{r_i},\ r_i \in \mathcal{R}$

Output:
$\mathsf{send}(m)_{r_i,s_j},\ m \in M,\ r_i \in \mathcal{R},\ s_j \in \mathcal{S}$
$\mathsf{read\text{-}ack}(v)_{r_i},\ v \in V,\ r_i \in \mathcal{R}$

Internal:
$\mathsf{read\text{-}phase1\text{-}fix}_{r_i}$
$\mathsf{read\text{-}phase2\text{-}fix}_{r_i}$

State:

$tag = \langle ts, wid\rangle \in T \times \mathcal{W}$, initially $\{0, \min(\mathcal{W}), 0\}$
$maxTag = \langle ts, wid\rangle \in T \times \mathcal{W}$, initially $\{0, \min(\mathcal{W}), 0\}$
$value \in V$, initially $\perp$
$phase \in \{1, 2\}$, initially 1
$retvalue \in V$, initially $\perp$
$rCounter \in \mathbb{N}^+$, initially 0
$status \in \{idle, active, done\}$, initially $idle$
$srvAck \subseteq M \times \mathcal{S}$, initially $\emptyset$
$maxAck \subseteq M \times \mathcal{S}$, initially $\emptyset$
$maxTagSrv \subseteq \mathcal{S}$, initially $\emptyset$
$replyQ \subseteq \mathcal{S}$, initially $\emptyset$
$failed$, a Boolean initially **false**

Transitions:

Input read_{r_i}
Effect:
  if $\neg failed$ then
   if $status = idle$ then
    $status \leftarrow active$
    $rCounter \leftarrow rCounter + 1$

Input $\mathsf{rcv}(\langle msgT, t, val, C\rangle)_{s_j,r_i}$
Effect:
  if $\neg failed$ then
   if $status = active$ and $rCounter = C$ then
    $srvAck \leftarrow srvAck \cup \{(s_j, \langle msgT, t, val, C\rangle)\}$

Output $\mathsf{send}(\langle msgT, t, val, C\rangle)_{r_i,s_j}$
Precondition:
  $status = active$
  $\neg failed$
  $[(phase = 1 \wedge \langle msgT, t, val, C\rangle =$
    $\langle \text{READ}, maxTag, value, rCounter\rangle) \vee$
   $(phase = 2 \wedge \langle msgT, t, val, C\rangle =$
    $\langle \text{INFORM}, maxTag, value, rCounter\rangle)]$
Effect:
  none

Output $\mathsf{read\text{-}ack}(v)_{r_i}$
Precondition:
  $\neg failed$
  $status = done$
  $v = retvalue$
Effect:
  $replyQ \leftarrow \emptyset$
  $srvAck \leftarrow \emptyset$
  $status \leftarrow idle$

Internal $\mathsf{read\text{-}phase2\text{-}fix}_{r_i}$
Precondition:
  $\neg failed$
  $status = active$
  $phase = 2$
  $\exists Q \in \mathbb{Q} : Q \subseteq \{s : (s, m) \in srvAck\}$
Effect:
  $status \leftarrow done$
  $phase \leftarrow 1$

Internal $\mathsf{read\text{-}phase1\text{-}fix}_{r_i}$
Precondition:
  $\neg failed$
  $status = active$
  $phase = 1$
  $\exists Q \in \mathbb{Q} : Q \subseteq \{s : (s, m) \in srvAck\}$
Effect:
  $replyQ \leftarrow Q$
  $maxTag \leftarrow max_{s \in replyQ \wedge (s,m) \in srvAck}(m.t)$
  $maxAck \leftarrow \{(s, m) : (s, m) \in srvAck \text{ and } m.t = maxTag\}$
  $maxTagSrv \leftarrow \{s : s \in replyQ \wedge (s, m) \in maxAck\}$
  $value \leftarrow \{m.val : (s, m) \in maxAck\}$

Internal $\mathsf{read\text{-}qview\text{-}eval}_{r_i}$
Precondition:
  $\neg failed$
  $replyQ \neq \emptyset$
Effect:
  $tag \leftarrow max_{s \in replyQ \wedge (s,m) \in srvAck}(m.t)$
  $maxAck \leftarrow \{(s, m) : (s, m) \in srvAck \text{ and } m.t = maxTag\}$
  $maxTagSrv \leftarrow \{s : s \in replyQ \wedge (s, m) \in maxAck\}$
  $retvalue \leftarrow \{m.val : (s, m) \in maxAck\}$
  if $replyQ = maxTagSrv$ then
   $status \leftarrow done$
  else
   if $\exists Q_j \in \mathbb{Q}, Q_j \neq replyQ$ s.t. $replyQ \cap Q_j \subseteq maxTsSrv$ then
    $tag \leftarrow maxTag$
    $retvalue \leftarrow value$
    $phase \leftarrow 2$
    $srvAck \leftarrow \emptyset$
    $rCounter \leftarrow rCounter + 1$
   else
    $replyQ \leftarrow replyQ - \{s : s \in maxTagSrv\}$

Input fail_{r_i}
Effect:
  $failed \leftarrow true$

Figure 3: Algorithm CWFR, $Reader_{r_i}$: Signature, State and Transitions

3. If ω_1 and ω_2 are two write operations from the writers w and w' respectively, such that $\omega_1 \rightarrow \omega_2$ in ξ, then $\tau_w(\omega_2) > \tau_{w'}(\omega_1)$.
4. If ρ_1 and ρ_2 are two read operations from the readers r and r' respectively, such that $\rho_1 \rightarrow \rho_2$ in ξ, then $\tau_r(\rho_2) \geq \tau_{r'}(\rho_1)$.

First we need to ensure that any process in the system maintains only monotonically nondecreasing tags. Hence once some process p sets its tag variable to a value k at time t of an execution ξ, then it cannot be the case that p sets its tag to a value $\ell \leq k$ at a time t' such that $t' > t$ in ξ.

Lemma A.1 *For each server process $s \in \mathcal{S}$, τ_s is alphanumerically monotonically nondecreasing and contains a $\tau_s.ts > 0$.*

Proof. It is easy to see that a server s_i modifies its tag variable only if the tag τ in the received messages is such that $\tau > tag$. This means that either: a) $\tau.ts > tag.ts$ or b) $\tau.ts = tag.ts$ and $\tau.wid > tag.wid$. So the server's tag is monotonically incrementing. Furthermore since the initial tag of the server is set to $\langle 0, \min wid \rangle$ and the tag is updated only if $\tau.ts \geq tag.ts$, then $tag.ts$ is always greater than 0. □

Lemma A.2 *If a server $s \in \mathcal{S}$ receives a message from a process p, for operation π, that contains a tag τ, then s replies to p with a tag $\tau_s(\pi) \geq \tau$.*

Proof. When the server receives the message from processor p it first compares τ with its local tag τ_s. If $\tau > \tau_s$ then the server sets $\tau_s = \tau$. From this it follows that τ_s is at least equal to the tag τ of the message. Since by Lemma A.1 the tag of the server is monotonically nondecreasing, then when the $\mathsf{send}_{s,p}$ event occurs, the server replies to p with a tag $\tau_s(\pi) \geq \tau_s \geq \tau$. Hence the lemma follows. □

Lemma A.3 *For each writer process $w \in \mathcal{W}$, τ_w is monotonically nondecreasing and contains a non-negative timestamp.*

Proof.
Each writer process w modifies its local tag during its first communication round. In particular when the write-phase1-fix event happens for a write operation ω, then τ_w becomes equal to $\tau_w(\omega) = \langle \mathsf{witnessed\text{-}maxTag}(\omega).ts + 1, w_i \rangle$. So it suffice to show that $\mathsf{start\text{-}maxTag}(\omega) \leq \mathsf{witnessed\text{-}maxTag}(\omega)$. Suppose that all the servers of a quorum $Q_j \in \mathbb{Q}$, received messages and replied to w, for ω. Every message sent from w to any server $s \in Q_j$ (when $\mathsf{send}_{w,s}$ occurs), contains a tag $\tau = \mathsf{start\text{-}maxTag}(\omega)$. By Lemma A.2, any $s \in Q_j$ replies with a tag $\tau_s(\omega) \geq \tau \geq \mathsf{start\text{-}maxTag}(\omega)$. Thus $\forall s \in Q_j$, $\tau_s(\omega) \geq \mathsf{start\text{-}maxTag}(\omega)$ and it follows that $\tau_s(\omega).ts \geq \mathsf{start\text{-}maxTag}(\omega).ts$. Since $\mathsf{witnessed\text{-}maxTag}(\omega).ts = \max(\tau_s(\omega).ts)$ then $\mathsf{witnessed\text{-}maxTag}(\omega).ts \geq \mathsf{start\text{-}maxTag}(\omega).ts$ and hence $\tau_w(\omega) = \langle \mathsf{witnessed\text{-}maxTag}(\omega) + 1, w \rangle > \mathsf{start\text{-}maxTag}(\omega)$. Therefore not only the tag of a writer is nondecreasing but we show explicitly that the writer's tag is monotonically increasing. Furthermore since the writer adopts the maximum tag sent from the servers, and since by Lemma A.1 the servers tags contain non-negative timestamps, then it follows that the writer contains non-negative timestamps as well. □

Lemma A.4 *For each reader process $r \in \mathcal{R}$, τ_r is monotonically nondecreasing and contains a non-negative timestamp.*

Proof. Notice that the tag variable of a reader is $\tau_r \leq \mathsf{start\text{-}maxTag}(\rho)$ when the read event occurs and becomes $\tau_r = \tau_r(\rho)$ at the end of the operation. So it suffices to show that $\tau_r(\rho) \geq \mathsf{start\text{-}maxTag}(\rho)$. With similar arguments to Lemma A.3 it can be shown that for every $s \in Q_j$ that replies to an operation ρ invoked by r, $\tau_s(\rho) \geq \mathsf{start\text{-}maxTag}(\rho)$. Since $\mathsf{witnessed\text{-}maxTag}(\rho) = \max(\tau_s(\rho))$ and $\mathsf{witnessed\text{-}minTag}(\rho) = \min(\tau_s(\rho))$ then it follows that both $\mathsf{witnessed\text{-}maxTag}(\rho), \mathsf{witnessed\text{-}minTag}(\rho) \geq \mathsf{start\text{-}maxTag}(\rho)$. By the algorithm the tag returned by the read operation is $\mathsf{witnessed\text{-}minTag}(\rho) \leq \tau_r(\rho) \leq \mathsf{witnessed\text{-}maxTag}(\rho)$. Hence $\tau_r(\rho) \geq \mathsf{start\text{-}maxTag}(\rho)$. Thus no matter which of the tags is chosen to be returned at the end of the read operation nondecreasing monotonicity is preserved. Also since by Lemma A.1 all the servers reply with a non negative timestamp, then it follows that r contains non-negative timestamps as well. □

Lemma A.5 *For each process $p \in \mathcal{R} \cup \mathcal{W} \cup \mathcal{S}$ the tag variable is monotonically nondecreasing and contains a non-negative timestamp.*

Proof. Follows from Lemmas A.1, A.3 and A.4 □

The following lemma states that if a read operation ρ returns a tag $\tau < maxTag$ it must be the case that any pairwise intersection of the replied quorum contains at least a single server s such that $\tau_s(\rho) = \tau$.

Lemma A.6 *In any execution ξ, if a read operation ρ from r receives replies from the members of quorum Q_i and returns a tag $\tau_r(\rho) <$ witnessed-maxTag(ρ), then $\forall Q_j \in \mathbb{Q}, j \neq i$ it must be true that $(Q_i \cap Q_j) - M_{Q_i,\tau_r(\rho)} \neq \emptyset$.*

Proof. By definition the intersection of two quorums $Q_i, Q_j \in \mathbb{Q}$ is not empty. Let us assume to derive contradiction that a read operation ρ may return a tag $\tau_r(\rho) = \tau <$ witnessed-maxTag(ρ) and may exist $(Q_i \cap Q_j) - M_{Q_i,\tau_r(\rho)} = \emptyset$. According to our algorithm, when read-qview-eval event occurs, we first check if either $qView(1)$ or $qView(3)$ is observed in Q_i. If neither of those quorum views is observed then we remove all the servers with the current maximum tag from Q_i and we repeat the check on the remaining servers. It follows that since all the servers $s \in Q_i \cap Q_j$ were removed from Q_i then it must be the case that $\tau_s(\rho) > \tau$. So there must be a tag $\tau' > \tau$ s.t. $A = (Q_i \cap Q_j) - M_{Q_i,\tau'} \neq \emptyset$ and all servers $s' \in A$ contain $\tau_{s'}(\rho) = \tau'$. If this happens there are two cases for the reader: (a) $\forall s' \in (Q_i) - M_{Q_i,\tau'}, \tau_{s'}(\rho) = \tau'$ and thus $qView(1)$ is observed and the reader returns $\tau_r(\rho) = \tau'$, or (b) $\forall s' \in A, \tau_{s'}(\rho) = \tau'$ and thus $qView(3)$ is observed and the reader returns $\tau_r(\rho) =$ witnessed-maxTag(ρ). Since witnessed-maxTag$(\rho) \geq \tau'$ then in any case the read operation ρ would return a tag $\tau_r(\rho) > \tau$ and that contradicts our assumption. □

Derived from the above lemma, the next lemma states that a read operation basically returns either the $maxTag$ or the maximum of the minimum tags of all the pairwise intersections of the replied quorum.

Lemma A.7 *If a read operation ρ from r receives replies from a quorum Q_i, then $\forall Q_j \in \mathbb{Q}, j \neq i$, $\tau_r(\rho) \geq \min(\tau_s(\rho))$ for $s \in Q_i \cap Q_j$.*

Proof. This lemma follows directly from Lemma A.6. Let a subset of servers in $Q_i \cap Q_j$ replied to ρ with the minimum tag among all the servers of that intersection, say τ. If the iteration of the read-eval-qview event of ρ reaches tag τ then either ρ observes $qView(1)$ and returns $\tau_r(\rho) = \tau$ or it observes $qView(3)$ and returns $\tau_r(\rho) =$ witnessed-maxTag$(\rho) \geq \tau$. This is true for all the intersections $Q_i \cap Q_j$, for $i \neq j$. And the lemma follows. □

Lemma A.8 *If the* read *event of a read operation ρ from reader r succeeds the* write-fix(ω) *event of a write operation ω from w in an execution ξ then, $\tau_r(\rho) \geq \tau_w(\omega)$.*

Proof. Assume w.l.o.g. that the write operation receives messages from two, not necessarily different, quorums Q_i and Q_j during its first and second communication rounds respectively. Furthermore let us assume that the read operation receives replies from a quorum Q_z, not necessarily different from Q_i or Q_j, during its first communication round. According to the algorithm the write operation ω detects the maximum tag from Q_i, increments that and propagates the new tag to Q_j. Since $\forall s \in Q_i$, witnessed-maxTag$(\omega) \geq \tau_s(\omega)$ then from the intersection property of a quorum system it follows that $\forall s \in (Q_i \cap Q_j) \cup (Q_i \cap Q_z), \tau_w(\omega) >$ witnessed-maxTag$(\omega) \geq \tau_s(\omega)$. From the fact that w propagates $\tau_w(\omega)$ in ω's second communication round and from Lemma A.2 it follows that at the write-fix(ω) every $s \in (Q_j \cap Q_z)$ contains a tag $\tau_s(\omega) \geq \tau_w(\omega)$.

Since the read operation succeeds the write-fix(ω), then from Lemma A.2 the read operation will obtain a tag $\tau_s(\rho) \geq \tau_s(\omega) \geq \tau_w(\omega)$, from every server $s \in Q_j \cap Q_z$ and so $\min(\tau_s(\rho)) \geq \tau_w(\omega)$. Thus from Lemma A.7 $\tau_r(\rho) \geq \tau_s(\rho)$ for $s \in Q_j \cap Q_z$ and hence $\tau_r(\rho) \geq \tau_w(\omega)$ completing the proof. □

Lemma A.9 *If ω_1 and ω_2 are two write operations from the writers w and w' respectively, such that $\omega_1 \rightarrow \omega_2$ in ξ, then $\tau_{w'}(\omega_2) > \tau_w(\omega_1)$*

Proof. From the precedence relation of the two write operations it follows that the write-fix(ω_1) occurs before the write event of ω_2. Recall that for a write operation ω, $\tau_w(\omega) = \langle$witnessed-maxTag$(\omega).ts + 1, w\rangle$. So it suffices to show here that witnessed-maxTag$(\omega_2) >$ witnessed-maxTag(ω_1). This however is straightforward from Lemma A.2 and the value propagated during the second communication round of ω_1. In particular let ω_1 propagate $\tau_w(\omega_1) >$ witnessed-maxTag(ω_1) to a quorum Q_i. Notice that every $s \in Q_i$ replies with $\tau_s(\omega_1) \geq \tau_w(\omega_1)$ to the second communication round of ω_1. Furthermore let the write operation ω_2 receive replies from a quorum Q_j, not necessarily different than Q_i, during its first communication round. Since write-fix(ω_1) occurs before the write event of ω_2 then, by Lemmas A.1 and A.2, $\forall s_g \in Q_i \cap Q_j$ $\tau_{s_g}(\omega_2) \geq \tau_{s_g}(\omega_1) \geq \tau_w(\omega_1)$. Thus witnessed-maxTag$(\omega_2) \geq \tau_{s_g}(\omega_2) \geq \tau_w(\omega_1)$ and hence, since $\tau_{w'}(\omega_2) = \langle$witnessed-maxTag$(\omega_2).ts + 1, w'\rangle >$ witnessed-maxTag(ω_2), then $\tau_{w'}(\omega_2) > \tau_w(\omega_1)$. □

Lemma A.10 *If ρ_1 and ρ_2 are two read operations from the readers r and r' respectively, such that $\rho_1 \rightarrow \rho_2$ in ξ, then $\tau_r(\rho_2) \geq \tau_{r'}(\rho_1)$.*

Proof. Since $\rho_1 \rightarrow \rho_2$ in ξ, then the read-ack event of ρ_1 occurs before the read event of ρ_2. Let consider that both read operations are invoked from the same reader $r = r'$. It follows from Lemma A.4 that $\tau_r(\rho_1) \leq \tau_r(\rho_2)$ because the tag variable is monotonically non-decrementing. So it remains to investigate what happens when the two read operations are invoked by two different processes, r and r' respectively. Suppose that every server s in a quorum Q_i receives the messages of operation ρ_1 with an event $\mathsf{rcv}(m)_{r,s}$, and replies with a tag $\tau_s(\rho_1)$ with an event $\mathsf{send}(m)_{s,r}$ to r. Notice that for every server that reply, as mentioned in Lemma A.2, $\tau_s(\rho_1) \geq$ start-maxTag(ρ_1). Let the members of the quorum Q_j (not necessarily different than Q_i) receive messages and reply to ρ_2. Again for every $s' \in Q_j$, $\tau_{s'}(\rho_2) \geq$ start-maxTag(ρ_2). We know that the tag of the read operation ρ_1 after the read-qview-eval event of ρ_1 may take a value between witnessed-maxTag$(\rho_1) \geq \tau_r(\rho_1) \geq$ witnessed-minTag(ρ_1). It suffice to examine the two extreme cases and every intermediate value can be proved similarly: (1) $\tau_r(\rho_1) =$ witnessed-minTag(ρ_1), and (2) $\tau_r(\rho_1) =$ witnessed-maxTag(ρ_1).

Case 1: Consider the case where $\tau_r(\rho_1) =$ witnessed-minTag(ρ_1), including the case where witnessed-minTag$(\rho_1) =$ witnessed-maxTag(ρ_1). This may happen only if the read-qview-eval event reaches an iteration with tag $\tau =$ witnessed-minTag(ρ_1) and observes $qView(1)$. In other words all the servers $s \in Q_i - M_{Q_i,\tau_s(\rho_1)}$ replied with $\tau_s(\rho_1) =$ witnessed-minTag(ρ_1). By Lemma A.6 it follows that $(Q_i \cap Q_j) - M_{Q_i \cap Q_j,\tau_s(\rho_1)} \neq \emptyset$ and thus every server $s' \in Q_i \cap Q_j$ replied to ρ_1 with a tag $\tau_{s'}(\rho_1) \geq$ witnessed-minTag(ρ_1). By Lemma A.1 it follows that every server $s' \in Q_i \cap Q_j$, replies with a tag $\tau_{s'}(\rho_2) \geq \tau_{s'}(\rho_1) \geq$ witnessed-minTag(ρ_1). The read operation ρ_2 may return a value within the interval witnessed-minTag$(\rho_2) \leq \tau_{r'}(\rho_2) \leq$ witnessed-maxTag(ρ_2). Since for every server $s' \in Q_i \cap Q_j$, $\tau_{s'}(\rho_2) \geq$ witnessed-minTag$(\rho_1) = \tau_r(\rho_1)$ then witnessed-maxTag$(\rho_2) \geq \tau_{s'}(\rho_2) \geq \tau_r(\rho_1)$. Hence if $\tau_{r'}(\rho_2) =$ witnessed-maxTag(ρ_2) it follows that $\tau_{r'}(\rho_2) \geq \tau_r(\rho_1)$. On the other hand if $\tau_{r'}(\rho_2) =$ witnessed-minTag(ρ_2) we need to consider two cases: (a) witnessed-minTag$(\rho_2) \geq$ witnessed-minTag(ρ_1) and (b) witnessed-minTag$(\rho_2) <$ witnessed-minTag(ρ_1). If the first case is valid then it follows immediately that $\tau_{r'}(\rho_2) \geq$ witnessed-minTag(ρ_q) and thus $\tau_{r'}(\rho_2) \geq \tau_r(\rho_1)$. If case (b) is valid then it follows that the iteration reached a tag equal to witnessed-minTag(ρ_2). Since however every server $s' \in Q_i \cap Q_j$, replied with $\tau_{s'}(\rho_2) \geq$ witnessed-minTag(ρ_1), then $\tau_{s'}(\rho_2) \geq$ witnessed-minTag(ρ_2) as well and thus all these servers should be removed by the iteration where tag is equal to witnessed-minTag(ρ_2). But this means that $(Q_i \cap Q_j) - M_{Q_j,\text{witnessed-minTag}(\rho_2)} = \emptyset$ and that contradicts Lemma A.6. So such a case is impossible.

Case 2: Here we examine the case where $\tau_r(\rho_1) =$ witnessed-maxTag(ρ_1). This may happen after the read-qview-eval of ρ_1 if either observes a quorum view $qView(1)$ or a quorum view $qView(3)$. Let us examine the two cases separately.

Case 2a: In this case ρ_1 witnessed a $qView(1)$. Therefore it must be the case that $\forall s \in Q_i, s$ replied with $\tau_s(\rho_1) =$ witnessed-maxTag$(\rho_1) =$ witnessed-minTag$(\rho_1) = \tau_r(\rho_1)$. Thus by Lemma A.1 $\forall s \in Q_i \cap Q_j$, s replies with a tag $\tau_s(\rho_2) \geq \tau_s(\rho_1)$ to ρ_2, and hence, ρ_2 witnesses a maximum tag

$$\text{witnessed-maxTag}(\rho_2) \geq \text{witnessed-maxTag}(\rho_1) \Rightarrow \text{witnessed-maxTag}(\rho_2) \geq \tau_r(\rho_1) \tag{1}$$

Recall that witnessed-minTag$(\rho_2) \leq \tau_{r'}(\rho_2) \leq$ witnessed-maxTag(ρ_2). Clearly if $\tau_{r'}(\rho_2) =$ witnessed-maxTag(ρ_2) then $\tau_{r'}(\rho_2) \geq \tau_r(\rho_1)$. So it remains to examine the case where $\tau_{r'}(\rho_2) <$ witnessed-maxTag(ρ_2). By Lemma A.7, $\tau_{r'}(\rho_2)$ must be greater or equal to the minimum tag of any intersection of Q_j. Since $\min(\tau_s(\rho_2)) \geq \tau_r(\rho_1)$, for every $s \in Q_i \cap Q_j$, then by that lemma $\tau_{r'}(\rho_2) \geq \tau_r(\rho_1)$.

Case 2b: This is the case where $\tau_r(\rho_1) =$ witnessed-maxTag(ρ_1), because r witnessed a quorum view $qView(3)$. In this case ρ_1 proceeds in phase 2 before completing. Since $\rho_1 \rightarrow \rho_2$ and since ρ_2 happens after the read-ack [1] action of ρ_1, it means that ρ_2 happens after the read-phase2-fix action of ρ_1 as well. However ρ_1 proceeds to phase 2 only after the read-phase1-fix and read-qview-eval actions. In the latter action ρ_1 fixes the $maxTag$ variable to be equal to the witnessed-maxTag(ρ_1). Once in phase 2, ρ_1 sends inform messages with its $maxTag =$ witnessed-maxTag(ρ_1) to a complete quorum, say Q_k. By Lemma A.5, every server $s \in Q_k$ replies with a tag

$$\tau_s(\rho_1) \geq \text{witnessed-maxTag}(\rho_1) \Rightarrow \tau_s(\rho_1) \geq \tau_r(\rho_1) \tag{2}$$

So ρ_2 will observe (by Lemma A.1) that at least $\forall s \in Q_j \cap Q_k$, $\tau_s(\rho_2) \geq \tau_r(\rho_1)$. Hence by Lemma A.7 ρ_2 returns a tag $\tau_{r'}(\rho_2) \geq \min(\tau_s(\rho_2))$ for $s \in Q_j \cap Q_k$, and thus $\tau_{r'}(\rho_2) \geq \tau_r(\rho_1)$ and this completes our proof. □

Lastly the following lemma states that if two read operations return two different tags then the values that correspond to these tags are also different.

[1] read-ack occurs only if all phases reach a fix point and the $status$ variable becomes equal to $done$

Lemma A.11 *If ρ_1 and ρ_2 two read operations from readers r and r' respectively, such that ρ_1 (resp. ρ_2) returns the value written by ω_1 (resp. ω_2), then if $\tau_r(\rho_1) \neq \tau_{r'}(\rho_2)$ then ω_1 is different than ω_2 otherwise they are the same write.*

Proof. This lemma is ensured because a unique tag is associated to each written value by the writers. So it cannot be the case that two readers such that $\tau_r(\rho_1) \neq \tau_{r'}(\rho_2)$ returned the same value. □

Using the above lemmas we can obtain:

Theorem A.12 *Algorithm* CWFR *implements a MWMR atomic read/write register.*

B Algorithm SFW

B.1 Formal Specification of SFW Algorithm

Algorithm SFW consist of four kinds of automata: writer $Writer_p$, reader $Reader_p$, server $Server_s$, and channels $Channel_{p,s}$ and $Channel_{s,p}$, for $p \in \mathcal{W} \cup \mathcal{R}$ and $s \in \mathcal{S}$. Figures 4-6 present the IOA specification of the server, writer and reader automata of algorithm SFW, respectively.

Signature:

Input:
$\mathsf{rcv}(m)_{p,s},\ m \in M,\ s \in \mathcal{S},\ p \in \mathcal{R} \cup \mathcal{W}$
fail_s

Output:
$\mathsf{send}(m)_{s,p},\ m \in M,\ s \in \mathcal{S},\ p \in \mathcal{R} \cup \mathcal{W}$

State:

$tag \in T$, initially $\{0, *\}$
$value \in V$, initially $\perp$
$pCount(p) \in \mathbb{N}^+, p \in \mathcal{R} \cup \mathcal{W}$, initially 0

$inprogress \subseteq T \times V \times (\mathcal{R} \cup \mathcal{W}) \times \mathbb{N}^+$
$confirmed \subseteq T \times V \times (\mathcal{R} \cup \mathcal{W}) \times \mathbb{N}^+$
$failed$, a Boolean initially **false**

Transitions:

Input $\mathsf{rcv}(\langle type, mtag, mvalue, opCount, Count\rangle)_{p,s}$
Effect:
  if $\neg failed \wedge count > pCount(p)$ then
    $pCount(p) \leftarrow count$
    if $tag < mtag$ then
      $(\langle tag.ts, tag.label, tag.opc\rangle, value) \leftarrow$
        $(\langle mtag.ts, mtag.label, mtag.opc\rangle, mvalue)$
    if $mtype = \mathsf{W}$ then
      $(tag.ts, tag.label, tag.opc) \leftarrow (tag.ts + 1, p, opCount)$
      $inprogress \leftarrow (inprogress - \langle\langle *, p, *\rangle, val\rangle) \cup \langle tag, mvalue\rangle$
    if $confirmed < mtag$ then
      $confirmed \leftarrow mtag$

Output $\mathsf{send}(\langle inprog, conf, pc\rangle)_{s,p}$
Precondition:
  $\neg failed$
  $\langle inprog, conf, pc\rangle$
    $\langle inprogress, confirmed, Counter(p)\rangle$
Effect:

Input fail_s
Effect:
  $failed \leftarrow$ **true**

Figure 4: Algorithm SFW, $Server_s$: Signature, State and Transitions

B.2 Correctness of SFW

We proceed to show the correctness (safety) of algorithm SFW, that is, to show that the algorithm satisfies the atomicity properties presented in Section 2. Let write-fix(π) (resp. read-fix(π)) denote the fix point of a write (resp. read) operation π, where the status becomes equal to $done$. In other words the write-fix(π) (resp. read-fix(π)) of a fast operation happens when write-phase1-fix (resp. read-phase1-fix) event occurs. If the write/read operation is slow then its fix point is reached when write-phase2-fix or read-phase2-fix occurs respectively. For the rest of the section we use $\tau_p(\pi)$ to denote the tag of a read/write operation π from a process p, after the read-fix(π) or write-fix(π) event of π respectively. For the server automaton $\tau_s(\pi)$ and $\tau_{confirmed_s(\pi)}$ denote the value of the tag and $confirmed$ variables respectively at server $s \in \mathcal{S}$ after the send action at s for π. Similarly $inprogress_s(\pi)$ denotes the set $inprogress$ which s sends to operation π and $\tau_{inprogress_s(\pi)} \in inprogress_s(\pi)$ the tag that the set contains for writer w that invoked operation π. Notice that a write operation can be characterized by the tuple $\langle w, wc\rangle$ where w is the id of the writer that invokes the write operation and wc the operation counter of w at the end of the write event. Thus we denote by $\omega = \langle w, wc\rangle$ and the tag assigned to ω by $\tau_w(\omega) = \langle ts, \omega\rangle$, where $ts \in \mathbb{N}$ the timestamp included in the tag. From this and by well-formedness assumption, it follows that two write operations $\omega_1 = \langle w, wc_1\rangle$ and $\omega_2 = \langle w, wc_2\rangle$ invoked by the same writer $w \in \mathcal{W}$, have the relation $\omega_1 \rightarrow \omega_2$ if and only if $wc_1 < wc_2$. Let for any process p, τ_p denote the value of the local tag variable at p. For a read/write operation

Signature:

Input:
$\mathsf{write}(v)_w,\ v \in V, w \in \mathcal{W}$
$\mathsf{rcv}(m)_{s,w},\ m \in M,\ s \in \mathcal{S}, w \in \mathcal{W}$
$\mathsf{fail}_w, w \in \mathcal{W}$

Output:
$\mathsf{send}(m)_{w,s},\ m \in M,\ s \in \mathcal{S}, w \in \mathcal{W}$
$\mathsf{write\text{-}ack}_w, w \in \mathcal{W}$

Internal:
$\mathsf{write\text{-}phase1\text{-}fix}_w, w \in \mathcal{W}$
$\mathsf{write\text{-}phase2\text{-}fix}_w, w \in \mathcal{W}$

State:

$tag \in T$, initially $\{0, *\}$
$value \in V$, initially $\perp$
$phase \in \{\mathsf{W}, \mathsf{RP}\}$, initially W
$pCount \in \mathbb{N}^+$, initially 0
$opc \in \mathbb{N}^+$, initially 0
$status \in \{idle, active, done\}$, initially $idle$
$srvAck \subseteq \mathcal{S} \times 2^T time(T)$, initially $\emptyset$
$failed$, a Boolean initially **false**

Transitions:

Input $\mathsf{write}(v)_w$
Effect:
  if $\neg failed \wedge status = idle$ then
    $status \leftarrow active$
    $srvAck \leftarrow \emptyset$
    $phase \leftarrow \mathsf{W}$
    $value \leftarrow v$
    $pCount \leftarrow pCount + 1$
    $opc \leftarrow opc + 1$

Input $\mathsf{rcv}(\langle inprogress, confirmed, count\rangle)_{s,w}$
Effect:
  if $\neg failed$ then
    if $status = active$ and $pCount = count$ then
      $srvAck \leftarrow srvAck \cup \{\langle s, inprogress, confirmed\rangle\}$

Output $\mathsf{send}(\langle type, mtag, mvalue, wc, count\rangle)_{w,s}$
Precondition:
  $status = active$
  $\neg failed$
  $\langle type, mtag, mvalue, wc, counter\rangle =$
    $\langle phase, tag, value, opc, pCount\rangle$
Effect:
  none

Output $\mathsf{write\text{-}ack}_w$
Precondition:
  $status = done$
  $\neg failed$
Effect:
  $status \leftarrow idle$

Internal $\mathsf{write\text{-}phase1\text{-}fix}_w$
Precondition:
  $\neg failed$
  $status = active$
  $phase = \mathsf{W}$
  $\exists Q \in \mathbb{Q} : Q \subseteq \{s : (s, ., .) \in srvAck\}$
Effect:
  $T \leftarrow \{\langle t, w, .\rangle : \langle t, w, .\rangle \in \cup_{s \in Q} inprogress \wedge \langle s, inprogress, .\rangle \in srvAck\}$
  if $\exists \tau, MS, A : \tau \in T \wedge$
      $MS = \{s : s \in Q \wedge \tau \in inprogress \wedge \langle s, inprogress, .\rangle \in srvAck\} \wedge$
      $A \subseteq \mathbb{Q}$ s.t. $0 \leq |A| \leq \frac{n}{2} - 1 \wedge (\cap_{Q' \in (A \cup Q)} Q') \subseteq MS$ then
   $\langle tag.ts, tag.label, tag.opc\rangle \leftarrow \langle t, w, opc\rangle$
   if $|A| \geq \max(0, \frac{n}{2} - 2)$ then
    $phase \leftarrow \mathsf{RP}$
    $pCount \leftarrow pCount + 1$
   else
    $status \leftarrow done$
  else
   $\langle tag.ts, tag.label, tag.opc\rangle \leftarrow \max_{\langle t,w,opc\rangle \in \bigcup_{s \in Q} s.inprogress}(\langle t, w, opc\rangle)$
   $pCount \leftarrow pCount + 1$
   $phase \leftarrow \mathsf{RP}$
  $srvAck \leftarrow \emptyset$

Internal $\mathsf{write\text{-}phase2\text{-}fix}_w$
Precondition:
  $status = active$
  $\neg failed$
  $phase = \mathsf{RP}$
  $\exists Q \in \mathbb{Q} : Q \subseteq \{s : (s, ., .) \in srvAck\}$
Effect:
  $status \leftarrow done$

Input fail_w
Effect:
  $failed \leftarrow$ **true**

Figure 5: Algorithm SFW, $Writer_w$, $w \in \mathcal{W}$: Signature, State and Transitions

Signature:

Input:
$\mathsf{read}_r,\ r \in \mathcal{R}$
$\mathsf{rcv}(m)_{s,r},\ m \in M,\ r \in \mathcal{R},\ s \in \mathcal{S}$
$\mathsf{fail}_r,\ r \in \mathcal{R}$

Output:
$\mathsf{send}(m)_{r,s},\ m \in M,\ r \in \mathcal{R},\ s \in \mathcal{S}$
$\mathsf{read\text{-}ack}(v)_r,\ v \in V,\ r \in \mathcal{R}$

Internal:
$\mathsf{read\text{-}phase1\text{-}fix}_r$
$\mathsf{read\text{-}phase2\text{-}fix}_r$

State:

$tag \in T \times \mathcal{W}$, initially $\{0, *\}$
$value \in V$, initially v_o
$phase \in \{\mathsf{R}, \mathsf{RP}\}$, initially R
$pCount \in \mathbb{N}^+$, initially 0
$status \in \{idle, active, done\}$, initially $idle$
$srvAck \subseteq \mathcal{S}$, initially $\emptyset$
$srvConfirmed \subseteq \mathcal{S} \times T$, initially $\emptyset$
$srvInProgress \subseteq \mathcal{S} \times T$, initially $\emptyset$ $failed$, a Boolean initially **false**

Transitions:

Input read_r
Effect:
  if $\neg failed \wedge status = idle$ then
   $phase \leftarrow \mathsf{R}$
   $status \leftarrow active$
   $pCount \leftarrow pCount + 1$

Input $\mathsf{rcv}(\langle inprog, conf, pc\rangle)_{s,r}$
Effect:
  if $\neg failed \wedge status = active$ then
   if $pCount = pc$ then
    $srvAck \leftarrow srvAck \cup \{\langle s, inprogress, confirmed\rangle\}$

Output $\mathsf{send}(\langle type, mtag, mvalue, wc, count\rangle)_{r,s}$
Precondition:
  $status = active$
  $\neg failed$
  $\langle type, mtag, mvalue, wc, count\rangle =$
    $\langle phase, tag, value, tag.opc, pCount\rangle)$
Effect:
  none

Output $\mathsf{read\text{-}ack}(v)_r$
Precondition:
  $\neg failed$
  $status = done$
  $v = retvalue$
Effect:
  $replyQ \leftarrow \emptyset$
  $srvAck \leftarrow \emptyset$
  $status \leftarrow idle$

Input fail_r
Effect:
  $failed \leftarrow true$

Internal $\mathsf{read\text{-}phase1\text{-}fix}_r$
Precondition:
  $\neg failed$
  $status = active$
  $phase = \mathsf{R}$
  $\exists Q \in \mathbb{Q} : Q \subseteq \{s : (s, ., .) \in srvAck\}$
Effect:
  $maxCT \leftarrow \max_{s \in Q}(s.confirmed)$
  $inPtag = \{\tau : \tau \in \bigcup_{s \in Q} s.inprogress\}$
  if $\exists \tau, MS, B$ **:**
   $\tau > maxCT \wedge$
   $maxInPtags = \{\tau' : \tau' \in s.inprogress \wedge s \in Q \wedge \tau' > \tau\} \wedge$
   $\tau = \max_{\tau'' \in inPtag - maxInPtags}(\tau'') \wedge$
   $MS = \{s : s \in Q \wedge \tau \in s.inprogress\} \wedge$
   $B \subseteq \mathbb{Q}$s.t. $0 \leq |B| \leq \frac{n}{2} - 2 \wedge (\cap_{Q' \in (B \cup Q)} Q') \subseteq MS$ then
    $\langle\langle tag.ts, tag.label, tag.opc\rangle, value\rangle \leftarrow \langle\langle \tau.ts, \tau.w, \tau.opc\rangle, \tau.value\rangle$
    if $|B| = \max(0, \frac{n}{2} - 2)$ then
     $phase \leftarrow \mathsf{RP}$
    else
     $status \leftarrow done$
  else
   $MC \leftarrow \{s : s \in Q \wedge s.confirmed = maxCTag\}$
   $\langle tag.ts, tag.label, tag.opc\rangle \leftarrow \langle maxCT.ts, maxCT.w, maxCT.opc\rangle$
   $value \leftarrow maxCT.value$
   if $\exists C : C \subseteq \mathbb{Q} \wedge |C| \leq m - 2 \wedge (\cap_{Q' \in C} Q') \cap Q \subseteq MC$ then
    $status \leftarrow done$
   else
    $phase \leftarrow \mathsf{RP}$

Internal $\mathsf{read\text{-}phase2\text{-}fix}_r$
Precondition:
  $\neg failed$
  $status = active$
  $phase = 2$
  $\exists Q \in \mathbb{Q} : Q \subseteq \{s : (s, ., .) \in srvAck\}$
Effect:
  $status \leftarrow done$
  $phase \leftarrow 1$

Figure 6: Algorithm SFW, $Reader_r$: Signature, State and Transitions

π invoked from a reader/'writer process p, we denote by $start(\tau_p(\pi))$ the value of the tag variable at the read/write event of π. Finally let $\mathbb{Q}^i \subseteq \mathbb{Q}$ s.t. $|\mathbb{Q}^i| = i$.

Recall that the two predicates used in the algorithm are the following assuming that an operation received messages from a quorum Q:

Writer predicate for a write ω (PW): $\exists\ \tau, A, MS$ where: $\tau \in \{\langle ., \omega\rangle : \langle ., \omega\rangle \in inprogress_s(\omega) \wedge s \in Q\}$, $A \subseteq \mathbb{Q}, 0 \leq |A| \leq \frac{n}{2} - 1$, and $MS = \{s : s \in Q \wedge \tau \in inprogress_s(\omega)\}$, s.t. either $|A| \neq 0$ and $I_A \cap Q \subseteq MS$ or $|A| = 0$ and $Q = MS$.

Reader predicate for a read ρ (PR): $\exists\ \tau, B, MS$, where: $\max(\tau) \in \bigcup_{s \in Q} inprogress_s(\rho)$, $B \subseteq \mathbb{Q}, 0 \leq |B| \leq \max(0, \frac{n}{2} - 2)$, and $MS = \{s : s \in Q \wedge \tau \in inprogress_s(\rho)\}$, s.t. either $|B| \neq 0$ and $I_B \cap Q \subseteq MS$ or $|B| = 0$ and $Q = MS$.

Notice that the "either..or" condition of the above predicates can be merged and written as $I_{A\cup\{Q\}} \subseteq MS$ since $I_{\emptyset\cup\{Q\}} = Q$. Thus we will use this notation throughout the proofs. The writer predicate is located in write-phase1-fix of Figure 5 and the reader predicate is located in read-phase1-fix of Figure 6.

Lemma B.1 *For each server process $s \in \mathcal{S}$, τ_s is alphanumerically monotonically nondecreasing and contains a $\tau_s.ts > 0$.*

Proof. It is easy to see that a server s modifies its tag variable only if the tag τ in a received message m is such that $\tau > \tau_s$. In addition, if m is a write message, $\tau_s.ts = \max(\tau_s.ts, \tau.ts) + 1$. So in both cases the server's tag is monotonically increasing. Furthermore since the initial tag of the server is set to $\langle 0, \min wid\rangle$ and since $\tau > \tau_s$ only if $\tau.ts \geq \tau_s.ts$, then $\tau_s.ts$ is always greater than 0. □

Lemma B.2 *For each server process $s \in \mathcal{S}$ the variable $confirmed_s$ of a server s is alphanumerically monotonically nondecreasing.*

Proof. From the algorithm it follows that the server s modifies the value of its $confirmed$ variable only if the tag τ in a message m received by s, is such that $\tau > confirmed_s$. □

Lemma B.3 *For each server process $s \in \mathcal{S}$ the tag maintained in the $inprogress_s$ set for a writer $w \in \mathcal{W}$ is alphanumerically monotonically increasing.*

Proof. Notice that the server s maintains just a single record for a writer w in its $inprogress_s$ set. Each time the server receives a new write request from a write operation $\omega = \langle w, wc\rangle$ from w, for $wc > 0$, it generates a new tag for $\tau_{inprogress_s(\omega)} = \langle \tau_s.ts + 1, \omega\rangle$. Since by Lemma B.1 the local tag of the server s is non decreasing and it becomes equal to any tag it assigns to a write operation, then τ_s is greater or equal to any tag associated with a previous write operation $\omega' = \langle w, wc'\rangle$, for $wc' < wc$, from w. Thus $\tau_{inprogress_s(w,wc)} > \tau_{inprogress_s(w,wc')}$, and the claim follows. □

Lemma B.4 *If a server $s \in \mathcal{S}$ receives a message m from a process π, that contains a tag τ then s replies to π with a $\tau_{confirmed_{s_i}(\rho)} = \langle ts', wid, wc'\rangle$ s.t. $\tau_{confirmed_{s_i}(\pi)} \geq \tau$.*

Proof. It follows by Lemma B.2 that s upgrades the $confirmed_s$ variable only if the tag enclosed in the message $\tau > confirmed_s$. If so then the s replies with $\tau_{confirmed_s(\pi)} = \tau$; otherwise it replies with $\tau_{confirmed_s(\pi)} \geq \tau$ to operation π. □

Lemma B.5 *If server $s \in \mathcal{S}$ receives a message from a process $w \in \mathcal{W}$, for the first communication round of a write operation ω (i.e. type* W*) that contains a tag τ, then s replies with an $inprogress_s(\omega)$ which contains $\tau_{inprogress_s(\omega)} > \tau$ and $\tau_{inprogress_s(\omega)} = \max_{\tau' \in inprogress_s(\omega)}(\tau')$.*

Proof. The server s may receive W message from the write operation ω. The server checks if the tag τ enclosed in the received message is higher than its local tag, and if so updates its tag to be equal to τ. Thus after this update it is true that the tag of s, $\tau_s \geq \tau$. If W message m is received from ω, then s generates a new tag $\tau' = \langle \tau_s.ts + 1, w, m.wc\rangle$. Since the timestamp contained in the new tag is greater than the timestamp in the local tag of the server then $\tau' > \tau_s$ and thus $\tau' > \tau$ as well. Then s replaces any previous operations from w in its $inprogress$ set and inserts the new tag. Since the tuple $\langle w, m.wc\rangle$ unifies ω from w then τ' is the unique value of ω in server s. Thus it follows that $\tau_{inprogress_s(\omega)} = \tau'$ and hence $\tau_{inprogress_s(\omega)} > \tau$.

For the second part of the proof notice that any tag added in the $inprogress$ set of s contains the timestamp of the local tag of s along with the id of the writer and the writer's operation counter. Furthermore, since by Lemma B.1 the tag of a server is monotonically incremented then, when the $\mathsf{rcv}_{w,s}$ event happens, $\tau_s \geq \max_{\tau \in inprogress_s}(\tau)$. Since the new tag entered in the set is $\tau_{inprogress_s(\omega)} = \langle \tau_s.ts + 1, \omega\rangle$ then it follows that $\tau_{inprogress_s(\omega)} > \tau_s$ and hence $\tau_{inprogress_s(\omega)} > \max_{\tau \in inprogress_s}(\tau)$. Therefore $\tau_{inprogress_s(\omega)}$ is the maximum tag in the set. That completes the proof. □

The following lemma shows the uniqueness of each tag in the $inprogress$ set of any server.

Lemma B.6 *If a server $s \in \mathcal{S}$ maintains two tags $\tau_1, \tau_2 \in inprogress_s$, such that $\tau_1 = \langle ts_1, w_1, wc_1\rangle$ and $\tau_2 = \langle ts_2, w_2, wc_2\rangle$, then $w_1 \neq w_2$ and $ts_1 \neq ts_2$.*

Proof. The first part of the lemma, namely that $w_1 \neq w_2$, follows from the fact that the server s adds a new tag for a write operation from w_1 by removing any previous tag in $inprogress_s$ associated with a previous write from w_1. Hence only a single write operation is recorded in the $inprogress_s$ for every writer process $w \in \mathcal{W}$, and thus our claim follows.

Let us assume, for the second part of the lemma, that w.l.o.g server s receives the message from ω_1 before receiving the message from ω_2. Before replying to ω_1, s adds in the $inprogress_s$ set the tag $\tau_1 = \langle \tau_s.ts + 1, w_1, wc_1 \rangle$, and sets $\tau_s = \tau_1$. The server s repeats the same process for ω_2. Since by Lemma B.1 the local tag τ_s of the server in monotonically non-decreasing, then it follows that $\tau_s \geq \tau_1$ when s receives the message from ω_2. Thus if $\tau_2 = \langle \tau_s.ts + 1, w_2, wc_2 \rangle$, then $\tau_2 \geq \langle \tau_1.ts + 1, w_2, wc_2 \rangle$, and hence $ts_2 \geq ts_1 + 1$. So $ts_2 > ts_1$ and the lemma follows. □

Lemma B.7 *For each writer process $w \in \mathcal{W}$ the τ_w is monotonically increasing and contains a non-negative timestamp.*

Proof. Each writer process w modifies its local tag during its first communication round. When the write-phase1-fix event happens for a write operation ω, τ_w becomes equal to either the tag that satisfies the predicate or the maximum tag, both derived from the $inprogress$ sets of the servers that replied to a write operation of w. So it suffice to show that $start(\tau_w(\omega)) < \min_{s \in Q}(\tau_{inprogress_s(\omega)})$, assuming that all the servers of a quorum $Q \in \mathbb{Q}$, received messages and replied to w, for ω. Notice that every message sent from w to any server $s \in Q$ (when $\mathsf{send}_{w,s}$ occurs), contains a tag $\tau = start(\tau_w(\omega))$. Since by Lemma B.5, every server $s \in Q$ replies with $\tau_{inprogress_s(\omega)} > \tau$ (to any communication round of ω), then $\tau_{inprogress_s(\omega)} > start(\tau_w(\omega))$ and the claim follows. Furthermore by Lemma B.1 and Lemma B.5 it follows that w contains non-negative timestamps as well. □

Lemma B.8 *For each reader process $r \in \mathcal{R}$ the τ_r is monotonically nondecreasing and contains a non-negative timestamp.*

Proof. The tag variable at r is modified only if r invokes some read operation ρ and becomes equal to either the maximum tag in the $inprogress$ set or the maximum $confirmed$ tag obtained from the servers that replied. Notice however that $\tau_r(\rho)$ is equal to some $\tau_{inprogress_*(\rho)}$ only if $\tau_{inprogress_*(\rho)} > \max(\tau_{confirmed_*(\rho)})$. So it suffices to show that $\max(\tau_{confirmed_*(\rho)}) \geq start(\tau_r(\rho))$. Assume that all the servers in a quorum $Q \in \mathbb{Q}$ replied to ρ. Since ρ includes its $start(\tau_r(\rho))$ in every message sent during the event $\mathsf{send}_{r,s}$ to any server $s \in Q$, then by Lemma B.4, s replies with a $\tau_{confirmed_s(\rho)} \geq start(\tau_s(\rho))$. Hence it follows that $\max(\tau_{confirmed_*(\rho)}) \geq start(\tau_\rho())$ as well and our claim holds. Also since by Lemma B.1 all the servers reply with a non negative timestamp, then it follows that r contains non-negative timestamps as well. □

Lemma B.9 *For each process $p \in \mathcal{R} \cup \mathcal{W} \cup \mathcal{S}$, τ_p is monotonically nondecreasing and contains a non-negative timestamp.*

Proof. Follows from Lemmas B.1, B.7 and B.8 □

Lemma B.10 *If a read/write operation π receives replies from a quorum $Q \in \mathbb{Q}$ and observes two tags τ_1 and τ_2 for a write operation $\omega = \langle w, wc \rangle$, s.t. $\tau_1 = \langle t_1, w, wc \rangle$, $\tau_2 = \langle t_2, w, wc \rangle$, $t_1 \neq t_2$ and τ_1 is propagated in a k-wise intersection, then τ_2 is propagated in at least k-wise intersection as well iff $k > \frac{n+1}{2}$.*

Proof. Let $S_{\tau_1} \subseteq Q$ be a set of servers such that $\forall s \in S_{\tau_1}$ replied with $\tau_{inprogress_s(\omega)} = \tau_1$ to π and $S_{\tau_2} \subseteq Q$ the set of servers such that $\forall s' \in S_{\tau_2}$ replied with $\tau_{inprogress_{s'}(\omega)} = \tau_2$ to π. Since both τ_1 and τ_2 are propagated in a k-wise intersection and since every server maintains just a single copy in its $inprogress$ set for ω, then there exists two sets of quorums $\mathbb{Q}^k$ and $\mathbb{Q}'^k$ such that $I_{\mathbb{Q}^k} \subseteq S_{\tau_1}$ and $I_{\mathbb{Q}'^k} \subseteq S_{\tau_2}$ and $I_{\mathbb{Q}^k} \cap I_{\mathbb{Q}'^k} = \emptyset$. From the fact that $S_{\tau_1}, S_{\tau_2} \subseteq Q$, it follows that $Q \in \mathbb{Q}^k, \mathbb{Q}'^k$. So $I_{\mathbb{Q}^k} = I_{\mathbb{Q}^{k-1}} \cap Q$ and $I_{\mathbb{Q}'^k} = I_{\mathbb{Q}'^{k-1}} \cap Q$, and hence $I_{\mathbb{Q}^k} \cap I_{\mathbb{Q}'^k} = I_{\mathbb{Q}^{k-1}} \cap I_{\mathbb{Q}'^{k-1}} \cap Q = \emptyset$. By definition we know that $\mathbb{Q}^i$ is the quorum set that contains i quorums. So the intersection contains at most $k-1+k-1+1 = 2k-1$ quorums. Since we assume an n-wise intersection then the two sets of quorums maintain an empty intersection only if they consist of more than n quorums. Hence it follows that the intersection is empty if and only if:

$$2k - 1 > n \Leftrightarrow k > \frac{n+1}{2}$$

This completes the proof. □

Lemma B.11 *If T is the set of tags witnessed by a write operation ω from a writer w, during its first communication round, and $\tau \in T$ a tag that satisfies the writer predicate, then $\nexists \tau' \in T$ such that τ' satisfies the writer predicate as well.*

Proof. Let us assume to derive contradiction that there exist a pair of tags $\tau, \tau' \in T$ that both satisfy the writer predicate. Furthermore assume that the write operation $cont_w(Q, \omega)$. According to the predicate a write operation accepts a tag only if $\exists A \subset \mathbb{Q}$ such that $|A| \in [0 \dots \frac{n}{2} - 1]$ and the tag is contained in all the servers $s \in I_{A \cup \{Q\}}$. If the predicate is valid for τ with $|A| = 0$ then clearly all the servers $s \in Q$ reply with $\tau_{inprogress_s(\omega)} = \tau$. Thus the write operation does not observe any server $s' \in Q$ that replies with $\tau_{inprogress_{s'}(\omega)} = \tau'$ and hence τ' cannot satisfy the predicate, contradicting our assumption.

Note that since we assume n-wise intersections any i-wise intersection ($i < n - 1$) contains an $i + 1$-wise intersection so if τ satisfies the predicate with $|A| < \frac{n}{2} - 1$ it also satisfies it with $|A| = \frac{n}{2} - 1$. Therefore it suffices to consider the later case. So if now τ satisfies the predicate with $|A| = \frac{n}{2} - 1$ then it follows that there exists an intersection $I_{A \cup \{Q\}}$ such that every $s \in I_{A \cup \{Q\}}$ replied to w with $\tau_{inprogress_s(\omega)} = \tau$. Since A is a set of quorums that contains $\frac{n}{2} - 1$ members then $|A \cup \{Q\}| = \frac{n}{2}$ and thus τ is propagated in an $\frac{n}{2}$-wise intersection. Since $\frac{n}{2}$ is smaller than $\frac{n+1}{2}$ then by Lemma B.10, τ' cannot be propagated in $\frac{n}{2}$-wise intersection and thus can only be propagated in at least $(\frac{n}{2} + 1)$-wise intersection. That however means that $\exists A'$ such that $|A' \cup \{Q\}| = \frac{n}{2} + 1$, and hence $|A'| = \frac{n}{2}$, and $\forall s' \in I_{A' \cup \{Q\}}, \tau_{inprogress_{s'}(\omega)} = \tau'$. Since the predicate is only satisfied if $A' \in [0 \dots \frac{n}{2} - 1]$ then it follows that A' does not satisfy the predicate. That contradicts our assumption and completes our proof. □

Lemma B.12 *If a write operation ω from w witnesses multiple tags and sets $\tau_w(\omega) = \tau$, then any read operation ρ from r that returns the value written by ω decides a tag $\tau_r(\rho) = \tau = \tau_w(\omega)$.*

Proof. We proceed in cases and we show that either the read operation returns the value written by ω and $\tau_r(\rho) = \tau_w(\omega)$ or the case is impossible and thus ρ does not return the value written by ω. Let us assume w.l.o.g. that $cont_w(Q_j, \omega)$ and $cont_r(Q_i, \rho)$, during their first communication round. There are two cases to consider for the write operation: (1) ω is fast and completes in one communication round, or (2) ω is slow and performs two communication rounds. Let us examine the two cases separately.

Case 1: Here the write operation ω is fast and thus its predicate is valid and completes in a single communication round. Since ω is fast then there is a set $M = \{s : s \in Q_j \wedge \langle \tau_w(\omega), \omega.w, \omega.w_c \rangle \in s.inprogress\}$ and a set of quorums $|A| \in [0 \dots \frac{n}{2} - 3]$ s.t. $I_{A \cup \{Q_j\}} \subseteq M$. Every server $s \in I_{A \cup \{Q_j, Q_i\}}$ replies with a $\tau_{inprogress_s(\omega)} = \tau_w(\omega)$ to ρ, if s receives messages from ω before ρ. Otherwise s replies with a tag for an older write operation of w. Since according to the predicate $|A| \leq \frac{n}{2} - 3$ then the union of the set $|A \cup \{Q_j, Q_i\}| \leq \frac{n}{2} - 1$ (strictly less if $Q_i = Q_j$ or $Q_i \in A$). Thus the intersection $I_{A \cup \{Q_j, Q_i\}}$ involves at most $|A| + 2 \leq \frac{n}{2} - 1$ quorums and hence by Lemma B.10 every tag $\tau' \neq \tau_w(\omega)$ is observed by ρ in a k-wise intersection, such that $k > \frac{n}{2} - 1$. Thus ρ, either observes a $B \subseteq A \cup Q_j$, such that $\forall s \in I_{B \cup \{Q_i\}}$ reply with $\tau_s(\rho) = \tau_w(\omega)$, and hence the predicate is valid for $|B| \leq |A| + 1 \leq \frac{n}{2} - 2$ and returns $\tau_w(\omega)$, or since no other tag satisfies its predicate it returns a value of a write $\omega' \neq \omega$. Notice that since the predicate is false for any read operation ρ' succeeding or concurrent with ρ then no tag other than $\tau_w(\omega)$ is propagated in the confirmed variable of any server for ω. Hence if ρ returns the value written by ω, then it returns a tag $\tau_r(\rho) = \tau_w(\omega)$.

Case 2: Here the write operation ω is slow. This may happen in three cases: (a) either the predicate was true with $|A| = \frac{n}{2} - 2$ or $|A| = \frac{n}{2} - 1$, or (b) the predicate was false and thus no tag $\tau \in T$ received from a set of servers $MS = \{s : s \in Q_j \wedge \tau \in s.inprogress\}$ such that $\exists |A| \leq \frac{n}{2} - 1$ and $I_{A \cup \{Q_j\}} \subseteq MS$.

Case 2a: Here the predicate was true with $|A| = \frac{n}{2} - 2$ or $|A| = \frac{n}{2} - 1$. Notice that the read operation ρ may observe the tag $\tau_w(\omega)$ in the intersection $I_{A \cup \{Q_j, Q_i\}}$. Thus the set $|B| \leq |A \cup \{Q_j\}|$ which in the first case it would be $|B| \leq \frac{n}{2} - 1$ and in the second case $|B| \leq \frac{n}{2}$. We should consider the two cases for A separately. Notice that since ω does not modify the inprogress set during its second communication round then the read ρ observes the same values in that set no matter if it succeeds the first or second communication round of ω. So our claims are valid for both cases.

Case 2a(i): Here $\tau_w(\omega)$ is propagated in the servers $s \in I_{A \cup \{Q_j\}}$, for $|A| = \frac{n}{2} - 2$. Since the value $\tau_w(\omega)$ is sent by any server $s \in I_{A \cup \{Q_j, Q_i\}}$ to ρ, then there are three possible cases for Q_i and B:

1) $Q_i = Q_j \Rightarrow |B| = |A| = \frac{n}{2} - 2$,
2) $Q_i \in A \Rightarrow |B| = |A - Q_i| = \frac{n}{2} - 3$,
3) $Q_i \notin A \cup \{Q_j\} \Rightarrow |B| = |A \cup \{Q_j\}| = \frac{n}{2} - 1$

By Lemma B.10 it follows that for any of the above cases, any tag $\tau' \neq \tau_w(\omega)$ may be propagated in a k-wise intersection, such that $k > \frac{n}{2} + 1$ and thus $|B| > \frac{n}{2}$ for such a tag.

In the first two cases the predicate of ρ holds since $|B| \leq \frac{n}{2} - 2$, and thus ρ returns $\tau_r(\rho) = \tau_w(\omega)$ for ω. It remains to examine the third case where $|B| = |A \cup \{Q_j\}| = \frac{n}{2} - 1$. In this case, $|B \cup \{Q_i\}| = \frac{n}{2}$, and thus by Lemma B.10, none of the tags assigned to ω will satisfy the predicate. So ρ returns the value of ω if it observers $\max_{s \in Q_i}(\tau_{confirmed_s(\rho)}) = \langle maxTs, \omega \rangle$, and hence $\langle maxTs, \omega \rangle = \tau_w(\omega)$. Let $s \in Q_i$ be the server that replied to ρ with $\tau_{confirmed_s(\rho)} = \langle maxTs, \omega \rangle$.

Server s sets its confirmed tag to $\langle maxTs, \omega\rangle$ if it receives one of the following messages: (a) a W message for a write ω' such that $\omega \rightarrow \omega'$, (b) a RP from a second communication round of ω, (c) a RP from the second communication round of a read operation ρ' that returns a tag $\langle maxTs, \omega\rangle$, or (d) a R from a read operation ρ'' that already returned $\langle maxTs, \omega\rangle$. If (a) or (b) is true, and since the writer propagates the tag it returns in any of those messages, then $\langle maxTs, \omega\rangle = \tau_w(\omega)$. Thus we are down to the case that some reads propagated the tag in the confirmed variable. Since both ρ' and ρ'' should proceed or be concurrent to ρ then they are also concurrent or succeed the first communication round of ω. So either they observed (as ρ) the tag $\tau_w(\omega)$ in a set of quorums $|B| \leq \frac{n}{2} - 1$, or no tag for ω satisfy their predicate. Since $\tau_w(\omega)$ is the only tag that may satisfy their predicate, then both reads must propagate $\langle maxTs, \omega\rangle = \tau_w(\omega)$. So it follows that $\tau_r(\rho) = \tau_w(\omega)$ in this case as well.

Case 2a(ii): Let us assume in this case that the write operation received $\tau_w(\omega)$ from every server $s \in I_{A\cup\{Q_j\}}$, and $|A| = \frac{n}{2} - 1$. With similar reasoning as in Case 2a(i) we have the following cases for Q_i and B for ρ:

1) $Q_i = Q_j \Rightarrow |B| = |A| = \frac{n}{2} - 1$,
2) $Q_i \in A \Rightarrow |B| = |A - Q_i| = \frac{n}{2} - 2$,
3) $Q_i \notin A \cup \{Q_j\} \Rightarrow |B| = |A \cup \{Q_j\}| = \frac{n}{2}$

Observe again that in the first two cases and by Lemma B.10 no other tag $\tau' \neq \tau_w(\omega)$ for ω is propagated in less than $\frac{n}{2}$-wise intersection, and thus τ' does not satisfy the predicate for ρ.

If $Q_i \in A$, then the predicate is satisfied with $|B| = \frac{n}{2} - 2$ for ρ and thus it returns $\tau_w(\omega)$. Also if $Q_i = Q_j$ and $|B| = \frac{n}{2} - 1$ then as showed in case 2a(i) ρ also returns $\tau_r(\rho) = \tau_w(\omega)$.

So it remains to examine the case where $Q_i \notin A \cup \{Q_j\}$ and $|B| = \frac{n}{2}$. It follows that $|B \cup \{Q_i\}| = \frac{n}{2} + 1$ and $\forall s \in I_{B\cup\{Q_i\}}$, $\tau_w(\omega) \in s.inprogress$. The read operation ρ may decide to return a tag τ' different from $\tau_w(\omega)$, if there are servers $s' \in I_{C\cup\{Q_i\}}$ such that $\tau' \in s'.inprogress$ and $|C| \leq \frac{n}{2} - 2$. Furthermore it must be true that $(I_{B\cup\{Q_i\}}) \cap (I_{C\cup\{Q_i\}}) = \emptyset$ otherwise a server in that intersection would reply either with $\tau_{inprogress_s(\omega)} = \tau_w(\omega)$ or $\tau_{inprogress_s(\omega)} = \tau'$. It suffices then to show that the aforementioned intersection is impossible. Since we know that $|C| = \frac{n}{2} - 2$ and $|B| = \frac{n}{2}$ quorums then the intersection $I_B \cap I_C \cap Q_i$ contains $\frac{n}{2} + \frac{n}{2} - 2 + 1 = n - 1$ quorums. Since we assumed n-wise quorum system then the intersection $I_B \cap I_C \cap Q_i \neq \emptyset$. That will be true even if we assume a smaller C. So no tag in this case satisfies the predicate of ρ either and thus ρ returns the value written by ω only if it observers $\max_{s\in Q_i}(\tau_{confirmed_s(\rho)}) = \langle maxTs, \omega\rangle$. With similar arguments as in Case 2a(i), we can show that no read or the write operation will propagate a tag different than $\tau_w(\omega)$ for ω and thus no server replies with a tag $\langle ts, \omega\rangle \neq \tau_w(\omega)$. Thus if ρ returns ω, then $\tau_r(\rho) = \tau_w(\omega)$ in this case as well.

Case 2b: In this case the predicate was false for the write operation ω. So any tag received from ω was observed in a set $|I_{A\cup\{Q_j\}}|$ such that $|A| \geq \frac{n}{2}$. So let us split this case in two subcases:(i) $|A| = \frac{n}{2}$, and (ii) $|A| > \frac{n}{2}$.

Case 2b(i): Based on the three cases presented in case 2a for Q_i and B then ρ may observe one of the following distributions for $\tau_w(\omega)$:

1) $Q_i = Q_j \Rightarrow |B| = |A| = \frac{n}{2}$,
2) $Q_i \in A \Rightarrow |B| = |A - Q_i| = \frac{n}{2} - 1$,
3) $Q_i \notin A \cup \{Q_j\} \Rightarrow |B| = |A \cup \{Q_j\}| = \frac{n}{2} + 1$

Observe that neither of the cases satisfies the predicate for ρ. Furthermore in the first two cases ρ observes $\tau_w(\omega)$ in the intersection $I_{B\cup\{Q_i\}}$ and $|B \cup Q_i| \leq \frac{n}{2} + 1$. So according to Lemma B.10 no tag $\tau' \neq \tau_w(\omega)$ will be propagated in a k-wise intersection, such that $k < \frac{n}{2} + 1$ and hence τ' will not satisfy the predicate for ρ either.

It remains to examine deeper the third case where $\tau_w(\omega)$ is received from every server $s \in I_{B\cup\{Q_i\}}$, and $|B \cup Q_i| = \frac{n}{2} + 2$. We need to examine if there could be set of quorums C such that $|C| \leq \frac{n}{2} - 2$ and every server $s' \in I_{C\cup\{Q_i\}}$ reply to ρ with a tag $\tau' \neq \tau_w(\omega)$ for ω. Such a set would satisfy the predicate for ρ and thus ρ would return τ'. This is only possible if $I_B \cap I_C \cap Q_i = \emptyset$. Since $|B| = \frac{n}{2} + 1$ and $|C| = \frac{n}{2} - 2$ then the intersection consists of $|B| + |C| + 1 = \frac{n}{2} + 1 + \frac{n}{2} - 2 + 1 = n$ quorums. Since we assume an n-wise quorum system then it follows that the intersection is not empty. Thus there exist no τ' and hence no tag will satisfy the predicate of ρ in this case. Since ρ will return the value written by ω only if it observes a maximum confirmed tag such that $\max_{s\in Q_i}(\tau_{confirmed_s(\rho)}) = \langle maxTs, \omega\rangle$. But since the predicate will be false for every read operation then the first to confirm a tag for ω will be the writer w in the second communication round of ω. Since though ω returns $\tau_w(\omega)$ then it propagates that tag during its second round to a full quorum. Thus any read operation that returns the value of ω must observe $\max_{s\in Q_i}(\tau_{confirmed_s(\rho)}) = \langle maxTs, \omega\rangle = \tau_w(\omega)$ and hence returns $\tau_r(\rho) = \tau_w(\omega)$.

Case 2b(ii): Suppose now that the predicate does not hold for the writer because it observed every tag to be distributed in at least some intersection $I_{A\cup\{Q_j\}}$, where $|A| > \frac{n}{2}$. Let us assume that $|A| = \frac{n}{2} + 1$. By that it follows that the read operation ρ would observe any tag obtained by the writer in one of the following distributions:

1) $Q_i = Q_j \Rightarrow |B| = |A| = \frac{n}{2} + 1$,

2) $Q_i \in A \Rightarrow |B| = |A - Q_i| = \frac{n}{2}$,

3) $Q_i \notin A \cup \{Q_j\} \Rightarrow |B| = |A \cup \{Q_j\}| = \frac{n}{2} + 2$

Obviously none of the cases satisfy the predicate of ρ. Furthermore, in the first two cases, by Lemma B.10 and as shown in case 2b(i), no tag assigned to ω satisfies the predicate for ρ and so if ρ returns the value written by ω, it returns the value propagated during ω's second round.

Finally we need to explore what happens in the case where $|B| = \frac{n}{2} + 2$. So can we devise a tag τ' for ω such that is distributed in some set of quorums C, such that $|C| \leq \frac{n}{2} - 2$ and every server $s' \in I_{C \cup \{Q_i\}}$ reply to ρ with a tag $\tau' \neq \tau_w(\omega)$. Such a set would satisfy the predicate for ρ and thus ρ would return τ'. This is only possible if $I_B \cap I_C \cap Q_i = \emptyset$. Since $|B| = \frac{n}{2} + 2$ and $|C| = \frac{n}{2} - 2$ then the intersection consists of $|B| + |C| + 1 = \frac{n}{2} + 2 + \frac{n}{2} - 2 + 1 = n + 1$ quorums. Thus such intersection is possible. If however ρ observed τ' in an intersection $I_{C \cup \{Q_i\}}$ and since every server $s \in I_{C \cup \{Q_i, Q_j\}}$ replied to ω with a tag τ' before replying to ρ, then there are three possible distributions for the write operation ω for τ':

1) $Q_j = Q_i \Rightarrow |A| = |C| = \frac{n}{2} - 2$,

2) $Q_j \in C \Rightarrow |A| = |C - Q_j| = \frac{n}{2} - 3$,

3) $Q_j \notin C \cup \{Q_i\} \Rightarrow |A| = |C \cup \{Q_i\}| = \frac{n}{2} - 1$

This however shows that in any case the predicate for ω should have been true for the tag τ'. But this contradicts our initial assumption for this case that the predicate for ω is false and hence such a case is impossible. Thus no read operation would observe a different tag τ' that satisfies its predicate and so every read ρ that returns the value of ω must observe $\max_{s \in Q_i}(\tau_{confirmed_s(\rho)}) = \langle maxTs, \omega \rangle = \tau_w(\omega)$ and as shown in case 2b(i) $\tau_r(\rho) = \tau_w(\omega)$. □

Since Lemma B.12 shows that every read operation returns the same tag for the same write operation then from this point onwards we can say that different tags represent different write operations. This is presented formally by the following corollary:

Corollary B.13 *If two read operations ρ_1 and ρ_2 return tags $\tau_*(\rho_1) = \tau_*(\omega)$ and $\tau_*(\rho_2) = \tau_*(\omega')$ respectively, then if $\tau_*(\rho_1) = \tau_*(\rho_2)$ then $\omega = \omega'$ otherwise $\omega \neq \omega'$.*

Lemma B.14 *If a server $s \in \mathcal{S}$ replies with a $\tau_{inprogress_s(\omega)}$ to the write operation ω from w, then s replies to any subsequent message from an operation π with $inprogress_s(\pi)$, s.t. $\max_{\tau \in inprogress_s(\pi)}(\tau) \geq \tau_{inprogress_s(\omega)}$ if π is a read and $\max_{\tau \in inprogress_s(\pi)}(\tau) > \tau_{inprogress_s(\omega)}$ if π is a write.*

Proof. If π is a write operation then by Lemma B.5 s replies to π with $\tau_{inprogress_s(\pi)} = \max_{\tau \in inprogress_s(\pi)}(\tau)$. But before s adds $\tau_{inprogress_s(\pi)}$ in $inprogress_s$, according again to Lemma B.5, $\tau_{inprogress_s(\pi)}$ was greater than any tag in that set. Since s also added $\tau_{inprogress_s(\omega)}$ before replying to ω then it follows that $\tau_{inprogress_s(\pi)} = \max_{\tau \in inprogress_s(\pi)}(\tau) > \tau_{inprogress_s(\omega)}$.

If π is a read operation then by the algorithm server s receives either a R or RP message. In none of those cases s updates its $inprogress_s$ set. By Lemma B.5 when server s replied to ω, $\tau_{inprogress_s(\omega)} = \max_{\tau \in inprogress_s(\omega)}(\tau)$. Thus if s did not receive any write message between the message from ω and π then the operation observes $\max_{\tau \in inprogress_s(\pi)}(\tau) = \tau_{inprogress_s(\omega)}$. Otherwise, with the combination of the first part of this proof, π observes $\max_{\tau \in inprogress_s(\pi)}(\tau) > \tau_{inprogress_s(\omega)}$. Hence our claim follows. □

Lemma B.15 *If a read operation ρ from r receives a confirmed tag $\tau_{confirmed_s(\rho)}$ from a server s, then $\tau_r(\rho) \geq \tau_{confirmed_s(\rho)}$.*

Proof. Let the read operation ρ receive replies from the servers in Q_i. By the algorithm a read operation returns either the $\max_{s' \in Q_i}(\tau_{confirmed_{s'}(\rho)})$ or the maximum tag τ that satisfies predicate **PR**. Notice that if $\max_{s' \in Q_i}(\tau_{confirmed_{s'}(\rho)}) \geq \tau$ then the reader does not evaluate the predicate but rather returns $\tau_r(\rho) = \max_{s' \in Q_i}(\tau_{confirmed_{s'}(\rho)})$ in one or two communication rounds. Since ρ returns either $\tau_r(\rho) = \max_{s' \in Q_i}(\tau_{confirmed_{s'}(\rho)}) \geq \tau_{confirmed_s(\rho)}$ or $\tau_r(\rho) = \tau > \max_{s' \in Q_i}(\tau_{confirmed_{s'}(\rho)})$, then in either case $\tau_r(\rho) \geq \tau_{confirmed_s(\rho)}$. □

Lemma B.16 *If the* read *event of a read operation ρ from reader $r \in \mathcal{R}$ succeeds the* write-fix(ω) *event of a write operation ω from $w \in \mathcal{W}$ in an execution ξ then, $\tau_r(\rho) \geq \tau_w(\omega)$.*

Proof. Let assume that every server in the quorums $Q_i, Q_z \in \mathbb{Q}$ (not necessarily $Q_i \neq Q_z$) received the messages for the first and the second (if any) communication rounds of the write operation ω respectively and replied to those messages. Also let Q_j be the quorum that replied to the first communication round of ρ operation, not necessarily different than Q_j or Q_z. Moreover let $T = \{\langle ., \omega\rangle : s \in Q_i \wedge \langle ., \omega\rangle \in inprogress_s(\omega)\}$ be the set of tags witnessed by ω during its first communication round. Notice that either: (1) $\tau_w(\omega) = \tau$ such that $\tau \in T$ and its distribution satisfies the predicate **PW**, or (2) $\tau_w(\omega) = \max_{s\in Q_i}(\tau_{inprogress_s(\omega)})$, otherwise. We should investigate these two cases separately. The read operation returns a tag $\tau_r(\rho)$ equal to either the $\max_{s\in Q_j}(\tau_{confirmed_s(\rho)})$ or the maximum tag in $\bigcup_{s\in Q_j} inprogress_s(\rho)$ that satisfies predicate **PR**. Therefore if $\max_{s\in Q_j}(\tau_{confirmed_s(\rho)}) \geq \tau_w(\omega)$ or $\exists \tau \in \bigcup_{s\in Q_j} inprogress_s(\rho)$ s.t. $\tau > \tau_w(\omega)$ that satisfies **PR** then ρ returns $\tau_r(\rho) \geq \tau_w(\omega)$. Also notice that if w invokes a write operation ω' such that $\omega \rightarrow \omega'$ then by Lemmas B.3, B.5 and B.4 it follows that every server receiving messages from ω' will reply to ρ with $\tau_{confirmed_s(\rho)} \geq \tau_w(\omega)$ since w will include $\tau_w(\omega)$ in its next write operation. Thus ρ returns $\tau_r(\rho) \geq \tau_w(\omega)$ in this case as well. So it suffices to examine the case where there is no write ω' s.t. $\omega \rightarrow \omega'$ and no $\tau' \in \bigcup_{s\in Q_j} inprogress_s(\rho)$ s.t. $\tau' \geq \tau_w(\omega)$ and τ' satisfies the predicate for the read operation ρ.

Case 1: Observe that by Lemma B.11, $\tau_w(\omega)$ is the only tag that satisfies the writer predicate for the write operation $\omega = \langle w, wc\rangle$. In this case we need to consider the following subcases for the set of quorums A that satisfies the predicate **PW**: (a) $|A| < \frac{n}{2} - 2$ and thus the write operation is fast, or (b) $|A| \in [\frac{n}{2} - 2, \frac{n}{2} - 1]$ and thus the write operation is slow. Notice that by Lemma B.14 it follows that every server $s \in I_{A\cup\{Q_i,Q_j\}}$ replied to ρ with $inprogress_s(\rho)$ that contains a tag $\tau \geq \tau_w(\omega)$. Since we only examine the cases where no s received messages from a write ω' from w s.t. $\omega \rightarrow \omega'$, thus it must hold that $inprogress_s(\rho) = \tau_w(\omega)$.

Case 1a: This is the case where the write operation is fast and hence $|A| < \frac{n}{2} - 2$ and every server $s \in I_{A\cup\{Q_i\}}$ replies with $\tau_w(\omega) \in inprogress_s(\omega)$ to ω. The read operation ρ will witness the tag $\tau_w(\omega)$ from the servers in $I_{B\cup\{Q_j\}}$ where Q_j and B may be as follows:

1) $Q_j = Q_i \Rightarrow |B| = |A| < \frac{n}{2} - 2$,
2) $Q_j \in A \Rightarrow |B| = |A - Q_i| \leq \frac{n}{2} - 3$,
3) $Q_j \notin A \cup \{Q_i\} \Rightarrow |B| = |A \cup \{Q_i\}| \leq \frac{n}{2} - 2$

In any case $|B| \leq \frac{n}{2} - 2$ and thus the predicate **PR** is valid for the ρ for $\tau_w(\omega)$. Hence ρ returns $\tau_r(\rho) = \tau_w(\omega)$ in one or two communication rounds.

Case 1b: This is the case where $|A| \in [\frac{n}{2} - 2, \frac{n}{2} - 1]$ and thus the predicate holds for ω, but ω proceeds to a second communication round before completing. During its second round, ω propagates the tag $\tau_w(\omega)$ to a complete quorum say Q_z. Since $\omega \rightarrow \rho$ and by Lemma B.4, then any server $s \in Q_j \cap Q_z$ replies to ρ with a $\tau_{confirmed_s(\rho)} \geq \tau_w(\omega)$. Thus by Lemma B.15 $\tau_r(\rho) \geq \tau_{confirmed_s(\rho)}$, and hence $\tau_r(\rho) \geq \tau_w(\omega)$.

Case 2: In this case the predicate does not hold for ω. Thus the writer discovers the maximum tag among the ones it receives from the servers and propagates that to a full quorum say Q_z, not necessarily different from Q_j or Q_i. It follows that by Lemma B.2 ρ will receive a $\tau_{confirmed_s(\rho)} \geq \tau_w(\omega)$ from any server $s \in Q_i \cap Q_z$. Thus by Lemma B.15 ρ returns $\tau_r(\rho) \geq \tau_w(\omega)$ in this case as well. □

Lemma B.17 *If ω_1 and ω_2 are two write operations from the writers w and w' respectively, such that $\omega_1 \rightarrow \omega_2$ in ξ, then $\tau_{w'}(\omega_2) > \tau_w(\omega_1)$.*

Proof. First consider the case where $w = w'$ and thus ω_1 and ω_2 are two subsequent writes of the same writer. It is easy to see by Lemmas B.7 and B.3 that $\tau_w(\omega_1) > \tau_w(\omega_2)$ since the tag of the writer is monotonically increasing. So for the rest of the proof we focus on the case where ω_1 and ω_2 are invoked from two different writers $w \neq w'$. Let us assume that every server in the quorums $Q_i, Q_z \in \mathbb{Q}$ (not necessarily $Q_i \neq Q_z$) received the messages for first and the second (if any) communication rounds of the write operation ω_1 respectively and replied to those messages, and let Q_j be the quorum that replied to the first communication round of ω_2's operation, not necessarily different than Q_j or Q_z. Notice here that since ω_2 decides about the tag $\tau_{w'}(\omega_2)$ in its first communication round, then it suffices to examine ω_2's first communication round alone. Moreover let $T_1 = \{\langle ., \omega_1\rangle : s \in Q_i \wedge \langle ., \omega_1\rangle \in inprogress_s(\omega_1)\}$ be the set of tags witnessed by ω_1 during its first communication round and T_2 the respective set of tags for ω_2. Notice that either: (1) $\tau_w(\omega_1) = \tau$ such that $\tau \in T_1$ and its distribution satisfies the predicate **PW**, or (2) $\tau_w(\omega_1) = \max_{s\in Q_i}(\tau_{inprogress_s(\omega)})$, otherwise. We now study these two cases individually.

Case 1: This is the case where the predicate **PW** holds for ω_1. Thus according to the predicate there exists some set of quorums $|A| \leq \frac{n}{2} - 1$ such that $\forall s \in I_{A\cup\{Q_i\}}$, $\tau_{inprogress_s(\omega_1)} = \tau_w(\omega_1)$. From the predicate we can see that if $n \leq 4$ then $|A| \in [0, 1]$. So we can split this case into two subcases: (a) $n > 4$, and (b) $n \leq 4$.

Case 1a: Here we assume that $n > 4$ and thus the predicate may be satisfied with $|A| \leq \frac{n}{2} - 1$ and $\frac{n}{2} - 1 > 0$. From the monotonicity of the servers (Lemma B.1) and from Lemmas B.5 and B.6, it follows that every server $s' \in I_{A \cup \{Q_i, Q_j\}}$ replies with a $\tau_{inprogress_{s'}(\omega_2)} > \tau_{inprogress_{s'}(\omega_1)}$ and thus $\tau_{inprogress_{s'}(\omega_2)} > \tau_w(\omega_1)$. There are three subcases for Q_j: (i) $Q_j = Q_i$, (ii) $Q_j \in A$, or (iii) $Q_j \notin A \cup \{Q_i\}$. If one of the first two cases is true then ω_2 will observe a set of quorums $|C| \leq \frac{n}{2} - 1$ such that every server $s' \in I_{C \cup \{Q_j\}}$ will reply with a tag greater than $\tau_w(\omega_1)$. Since $|C \cup \{Q_j\} \geq \frac{n}{2}$, then according to Lemma B.10 no tag $\tau' < \tau_w(\omega_1)$ will be propagated in an intersection $I_{D \cup \{Q_j\}}$ such that $|D| \leq \frac{n}{2}$. Thus no such tag will satisfy predicate **PW** for ω_2. It follows that ω_2 returns a tag $\tau_{w'}(\omega_2)$ either because $\tau_{w'}(\omega_2) \in inprogress_s(\omega_2)$, and $s \in I_{C \cup \{Q_j\}}$ and **PR** is satisfied or $\tau_{w'}(\omega_2) = \max_{s \in Q_j}(\tau_{inprogress_s(\omega_2)})$. In both cases $\tau_{w'}(\omega_2) > \tau_w(\omega_1)$.

It remains to investigate the subcase (iii) where $Q_j \notin A \cup \{Q_i\}$. If $|A| \leq \frac{n}{2} - 2$ then ω_2 will observe a set of quorums $|C| \leq \frac{n}{2} - 1$ and the proof is similar as in cases (i) and (ii). If however $|A| = \frac{n}{2} - 1$ then before ω_1 completes it performs a second communication round and propagates $\tau_w(\omega_1)$ to a full quorum Q_z. But every server $s \in Q_z$ that receives this message sets its local tag to $\tau_s = \tau_w(\omega_1)$ if $\tau_w(\omega_1) > \tau_s$; otherwise they do not update their tag. Thus every server $s \in Q_z$ contain a tag $\tau_s \geq \tau_w(\omega_1)$ when ω_1 completes. Since by Lemma B.1 the local tag of a server is monotonically increasing, then by Lemmas B.5 and B.6, every server $s \in Q_j \cap Q_z$ reply with $\tau_{inprogress_s(\omega_2)} > \tau_w(\omega_1)$ to ω_2. So $|C| = |\{Q_z\}| = 1$. Since we assume that $n > 4$ then $|C| \leq \frac{n}{2} - 1$ and hence as before and by Lemma B.10 there cannot exist tag $\tau' < \tau_w(\omega_1)$ that satisfies the predicate for ω_2. Thus in this case $\tau_{w'}(\omega_2) = \tau_{inprogress_s(\omega_2)}$ for some $s \in Q_j \cap Q_z$ and hence $\tau_{w'}(\omega_2) > \tau_w(\omega_1)$.

Case 1b: Here $n \leq 4$. In this case it follows that the predicate is valid for ω_1 with $|A| \in [0, 1]$. If the predicate is valid for $|A| = 0$ then it follows that ω_1 received $\tau_w(\omega_1)$ from all the servers in Q_i while if $|A| = 1$ it received that tag from a pairwise intersection. Notice that the predicate for ω_1 holds for $|A| = 1$ only in the case where $n = 4$ and with $|A| = 0$ for $n \leq 3$. Thus in any case $I_{A \cup \{Q_j, Q_i\}} \neq \emptyset$. Hence in case the predicate does not hold for ω_2 and returns the $\max_{s \in Q_j}(\tau_{inprogress_s(\omega_2)})$ then $\tau_{w'}(\omega_2) > \tau_w(\omega_1)$. So it remains to explore the two cases where the predicate holds for ω_2.

If the predicate for ω_1 holds with $|A| = 0$ then it follows that all the servers $s \in Q_j \cap Q_i$ reply, by Lemmas B.5 and B.6, to ω_2 with $\tau_{inprogress_s(\omega_2)} > \tau_{inprogress_s(\omega_1)}$ and thus greater than $\tau_w(\omega_1)$. Notice that for ω_2 the predicate may also hold for a quorum set $|C| \in [0, 1]$. If the predicate for ω_2 holds with $|C| = 0$, then it follows that every server $s \in Q_j$ replies with $\tau_{inprogress_s(\omega_2)} = \tau_{w'}(\omega_2)$. Since the servers in $Q_j \cap Q_i$ reply with $\tau_{inprogress_s(\omega_2)} > \tau_w(\omega_1)$, then it follows that every server $s \in Q_j$ replies with that same tag, and hence $\tau_{w'}(\omega_2) > \tau_w(\omega_1)$. Otherwise, if $|C| = 1$, let us assume to derive contradiction that $C = \{Q_g\}$ for $Q_g \neq Q_i, Q_j$, and every server $s \in I_{C \cup \{Q_j\}} = Q_g \cap Q_j$ reply to ω_2 with a $\tau' < \tau_w(\omega_1)$. Since $\tau' < \tau_w(\omega_1)$, then it must be the case that $(Q_g \cap Q_j) \cap (Q_i \cap Q_j) = \emptyset$. Since we assume $|C| = 1$ it follows that $n = 4$ and hence this is impossible. Thus the predicate may only hold in this case for $C = \{Q_i\}$ and for a tag obtained by the servers in $Q_i \cap Q_j$ and hence $\tau_{w'}(\omega_2) > \tau_w(\omega_1)$.

If the predicate for ω_1 holds with $|A| = 1$ then ω_1 performs a second communication round propagating $\tau_w(\omega_1)$ to a full quorum, say Q_z. Thus every server $s \in Q_j \cap Q_z$ replies by Lemma B.5 with a tag $\tau_{inprogress_s(\omega_2)} > \tau_w(\omega_1)$. Since a full intersection replies to ω_2 with $\tau_{inprogress_s(\omega_2)} > \tau_w(\omega_1)$ then following similar analysis as in the previous case (and by Lemma B.10) we can show that there cannot exist tag $\tau' < \tau_w(\omega_1)$ to satisfy ω_2's predicate. Thus ω_2 retruns $\tau_{w'}(\omega_2) > \tau_w(\omega_1)$.

Case 2: In this case the predicate does not hold for ω_1 and thus proceeds to a second communication round propagating a tag $\tau_w(\omega_1) = \max s \in Q_j(\tau_{inprogress_s(\omega_1)}$ to a full quorum, say Q_z. Since every server $s \in Q_j \cap Q_z$ replies by Lemma B.5 with a tag $\tau_{inprogress_s(\omega_2)} > \tau_w(\omega_1)$ then by Lemma B.10 and following similar analysis as in Case 1b, we can show that there cannot exist tag $\tau' < \tau_w(\omega_1)$ to satisfy ω_2's predicate. Thus ω_2 retruns $\tau_{w'}(\omega_2) > \tau_w(\omega_1)$ in this case as well. □

Lemma B.18 *If ρ_1 and ρ_2 are two read operations from the readers r and r' respectively, such that $\rho_1 \rightarrow \rho_2$ in ξ, then $\tau_r(\rho_2) \geq \tau_{r'}(\rho_1)$.*

Proof. Let us assume w.lo.g. that ρ_1 to $scnt_r(\rho_1, Q_i)$ and $scnt_r(\rho_1, Q_j)$ (not necessarily different than Q_i) during its first and second communication round respectively. Moreover let ρ_2 to $scnt_{r'}(\rho_2, Q_j)$ during its first communication round. Notice here that since a read operation decides on the tag that it returns when read-phase1-fix happens then we only need to investigate the first communication round of ρ_2. Let us first consider the case where the two read operations are performed by the same reader, i.e. $r = r'$. In this case r will enclose in every message sent out a tag greater or equal to $\tau_r(\rho_1)$ Thus every server $s \in Q_j$, by Lemma B.4, replies to ρ_2 with $\tau_{confirmed_s(\rho_2)} \geq \tau_r(\rho_1)$. Thus by Lemma B.15 ρ_2 returns a tag $\tau_{r'}(\rho_2) \geq \tau_r(\rho_1)$.

So it remains to investigate the case where $r \neq r'$. Notice that if ρ_1 proceeds to a second communication round, either because the predicate holds for $|B| = \frac{n}{2} - 2$ or not enough confirmed tags where received, then ρ_1 propagates $\tau_r(\rho_1)$ in Q_z before completing. By Lemmas B.4 and B.15 it follows that every server $s \in Q_z \cap Q_j$ replies to ρ_2 with a $\tau_{confirmed_s(\rho_2)} \geq \tau_{r_1}(\rho_1)$ and thus ρ_2 returns $\tau_{r'}(\rho_2) \geq \tau_r(\rho_1)$ in one or two communication rounds. Thus we left to explore the case where ρ_1 is fast and returns in a single communication round. This may happen in two cases: (1) predicate

PR holds for ρ_1 with $|B| \leq \frac{n}{2} - 3$ or (2) ρ_1 returns the maximum confirmed tag which he observed in a k-intersection with $k \leq n - 1$. Let us examine those two cases.

Case 1: In this case ρ_1 returns $\tau_r(\rho_1)$ because it receives that tag from every server $s \in I_{B \cup \{Q_i\}}$, s.t. $|B| \leq \frac{n}{2} - 3$ and $\tau_r(\rho_1) = \tau_{inprogress_s(\omega)}$ for some write operation $\omega = \langle w, wc \rangle$ from writer w. Thus by Lemma B.14 every server $s \in I_{B \cup \{Q_i, Q_j\}}$, replies to ρ_2 with a $\tau_{inprogress_s(\omega')} \geq \tau_{r_1}(\rho_1)$ as the tag for a write ω' from the writer w. So there are two subcases to consider: (a) there exists server $s \in I_{B \cup \{Q_i, Q_j\}}$ such that replies with $\tau_{inprogress_s(\omega')} > \tau_{r_1}(\rho_1)$, and (b) all servers in $I_B \cap Q_i \cap Q_j$ reply with $\tau_{inprogress_s(\omega')} = \tau_{r_1}(\rho_1)$.

Case 1a: If there exists $s \in I_{B \cup \{Q_i, Q_j\}}$ such that $\tau_{inprogress_s(\omega')} > \tau_{r_1}(\rho_1)$, then it follows that the $\tau_{inprogress_s(\omega')} > \tau_{inprogress_s(\omega)}$ and thus writer w performed a write ω' such that $\omega \rightarrow \omega'$. But according to the algorithm the message that w sent to s for ω' contains a tag $\tau \geq \tau_w(\omega)$. Hence by Lemma B.4 s replies with $\tau_{confirmed_s(\omega)} \geq \tau$ to ω and thus, by monotonicity of the confirmed tag (Lemma B.2), s replies with $\tau_{confirmed_s(\rho_2)} \geq \tau \geq \tau_w(\omega)$ to ρ_2 as well. Therefore from Lemma B.15 ρ_2 returns a tag $\tau_{r'}(\rho_2) \geq \tau_w(\omega)$ and thus $\tau_{r'}(\rho_2) \geq \tau_r(\rho_1)$.

Case 1b: If all the servers in $s \in I_{B \cup \{Q_i, Q_j\}}$ reply with $\tau_{inprogress_s(\omega')} = \tau_{r_1}(\rho_1)$ then there are three different values for Q_j to consider: (i) $Q_j = Q_i$, (ii) $Q_j \in B$, and (iii) $Q_j \notin B \cup Q_i$. Since $|B| \leq \frac{n}{2} - 3$ then in all three cases the predicate **PR** will hold for ρ_2 for at least tag $\tau_{inprogress_s(\omega')}$ and with a quorum set $|C| \leq \frac{n}{2} - 2$. Thus ρ_2 will either return a $\tau_{confirmed_*(\rho_2)} \geq \tau_{inprogress_s(\omega')}$, a tag $\tau_{inprogress_s(*)} > \tau_{inprogress_s(\omega')}$ or $\tau_{inprogress_s(\omega')}$. Hence $\tau_{r'}(\rho_2) \geq \tau_r(\rho_1)$ in this case as well.

Case 2: In this case ρ_1 is fast and returns in one communication round since he observed a $\tau_r(\rho_1) = \tau_{confirmed_*(\rho_1)}$ tag in an intersection $I_{B \cup \{Q_i\}}$ such that $|B| = n - 2$. Since we assume that $\mathbb{Q}$ is an n-wise quorum system then it follows that $I_{B \cup \{Q_i, Q_j\}} \neq \emptyset$ (since $|B \cup \{Q_i, Q_j\}| \leq n$), and hence ρ_2 will receive a $\tau_{confirmed_s(\rho_2)} \geq \tau_r(\rho_1)$ from at least a single server in $I_{B \cup \{Q_i, Q_j\}}$. Thus by Lemma B.15, ρ_2 returns $\tau_{r'}(\rho_2) \geq \tau_r(\rho_1)$ and that completes the proof. □

Theorem B.19 SFW *implements a near optimal MWMR atomic read/write register.*

Proof. It follows from Lemmas A.5, B.16, B.18, B.17 and B.12. □

C Write Optimality: Omitted Proofs

Lemma 5.3 Omitted Proof. The proof follows the fact that a server is not aware of a written value v unless: 1) it receives messages from the writer that propagates that value v, or 2) It receives messages from a process that already observed value v in the system. Moreover a server may infer the latest value at time t in any execution if: 1) receives messages from all the write operations invoked at time $t' < t$ (and thus contains all the values written), or 2) received a message that contained the value history at time $t' < t$ and received messages from the write operations invoked at time t'' s.t. $t' < t'' < t$ thereafter.

It is easy to see that a server may not be aware of the latest value even if all the write operations are not concurrent with each other. Assume to derive contradiction that a server may return the latest value to a read operation even if it does not receive messages from all the write operations. Let consider a server s at time t of an execution ξ. Assume that s received all the messages from every write operation invoked at time $t' < t$. Suppose w.l.o.g. that the latest value that the server s received was v. Since the write operations are totally ordered, then it follows that v is the latest written value in the system. Consider now an extension of the above execution ξ' by a write operation ω that writes value v_1. Assume that s does not receive messages from ω in ξ'. Thus s cannot distinguish ξ from ξ' and replies with a latest value v to any read operation. Since however the write operations are totally ordered then the latest value in the system is v'. Thus contradiction. □

Lemma 5.4 Omitted Proof. Let $\mathbb{Q}$ be some *n-wise* quorum system, for $2 \leq n < |\mathbb{Q}|$. We provide a series of execution constructions that depend on the intersection degree n. If $n = 2$ then ξ_0 is the execution that consists of a single $(n - 1 = 1)$ complete fast write operation $\omega(1)$ invoked by w_1 with $scnt_{w_1}(Q_1, \omega(1))$. If $n = 3$ then we extend ξ_0 by a complete fast write operation, $\omega(2)$, from w_2 with $scnt_{w_2}(Q_2, \omega(2))$, to obtain execution ξ_1.

In general we construct execution ξ_i if $\mathbb{Q}$ is an *n-wise* system with $n = i + 2$, by extending execution ξ_{i-1} with a complete fast write operation, $\omega(i + 1)$, from $w_{(i \mod 2)+1}$ with $scnt_{w_{(i \mod 2)+1}}(Q_{i+1}, \omega(i + 1))$. By this construction any execution ξ_i contains $i + 1$ (or $n - 1$) consecutive, quorum shifting fast write operations.

We proceed by induction on the intersection degree n, to show that extending any of the above executions with a read operation by the reader r preserves property S1. In other words the read operation is able to discern the latest write operation and return its value.

Induction base: We assume that $n = 2$ and hence pairwise intersection between the quorums of $\mathbb{Q}$. In this case we extend execution ξ_0 by a read operation ρ from r to obtain the following execution ξ_0':

a) a complete fast write operation ω_1 by w_1 with $scnt_{w_1}(Q_1, \omega_1)$, and

b) a complete read operation ρ by r with $scnt_r(Q_j, \rho)$.

It is easy to see that the read operation ρ, for any $Q_j \in \mathbb{Q}$, observes the value written by ω_1 in $Q_1 \cap Q_j$ ($\neq \emptyset$). Since ω_1 is the only write operation then ρ will return the value written by ω_1 and preserve the S1 property.

Inductive hypothesis: Assume that $n = k + 2$ and that extending execution construction ξ_k with a read operation ρ preserves property S1. It follows that ρ returns the value written by the last proceeding write operation which in ξ_k is $\omega(k+1)$ that $scnt_{w_{(k \mod 2)+1}}(Q_{k+1}, \omega(k+1))$.

Induction step: We now investigate the case where $\mathbb{Q}$ is a $(k+3)$-*wise* quorum system. We need to verify if execution ξ_{k+1} preserves property S1. Recall that ξ_{k+1} is constructed by extending ξ_k with a fast complete write operation $\omega(k+2)$. We further extend ξ_k by a read operation ρ by r to obtain ξ'_{k+1}. The last three operations of ξ'_k are the following:

a) a complete fast write operation $\omega(k+1)$ by $w_{(k \mod 2)+1}$ that $scnt_{w_{(k \mod 2)+1}}(Q_k + 1, \omega(k+1))$

b) a complete fast write operation $\omega(k+2)$ by $w_{(k+1 \mod 2)+1}$ that $scnt_{w_{(k+1 \mod 2)+1}}(Q_{k+2}, \omega(k+2))$, and

c) a complete read operation ρ by r that $scnt_r(Q_j, \rho)$.

By the inductive hypothesis we know that the execution fragment of ξ_k preserves property S1. Furthermore any $k+3$-*wise* quorum system is also a $k+2$-*wise* quorum system. So it follows that if ξ_k was extended by a read operation then that read operation would have returned, by induction hypothesis, $\omega(k+1)$. Thus in execution ξ'_{k+1} ρ may return either $\omega(k+1)$ or $\omega(k+2)$. If $\omega(k+1)$ is returned then it follows that ρ cannot distinguish ξ'_{k+1} from ξ'_k and hence does not observe $\omega(k+2)$ and violates property S1. Since $\mathbb{Q}$ is also a $k+2$-*wise* system then it must be true that under $\mathbb{Q}$, $\bigcap_{i=1}^{k+2} Q_i \neq \emptyset$, and hence the two writers w_1 and w_2 and the servers in $\bigcap_{i=1}^{k+2} Q_i \neq \emptyset$ can distinguish between ξ'_k and ξ'_{k+1} since those are the only servers that received messages from all the write operations. From the quorum construction however we know that $\mathbb{Q}$ has an intersection degree of $k+3$ and thus $\bigcap_{i=1}^{k+3} Q_i \neq \emptyset$. So for any quorum Q_j that replies to ρ it must hold that $\left(\bigcap_{i=1}^{k+2} Q_i\right) \cap Q_j \neq \emptyset$. Thus the read operation receive replies from the set of servers that distinguish the two executions and hence ρ also distinguishes ξ'_{k+1} from ξ'_k. So ρ returns $\omega(k+2)$ and preserves property S1. □