ANALYZING THE NUMBER OF SLOW READS FOR
SEMIFAST ATOMIC READ/WRITE REGISTER IMPLEMENTATIONS

Chryssis Georgiou  
Department of Computer Science  
University of Cyprus, Nicosia, Cyprus  
Email: chryssis@cs.ucy.ac.cy

Sotiros Kentros, Nicolas Nicolaou, and Alexander A. Shvartsman  
Department of Computer Science and Engineering  
University of Connecticut, Storrs, CT 06268, USA  
Email: {skentros, nicolas, aas}@engr.uconn.edu

ABSTRACT
Developing fast implementations of atomic read/write registers in the message passing model is among the fundamental problems in distributed computing. Typical implementations require two communication round trips for read and write operations. Dutta et al. [4] developed the first fast single writer, multiple reader (SWMR) atomic memory implementation, where all read and write operations complete in a single communication round trip. It was shown that fast implementations are possible only if the number of readers is constrained with respect to the number of register replicas and the number of replica failures. Addressing this constraint, Georgiou et al. [5] developed a solution for an arbitrary number of readers at the cost of allowing some reads to be slow, i.e., taking two round trips. They termed such implementations semifast.

Once some reads are allowed to be slow, it is interesting to quantify the number of occurrences of slow reads in executions of semifast implementations. This paper analyzes the implementation [5], yielding high probability bounds on the number of slow read operations per write operation. The analysis is performed for the settings with low and high contention of read and write operations. For scenarios with low contention it is shown that $O(\log R)$ slow read operations may suffice per write operation. For scenarios with high contention it is shown that if $\Omega(\log R)$ reads occur then the system may reach, with high probability, a state from which up to $R$ slow reads may be performed. These probabilistic results are further supported by algorithm simulations.

KEY WORD: atomic memory, message-passing, fault-tolerance, probabilistic analysis

1 Introduction

Implementing atomic (linearizable) read/write memory in the message passing asynchronous systems is a challenging task in distributed computing [1, 2, 3, 7]. Fault-tolerant distributed implementations of atomic objects use replication and allow processes to share information with precise consistency guarantees despite the presence of asynchrony and failures. Following the development of a single writer, multiple reader (SWMR) atomic implementation in [2], where read operations perform two communication round trips with the second communication round in essence performing a write, a folklore belief developed that “atomic reads must write.” However, the work by Dutta et al. [4] established that if the number of readers is appropriately constrained with respect to the number of replicas, then both read and write operations can be implemented using a single communication round. Such implementations are called fast. Seeking to relax the constraint on the number of readers, Georgiou et al. [5] introduced the notion of semifast implementations, where some read operations are allowed to be slow, i.e., involve two communication round trips. They provided an algorithm for an arbitrary number of readers, where the readers are grouped into virtual nodes, with each reader assigned to a unique virtual node.

To compare the communication efficiency of semifast implementations [5], where some reads are allowed to be slow, to the efficiency of fast implementations [4], where all reads are fast, it is interesting to quantify the number of slow reads that can occur in executions of semifast implementations. The goal of this work is to evaluate the algorithm in [5] in order to establish analytical and empirical bounds on the number of slow read operations per write operation under sensible environmental assumptions.

Background. The seminal work by Attiya et al. [2] introduced an algorithm that implements atomic SWMR read/write registers in the asynchronous message-passing model. The register is replicated at all processors and value-timestamp pairs are used to impose a partial order on read and write operations. To perform a write operation, the writer increments its local timestamp and sends a message with the value-timestamp pair to all processors. When a majority of processors reply, the write completes. The processor performing a read operation broadcasts a read request and awaits replies. When a majority replies with their value-timestamp pairs, the reader detects the highest timestamp and broadcasts the pair consisting of this timestamp and its associated value. The read completes when the reader receives responses from a majority. Although the value of the read is established after the first communication round, skipping the second round may lead to violations of atomicity when reads are concurrent with a write.

Dutta et al. [4] presented the first implementation of atomic SWMR registers where all read and write operations are fast, i.e., involving a single communication round. Their setting consists of a single writer, $R$ readers, and $S$
servers (the replica hosts), where any subset of readers, the
writer, and up to \( f \) servers may crash. To perform a write
operation, the writer sends messages to all the servers and
waits for \( S - f \) servers to reply. When those replies are
received the write operation completes. The read protocol
operates similarly to a write, by sending a read request to
all the servers and waiting for \( S - f \) server replies. Once
those replies are received, the reader returns, depending on
its logic, either the value associated with the highest times-
tamp detected or the value associated with the immediately
preceding timestamp. The authors also show that fast im-
plementations are possible only if the number of readers
is less than \( \frac{2}{3} - 2 \). Moreover they show that fast imple-
mentations are not possible in the MWMR setting, even in the
case of two writers, two readers, and a single server failure.

To relax the constraint on the number of readers in [4],
Georgiou et al. [5] explored an implementation for arbi-
trary number of readers at the expense of allowing some
operations to be slow. Their investigation led to a semi-
fast implementation of an atomic SWMR read/write regis-
ter that allowed arbitrary number of readers and provided
fast write operations, where a single complete slow (two
communication round) read may be required per write op-
eration. The algorithm guaranteed that every read operation
that returned the same value and succeeded or preceded a
slow read was fast. It did not provide any guarantees how-
ever on the “fastness” of read operations that are concurrent
with the slow read.

**Contributions.** Our goal is to analyze the performance
of semifast implementations of atomic SWMR read/write
registers with unbounded number of readers and quantify
the number of slow read operations per write operation.
Specifically, we deal with the performance of the algorithm
in [5], focusing on how the number of slow reads depends
on the number of readers \( R \). We assume that \( R \) can be
arbitrarily large and it is independent of \( S \), the number
of replica servers, and \( V \), the number of virtual nodes used by
the algorithm. Our contributions are as follows:

1. We analyze the behavior of the algorithm in settings
with low and high contention. Informally, contention
refers to the number of replica owners that receive
messages from a write operation, before any read op-
eration observes the value written by that write (larger
number means lower contention).

   (a) For scenarios with low contention we show that
   with high probability at most \( \beta SV \log R \) slow
   reads occur for each write operation, where \( \beta \) is
   a constant. Given that \( R \) can be arbitrarily large
   relative to \( S \) and \( V \), this bound can be expressed
   as \( O(\log R) \), when \( S \) and \( V \) are considered
   constant with respect to \( R \).

   (b) For scenarios with high contention we show that
   after \( \Omega(\log R) \) reads, the system can reach, with
   high probability, a state where up to \( R \) slow read
   operations may take place.

2. We compare our probabilistic analysis with simulation
results. Our empirical results confirm that our analy-
sis reasonably characterizes the behavior of the algo-

**Paper Organization.** In Section 2 we present our model
assumptions and definitions. Section 3 presents a brief
description of the implementation presented in [5]. We
present our probabilistic analysis in Section 4 and we com-
pare our analysis with simulation results in Section 5. We
conclude in Section 6.

**2 Definitions and Notation**

Our system model (as in [5]) is formulated in terms of three
distinct sets of processors: a distinguished writer processor
\( w \), a set of \( R \) readers with unique ids from the set
\( \mathcal{R} = \{r_1, \ldots, r_R\} \), and a set of \( S \) object replica servers
with unique ids from the set \( \mathcal{S} = \{s_1, \ldots, s_S\} \). Any subset
of readers, the writer, and up to \( f \) servers may crash. Pro-
cessor \( p \) is faulty in an execution if \( p \) crashes; otherwise \( p \)
is correct.

A virtual node is an abstract entity that consists of a
group of reader processors. Each virtual node has a unique
id from the set \( \mathcal{V} = \{\nu_1, \ldots, \nu_V\} \), where \( V < \frac{S}{f} - 2 \). A reader
\( r_i \) that is a member of a virtual node \( \nu_j \) maintains its
virtual id \( \nu(r_i) = \nu_j \). Processors that share the same virtual
identifier are called siblings. Note that \( R \) is not bounded in
terms of \( S \) or \( V \).

We consider interleaving execution semantics, where
an execution consists of an alternating sequence of states
and actions, where each action represents an atomic transi-
tion at a particular processor [6]. A read or write request ac-
tion is called invocation and a response action correspond-
ing to a previous invocation is called a response. An op-
eration is incomplete in an execution if its invocation does
not have a matching response; otherwise the operation is
complete. We assume that the requests of a client are well-
formed meaning that the client does not invoke an operation
before receiving a response from a previously invoked op-
eration. For an execution we say that an operation (read
or write) \( \pi_1 \) precedes another operation \( \pi_2 \), or \( \pi_2 \) succeeds
\( \pi_1 \), if the response step for \( \pi_1 \) precedes the invocation step
of \( \pi_2 \); this is denoted by \( \pi_1 \rightarrow \pi_2 \). Two operations are
concurrent if neither precedes the other.

A processor performs a communication round in the
following way: (a) it sends messages to a subset of proces-
sors, (b) every processor that receives such a message and
does not crash replies without delay and (c) when a pro-
cessor receives enough replies it completes or starts a new
communication round. If a processor invokes an operation
that completes at the end of a single communication round,
the operation is fast. If the operation requires a second
communication round, the operation is slow. A semifast
implementation of an atomic read/write register provides
fast writes and reads, with the exception that under cer-
tain conditions reads are allowed to be slow. The following
Definition 2.1 ([5]) An implementation I of an atomic object is semifast if the following are satisfied:
P1. Every write operation is fast.
P2. Any complete read operation performs one or two communication rounds.
P3. For any execution of I, if ρ1 is a two-round read operation, then any read operation ρ2 with \(\mathcal{R}(\rho_1) = \mathcal{R}(\rho_2)\), such that \(\rho_1 \rightarrow \rho_2\) or \(\rho_2 \rightarrow \rho_1\), must be fast.
P4. There exists an execution of I that contains at least one write operation \(\omega\) and at least one read operation \(\rho_1\) with \(\mathcal{R}(\rho_1) = \omega\), such that \(\rho_1\) is concurrent with \(\omega\) and all read operations \(\rho\) with \(\mathcal{R}(\rho) = \omega\) (including \(\rho_1\)) are fast.

P3 states that if any slow read operation returns a value written by a certain write, then any other read operation returning the same value and not concurrent with the slow read operation must be fast. P4 requires the existence of an execution where all operations are fast and at least one read operation is concurrent with some write operation. This rules out trivial solutions that are fast in the absence of read and write concurrency.

3 Semifast Algorithm SF

![Figure 1. Writer Pseudocode](image1)

![Figure 2. Reader Pseudocode](image2)

![Figure 3. Server Pseudocode](image3)

The maximum timestamp a server learns, from the second communication round of a read operation. The set seen is used to record the virtual ids of processors that inquire the server’s latest timestamp. Each message received by a server from a process \(p \in \{w\} \cup R\) includes a message type, a timestamp and a virtual id vid. If the message received contains a timestamp \(ts\), higher than the local
timestamp, then the server sets its timestamp to \( ts \), and sets \( seen = \{ vid \} \). Otherwise, the server just adds \( vid \) in \( seen \) set. Every server replies to any message with its timestamp, its \( seen \) set, and its \( postit \) variable.

**Read Protocol:** A reader invokes a read operation by sending messages to all servers, then awaiting \( S-f \) responses. Upon receipt of needed responses it determines the maximum timestamp \( maxTS \) and the maximum postit value \( maxPS \) from the received messages, and it computes the set \( maxTSmsg \) of the messages that contain the discovered \( maxTS \). Then the following key read predicate is used to decide on the return value:

\[
RP: \text{ if } \exists \alpha \in [1, V+1] \text{ and } \exists MS \subseteq maxTSmsg \\
\text{ s.t. } (|MS| \geq S-\alpha f) \land (|\cap_{m \in MS} m.seen| \geq \alpha)
\]

If the predicate is true or if \( maxPS = maxTS \) the reader returns \( maxTS \), and may perform one or two communication rounds (we further discuss below when a read will have to perform two communication rounds). Otherwise it returns \( maxTS - 1 \), in one communication round.

This predicate is derived from the observation that for any two read operations \( \rho_1 \) and \( \rho_2 \) that witness the same \( maxTS \) and compute \( maxTSmsg_1 \) and \( maxTSmsg_2 \) respectively, the sizes of the sets, \( |maxTSmsg_1| \) and \( |maxTSmsg_2| \) differ by at most \( f \). Assume the following example that will help to visualize the idea behind the predicate. Let \( \varphi \) be an execution fragment that contains a complete write operation \( \omega \) that propagates a value associated with \( maxTS \) to \( S-f \) servers. Let \( \rho_1 \) discover \( maxTS \) in \( S-2f \) server replies, missing \( f \) of the servers that replied to \( \omega \). Since \( \omega \rightarrow \rho_1 \) then \( \rho_1 \) has to return the value associated with \( maxTS \) to preserve atomicity. Assume now an execution fragment \( \varphi' \) that contains an incomplete write \( \omega \) that propagates the new value with \( maxTS \) to \( S-2f \) servers. Let extend \( \varphi' \) by a read \( \rho_2 \) from \( r_j \). If \( \rho_2 \) discovers \( maxTS \) in \( S-2f \) servers – by receiving replies from all the servers that received messages from \( \omega \) – then it cannot distinguish \( \varphi \) from \( \varphi' \) and thus has to return \( maxTS \) in \( \varphi' \) as well. Let \( \rho_2 \) be a read operation from \( r_j \) s.t. \( \rho_1 \rightarrow \rho_2 \). The read \( \rho_2 \) may discover \( maxTS \) in \( S-3f \) replies by missing \( f \) of the servers that replied to \( \omega \). Let \( r_i \) belong in the virtual node \( v_i \) and \( r_j \) belong in the virtual node \( v_j \). There are two cases to consider for \( v_i \) and \( v_j \): (a) either \( v_i \neq v_j \) (b) or \( v_i = v_j \) (\( r_i \) and \( r_j \) are siblings). Notice that every server adds the virtual node of a reader to its \( seen \) set before replying for a read operation. Thus all the \( S-2f \) servers that contained \( maxTS \) replied with a \( seen = \{ 0, v_i \} \) to \( r_i \) because they added the virtual node of the writer (0) and the virtual node of \( r_i \) before replying for \( \rho_1 \). With similar reasoning all the \( S-3f \) servers that replied for \( \rho_2 \) send a \( seen = \{ 0, v_j, v_i \} \) to \( r_j \). So if \( v_i \neq v_j \) then the predicate will hold with \( \alpha = 2 \) for \( r_i \) and with \( \alpha = 3 \) for \( r_j \). Thus \( r_j \) will also return \( maxTS \) preserving atomicity. If, however, \( v_i = v_j \) then the predicate will hold for \( r_j \) but will not hold for \( r_j \) and thus \( r_j \) will return an older value violating atomicity.

Hence to avoid such situation a read operation \( \rho \) is slow when it returns \( maxTS \) and either of the following cases hold:

**SL1.** if \( \rho \) observes \( |\cap_{m \in MS} m.seen| = \alpha \), or

**SL2.** if \( |\cap_{m \in MS} m.seen| < \alpha \) and \( maxPS = maxTS \) and less than \( f+1 \) messages contained \( maxPS \).

During the second communication round the reader sends \( inform \) messages to 3\( f+1 \) servers and waits for 2\( f+1 \) servers to reply. Once those replies are received, it returns \( maxTS \) and completes.

A slight modification needs to be applied to the algorithm to associate returned timestamps with values. To do this, the writer attaches two values to the timestamp in each write operation: (1) the current value to be written, and (2) the value written by the immediately preceding write operation (for the first write this is \( \perp \)). The reader receives the timestamp with its two attached values. If, as before, it decides to return \( maxTS \), then it returns the current value attached to \( maxTS \). If the reader decides to return \( maxTS - 1 \), then it returns the second value (corresponding to the preceding write).

**Discussion.** Looking at the conditions causing a read operation to be slow and the recording mechanism on the server side, we observe that \( |\cap_{m \in MS} m.seen| \leq |\{v_1, \ldots, v_V\} \cup \{w\}| \) and thus \( |\cap_{m \in MS} m.seen| \leq V + 1 \). We are interested in obtaining upper bounds for the worst case analysis of the algorithm and in Section 4 we compute the number of reads required, so that every server \( s \) receives at least one message from some reader of every virtual node, and maintains \( V \leq |s.seen| \leq V + 1 \), with high probability. This will help us specify (with high probability), the maximum number of slow read operations that may occur after a slow read that observed \( \alpha < V \) and before any subsequent read observes \( |\cap_{m \in MS} m.seen| \geq V + \alpha \) and thus is fast. When the slow read observes \( \alpha \in [V, V + 1] \), the maximum number of slow reads may reach \( R \), the number of readers participants in the system. These cases are explored in more detail in Section 4.3. Lastly, if all virtual nodes are recorded at the server side, then condition SL2 does not hold. Thus slow reads cannot be introduced because of SL2 and hence we omit examining that condition individually.

### 4 Probabilistic Bounds for the Number of Slow Reads in SF

For our analysis we assume that for any read operation \( \rho \) we have \( \Pr[\rho \text{ invoked by some } r \in v_i] = \frac{1}{c} \). That is, the read invocations may happen uniformly from the readers of any virtual node. We also assume that readers are uniformly distributed within groups. We say that event \( e \) happens with high probability (whp) if \( \Pr[e] = 1 - R^{-c} \), for \( R \) the number of readers and \( c > 1 \) a constant, and we say that event \( e \) happens with negligible probability if \( \Pr[e] = R^{-c} \).
We start by examining how the cardinality of set seen of a specific server is affected by the read operations and then we investigate the read bounds under executions with low and high contention. For the rest of the section let \( m_s(\pi) \) denote the message sent from process \( p \), that invoked operation \( \pi \), to server \( s \). Furthermore let \( ts_s(\pi) \) denote the timestamp that server \( s \) includes in the reply to process \( p \) for operation \( \pi \) and \( ts_o \), the timestamp propagated by the write operation \( \omega \). Finally let \( \sigma.ts_s \) and \( \sigma.seen_s \) be the local timestamp and the seen set of server \( s \) in state \( \sigma \) of an execution \( E \).

### 4.1 The set seen and fast read operations

We seek the number of read operations required, for a single server to record all virtual nodes in its seen set.

**Definition 4.1** Consider an execution of algorithm SF, and let \( M \) be the ordered set of read messages received by server \( s \) in the execution. A message \( m_s(\pi_1) \in M \) is ordered before a message \( m_s(\pi_2) \in M \) if server \( s \) receives \( m_s(\pi_1) \) before \( m_s(\pi_2) \). Let messages \( m_s(\rho), m_s(\rho') \) be the first and last messages in \( M \) respectively. We say that \( M \) is a set of consecutive read messages if \( M \) includes all messages received by \( s \) after \( m_s(\rho) \) and before \( m_s(\rho') \) are received by \( s \) and \( ts_s(\rho) = ts_s(\rho') \).

In other words, set \( M \) is consecutive when server \( s \) does not increase its timestamp and thus does not reset its seen set between messages \( m_s(\rho) \) and \( m_s(\rho') \). Now we can compute the number of consecutive reads required to contact server \( s \) so that every virtual node \( v_i \) is included in the seen set of \( s \).

**Lemma 4.1** In any execution of SF, there exists constant \( \beta \) s.t. if server \( s \) receives \( \beta V \log R \) consecutive read messages, then \( V = \{v_1, \ldots, v_V\} \subseteq \sigma.seen_s \), whp, where \( \sigma \) is the state of the system when server \( s \) receives the last of the \( \beta V \log R \) messages.

**Proof.** Recall that we assume that \( \Pr[\rho \text{ invoked by some } r \in v_i] = \frac{1}{V} \). Let \( k \) be the number of reads. From Chernoff Bounds [8] we have

\[
\Pr \left[ \text{# of reads from readers of group } v_i \leq (1 - \delta) \frac{k}{V} \right] \leq e^{-\delta^2 k}
\]

where \( 0 < \delta < 1 \). We compute \( k \), s.t. the probability in Equation (1) is negligible.

\[
e^{-\frac{\delta^2}{2}} = R^{-\gamma} \Rightarrow \frac{-\delta^2 \cdot k}{2 \cdot V} = -\gamma \cdot \log R \Rightarrow \]

\[
k = \frac{2 \cdot \gamma \cdot V \cdot \log R}{\delta^2}
\]

Let \( \delta = 0.5 \). From Equation (2) we have \( k = 8 \cdot \gamma \cdot V \cdot \log R \). We set \( \beta = 8 \cdot \gamma \) and we have that \( k = \beta \cdot V \cdot \log R \.

If \( \beta V \log R \) consecutive reads contact a server \( s \), at least \( \frac{\beta}{2} \log R \) reads will be performed by readers in group \( v_i \) whp, for any group \( v_i \) and thus \( v_i \in \sigma.seen_s \), since the seen set of \( s \) has not been reset. \( \square \)

Notice that the larger the constant \( \beta \) in the above lemma, the smaller the probability of having a server that did not receive a read message from a reader from every virtual node.

**Lemma 4.2** If in a state \( \sigma \) of some execution of SF, there exists set \( S' \subseteq S \) s.t. \( |S'| \geq 4f \) and \( \forall s \in S' \sigma.ts_s = ts_o \), and \( \sigma.seen_s = \{w\} \cup \mathcal{V} \), where \( ts_o \) the timestamp of write operation \( \omega \), then any read operation \( \rho \) with \( \mathcal{R}(\rho) = \omega \) invoked after \( \sigma \) is fast.

**Proof.** From the predicate \( RP \) of algorithm SF and the fact that \( V < \frac{S}{f} - 2 \) (thus \( V \leq \frac{S}{f} - 3 \)), it follows that if a reader observes \( |MS| \geq S - Vf \geq (V + 3)f - Vf \geq 3f \), then \( \alpha \leq V \). Assuming that \( \rho \) observes \( |MS| \geq 3f \) and all the servers with messages in \( MS \) replied with a seen = \( \{w\} \cup \mathcal{V} \) then the predicate of SF for \( \rho \) holds for:

\[
\bigcap_{m \in MS} m.seen = |\{w, v_1, \ldots, v_V\}| = V + 1 > \alpha.
\]

If a read operation \( \rho \) with \( \mathcal{R}(\rho) = \omega \) is invoked after \( \sigma \), then we have two cases for the maximum timestamp \( \max TS \) observed by \( \rho \): (a) \( \max TS = ts_o \) and (b) \( \max TS \geq ts_o + 1 \). For case (a), since up to \( f \) servers may fail, \( \rho \) observed an \( |MS| \geq 3f \) with \( |\bigcap_{m \in MS} m.seen| = |\{w, v_1, \ldots, v_V\}| = V + 1 \). Thus the predicate holds for \( \alpha \leq V \) and \( \rho \) is fast. For case (b) since \( \mathcal{R}(\rho) = \omega \) from algorithm SF, \( \rho \) is fast (see Section 3). \( \square \)

Note that if less than \( 4f \) servers contain seen = \( \{w\} \cup \mathcal{V} \), then a read operation \( \rho \) may observe \( |MS| = 3f \) or \( |MS| = 2f \). If \( \rho \) observes \( |MS| = 2f \) messages with seen = \( \{w\} \cup \mathcal{V} \), the predicate of SF for \( \rho \) holds for \( |\bigcap_{m \in MS} m.seen| = V + 1 = \alpha \) and \( \rho \) is slow.

From Lemma 4.2 it follows that the predicate \( RP \) of algorithm SF naturally yields two separate cases for investigation: (a) \( 4f \) servers or more contain the maximum timestamp and (b) less than \( 4f \) servers contain the maximum timestamp. In both cases we can use Lemma 4.1 to bound the number of read operations needed until seen = \( \{w\} \cup \mathcal{V} \) for all the servers with the maximum timestamp. To obtain the worst case scenario we assume that there are \( O(R) \) reads concurrent with the first slow read operation.

We now define formally the execution conditions that capture the idea behind the aforementioned cases. For an operation \( \pi \) let \( inv(\pi) \) and \( res(\pi) \) be its invocation and response events. We say that two operations \( \pi, \pi' \) are successive if \( \pi \) and \( \pi' \) are invoked by the same process \( p \) and \( p \) does not invoke any operation \( \pi'' \) between \( \text{res}(\pi) \) and \( \text{inv}(\pi') \). Let \( rcv_o(\pi) \) be the receipt event of a message sent for the operation \( \pi \). Notice that by SF, at least \( S - f \), \( rcv_o(\pi) \) events appear between an \( \text{inv}(\pi) \) and \( \text{res}(\pi) \) in
any execution. Since a message may be delayed, then a \texttt{recv}(\pi) event may succeed \texttt{res}(\pi). First we define the set of events that occur between two successive operations invoked by a process \( p \).

**Definition 4.2 (Idle Set)** For any execution \( E \) and for any pair of successive operations \( \pi, \pi' \) invoked by a process \( p \), we define \( \text{idle}(\pi, \pi') \) to be the set of all events that appear in \( E \) and succeed \( \text{res}(\pi) \) and precede \( \text{inv}(\pi') \).

Given the above definition we now define the contention conditions that affect our analysis. These conditions characterize cases (a) and (b).

**Definition 4.3 (Contention)** Let \( \rho, \rho' \) be any pair of successive read operations invoked by a reader \( r \). We say that an execution fragment \( \varphi \) has low contention if every set \( \text{idle}(\rho, \rho') \) contains an \( \text{inv}(\omega) \) event for some write operation \( \omega \), it also contains at least \( 4f \), \( \text{recv}_w(\omega) \) events. Otherwise we say that \( \varphi \) has high contention.

### 4.2 Slow reads under low contention

We consider the case of low contention where a set of at least \( 4f \) servers received messages from a write operation \( \omega \), before the first slow read operation \( \rho \), with \( \mathcal{R}(\rho) = \omega \). For an implementation to be semifast, any read operation \( \rho' \) that precedes or succeeds \( \rho \), with \( \mathcal{R}(\rho') = \mathcal{R}(\rho) = \omega \), is fast. We now bound the number of slow read operations.

**Theorem 4.3** In an execution \( E \) of SF, if before the invocation of the first slow read operation \( \rho \), with \( \mathcal{R}(\rho) = \omega \), for a write operation \( \omega \), there exists state \( \sigma \) and a set of servers \( S' \subseteq S \), s.t. \( |S'| \geq 4f \) and \( \forall s \in S', \sigma.s_t = t_{s_w} \) and \( w \in \sigma.\text{seen}_s \), then there exists constant \( \beta \), s.t. whp at most \( \beta \cdot S \cdot V \cdot \log R \) slow reads can occur that return the value and timestamp of \( \omega \).

**Proof.** For each \( s \in S' \), we examine two cases:

**Case 1:** After \( s \) receives a write message from \( \omega \), \( s \) receives a set \( M \) of consecutive read messages, s.t. \( |M| = \beta V \log R \) and \( t_{s_w}(\rho_{\text{last}}) = t_{s_w}, \) where \( m_s(\rho_{\text{last}}) \) is the last message in \( M \). From Lemma 4.1 any read \( \rho' \) with message \( m_s(\rho') \) receives \( \mathcal{R}(\rho') = \omega \) after \( m_s(\rho_{\text{last}}) \). If a read \( \rho' \) before \( t_{s_w} \), \( s \) will observe \( \text{seen} = \{w\} \cup V \) for \( s \) if \( t_{s_w}(\rho_{\text{last}}) = t_{s_w} \).

**Case 2:** After \( s \) receives a write message from \( \omega \), \( s \) receives message \( m_s(\pi) \) with \( m_s(\pi).t_s > t_{s_w}, \) before it receives \( \beta V \log R \) read messages. It follows that a write operation \( \omega' \), s.t. \( t_{s_w'} > t_{s_w} \) has been invoked. Any read \( \rho' \) that receives \( t_{s_w'}, \) will either return \( t_{s_w'}, 1 \) or \( t_{s_w'} \). From the construction of SF, if \( \rho' \) returns \( t_{s_w'}, 1 \), \( \rho' \) will be fast. Thus if a read \( \rho' \) contacts server \( s \) after \( m_s(\pi) \) and \( \mathcal{R}(\rho') = \omega \), then \( \rho' \) is fast and \( t_{s_w} = t_{s_w'} - 1 \).

From the above cases, we have a total of \( \beta \cdot |S'| \cdot V \cdot \log R \leq \beta \cdot S \cdot V \cdot \log R \) read messages for the servers in \( S' \). For any read operation \( \rho' \) that succeeds \( \text{res}(\pi) \), any read operation \( \rho' \) in \( S' \) is semifast. For any read \( \rho' \) invoked after \( \rho \), s.t. \( \rho' \neq \rho \), we have the following cases:

**Case (i):** \( \mathcal{R}(\rho') = \omega \) after the first slow read operation \( \rho \), with \( \mathcal{R}(\rho') = \omega \). As discussed in Lemma 4.2 \( \alpha \leq V \) and thus by the predicate of SF, \( \rho' \) will be fast and \( \mathcal{R}(\rho') = \omega \).

**Case (ii):** \( \mathcal{R}(\rho') \neq \omega \) after all \( f \) servers receive messages. From algorithm SF and the discussion in case 2, if \( \mathcal{R}(\rho') = \omega \), then \( \rho' \) is fast.

**Theorem 4.3** proves that under low contention, a write can be followed by at most \( O(\log R) \) slow reads whp (recall that we are interested in asymptotics with respect to \( R \)).

### 4.3 Slow reads under high contention

Here we deal with the high contention case, where a set of less than \( 4f \) servers receive messages from a write operation \( \omega \), before the first slow read operation \( \rho \), with \( \mathcal{R}(\rho) = \omega \). We examine the case where \( V = \frac{S}{2} - 3 \), which is the maximum number of virtual nodes allowed by algorithm SF. As we discussed in Section 4.1, in this case if a reader receives replies from only \( 2f \) updated servers and for these \( 2f \) servers \( \text{seen} = \{w\} \cup V \), we have a slow read, since \( S - (V + 1)f = S - \left( \frac{S}{2} - 2 \right) f = 2f \).

In contrast with the low contention case, we show that in the high contention case, the system relatively fast reaches a state where all reads that receive replies from less than \( 3f \) servers will be slow.

In the semifast implementation SF, after a slow read completes, any subsequent reads (for the same write) will be fast. Thus any reader can perform at most one slow read that returns the value and timestamp of \( \omega \). This gives a bound of at most \( R \) slow reads per write operation. We next prove that under high contention, \( \beta \cdot 4f \cdot V \cdot \log R \) reads can lead the system to a state where all reads concurrent with the first slow read can be slow if they receive replies from less than \( 3f \) updated servers.

**Theorem 4.4** In an execution \( E \) of SF, if before the invocation of the first slow read operation \( \rho \), with \( \mathcal{R}(\rho) = \omega \), there exists a set of reads messages, s.t. \( |M| \leq \beta \cdot 4f \cdot V \cdot \log R \) and any read operation \( \rho' \) invoked after all messages in \( M \) are received by servers in \( S \), will be slow if \( \max TS = t_{s_w} \) and \( |M| < 3f \).

**Proof.** For any server \( s \in S' \) if \( \beta V \cdot \log R \) consecutive read messages were received by \( s \) after \( \sigma \), where \( \beta \) is taken from Lemma 4.1, then whp the \( \text{seen} \) set of \( s \) becomes \( \{w\} \cup V \) after the last message is received.

If we consider such reads for all servers in \( S' \), we have a total of \( \beta |S'| \cdot V \cdot \log R < \beta \cdot 4f \cdot V \cdot \log R \) read messages. After these read messages, the system reaches a state \( \sigma' \) where \( \forall s \in S' \), \( s'.t_s = t_{s_w} \) and \( s'.\text{seen}_s = \{w\} \cup V \). As
discussed in Section 4.1, any read operation that contacts less than \(3f\) servers with the maximum timestamp, will be slow. This is possible, since up to \(f\) servers may fail. \(\square\)

From Theorem 4.4, observe that under high contention an execution relatively fast can reach a state (after \(\Omega(\log R)\) reads) that could lead to \(O(R)\) slow reads.

5 Simulation Results

This section reports performance results obtained by simulating algorithm SF using the NS-2 network simulator and a testbed similar to that presented in [5]. We compare the results of our probabilistic analysis with the simulation results, confirming that our analysis correlates with the simulated scenarios for the number of servers used in the simulation. The simulation designed and performed for this paper focuses on counting the average number of slow reads per write operation in settings of interest, whereas the simulation in [5] dealt with the overall percentage of slow reads throughout the entire execution in different settings.

In this work our setting involves 20 servers and a variable number of readers, ranging between 10 and 80. Algorithm SF uses a number of virtual nodes \(V\) such that \(V < \frac{2}{3} - f\). Thus the number of crashes \(f\) cannot be greater than 5. We introduce up to 5 server crashes that occur at arbitrary times during the execution. We model nodes that are connected with duplex links at 1 MB bandwidth, a latency of 10 ms, and a DropTail queue. To model asynchrony the processors send messages after a random delay between 0 and 0.3 seconds. The contention level is modeled by specifying positive time intervals \(rInt\) and \(wInt\) between any two successive read and write operations respectively. The intervals are the following:

(a) \(rInt = 2.3s\) and \(wInt = 4.3s\)
(b) \(rInt = 4.3s\) and \(wInt = 4.3s\)
(c) \(rInt = 6.3s\) and \(wInt = 4.3s\)

Figure 4 presents the results obtained by using the intervals of the scenarios (a) and (b) in the simulation. The plots illustrate the average slow reads per write as a function of the number of readers and number of server failures. Examining the plots we can see that under low contention (i.e., similar to our best case analysis) the number of slow reads per write for every execution scenario is bounded by \(O(\log R)\): in no run the number of slow reads exceeds \(\log(80) = 6.3\). In scenarios with high contention (i.e., similar to the worst case analysis) the number of slow reads exceeds \(\log R\) but is always bounded under \(R\). In the case of scenario (c) every write operation is completed before the invocation of a read operation. Thus the simulation exhibited fast behavior and no read operation was slow (thus no plot is given). This is a special case of low contention that trivially conforms to our analysis.

6 Conclusions

We analyzed high probability bounds on the number of slow read operations for the semifast implementation of atomic registers presented in [5]. We first examined a scenario with low read/write operation contention and we showed that \(O(\log R)\) slow read operations may take place, whp, between two consecutive write operations. Then we studied a high contention scenario and we showed that after \(\Omega(\log R)\) reads, the system may reach whp a state where up to \(R\) slow read operations may take place. Our analysis is based on a single probabilistic assumption, namely that reads are performed uniformly at random from processors in all virtual groups. The probabilistic bounds are supported and illustrated by simulation results.

Acknowledgments. This work is supported in part by the NSF Grants 0311368 and 0702670. The work of the first author is supported in part by research funds from the University of Cyprus. The work of the second author is supported in part by the State Scholarships Foundation of Greece.
References


