Brief Announcement: Fault-Tolerant SemiFast Implementations of Atomic Read/Write Registers

Chryssis Georgiou[1], Nicolas C. Nicolaou[2], and Alexander A. Shvartsman[2,3]

[1] Dept. of Computer Science, University of Cyprus, CY-1678 Nicosia, Cyprus
[2] Dept. of Computer Science, University of Connecticut, USA
[3] Computer Science and Artificial Intelligence Laboratory, MIT, USA

Problem and Motivation. Atomic (linearizable) read/write memory is among the fundamental abstractions in distributed computing. Fault-tolerant implementations of atomic objects in message-passing systems allow processes to share information with precise consistency guarantees in the presence of asynchrony and failures. A seminal implementation of atomic memory of Attiya *et al.* [1] gives a single-writer, multiple reader (SWMR) solution where each data object is replicated at n message-passing nodes. Following this development, a folklore belief developed that in messaging-passing atomic memory implementations "atomic reads must write". However, recent work by Dutta *et al.* [2] established that if the number of readers is appropriately constrained with respect to the number of replicas, then single communication round implementations of reads are possible. Such an implementation given in [2] is called *fast*. Furthermore it was shown that any implementation with a larger set of readers cannot have only the single round-trip reads. Thus when the number of readers can be large, it is interesting to consider *semifast* implementations where the writes involve a single communication round and where the reads may involve one or two rounds with the goal of having as many as possible single round reads.

Our Contributions. Our goal is to develop atomic memory algorithms where a large number of read and write operations are fast. In particular, we want to remove constraints on the number of readers while preserving atomicity. We say that an atomic SWMR implementation is *semifast* if write operations take a single communication round and where read operations take one or two rounds. We show that one can obtain semifast implementations with unbounded number of readers, where in many cases reads take a single round. Our approach is based on forming groups of processes where each group is given a unique virtual identifier. The algorithm is patterned after the general scheme of the algorithm in [2]. We show that for each write operation at most one complete read operation returning the written value may need to perform a second communication round. Furthermore, our implementation enables non-trivial executions where both reads and writes are fast. We also provide simulation results for our algorithm, and we consider semifast implementations for multiple writers.

Semifast Implementations and Virtual Nodes. We consider the single writer, multiple reader (SWMR) model, where a distinguished process w is the writer, the set of R readers are processes with unique identifiers from the set $\mathcal{R} = \{r_1, \ldots, r_R\}$, and where the object replicas are maintained by the set of S servers with unique identifiers from the set $\mathcal{S} = \{s_1, \ldots, s_S\}$ such that at most t servers can crash.

To accommodate arbitrarily many readers, we introduce the notion of *virtual identifiers*. We allow multiple readers to share the same virtual identifier, thus forming groups of nodes that we call *virtual nodes*. More formally each virtual node has a unique identifier from a set $\mathcal{V} = \{\nu_1, \ldots, \nu_V\}$, and each reader r_i that is a member of a virtual node ν_j maintains its own identifier r_i and its virtual identifier $\nu(r_i) = \nu_j$; we identify such process by the pair $\langle r_i, \nu_j \rangle$. We assume that some external service is used to create virtual nodes by assigning virtual identifiers to reader processes. For a read operation, the determination of the proper return value is based on the cardinality of a set maintained by the servers, known as *seen* set, which contains virtual node identifiers and probably the writer identifier. Thus we use virtual nodes to set the boundary limits of the *seen* set even though arbitrarily many readers may use the service. To ensure the correctness of our algorithm we restrict the cardinality of the seen set to be less than $\frac{S}{t} - 1$ and hence the number of virtual nodes $|\mathcal{V}|$ to be less than $\frac{S}{t} - 2$.

A semifast atomic implementation, as suggested in [2], is an implementation that either has all reads that are fast *or* all writes that are fast. We formalize the definition of *semifast* implementations that requires all writes to be fast and that specifies which atomic reads are required to perform a second communication round. In this brief announcement we present an informal version of our definition: here, for each write operation, only *one complete* read operation is allowed to perform two communication rounds. In more detail a SWMR implementation I is *semifast* if the following properties are satisfied: (1) All writes are *fast*, (2) all complete read operations perform *one* or *two* communication rounds, (3) if a read operation ρ_1 performs two communication rounds, then all read operations that precede or succeed ρ_1 and return the same value as ρ_1 are fast, and (4) there exists some execution of I which contains only fast read and write operations.

Implementation SF. We now overview a semifast implementation, called SF, that supports one writer and arbitrarily many readers. We use timestamps to impose a partial order on the read and write operations, as in [1]. Of interest is the way in which timestamps are associated with the values.

To perform a write operation, the writer increases the timestamp and sends the new value to $S - t$ servers. The timestamps impose a natural order on the writes since there is only one writer.

The server processes maintain object replicas and do not invoke any read or write operations. To implement the fast operation behavior, the servers use a bookkeeping mechanism to record all the processes to whom they sent their latest timestamp. Therefore when a server s_i receives a message $\langle msgType, ts, *, vid \rangle$ from a non-server process p_j, it updates its local timestamp ts_ℓ to be equal to ts, if $ts > ts_\ell$, and it initializes the recording set called $seen$, to $\{vid\}$. Otherwise, if $ts \leq ts_\ell$, s_i sets its *seen* set to be equal to $seen \cup \{vid\}$ declaring that p_j inquired s_i's local timestamp. When a reader performs a second communication round, then the server s_i updates its postit value ps if $ts \geq ps$. This declares that the timestamp ts is about to be returned by some reader.

When a reader process invokes a read operation it sends messages to all servers and waits for $S - t$ responses. It determines the maximum timestamp $maxTS = ts'$ and the maximum postit $maxPS = ps'$ value contained among the received messages, and it computes the set of the messages that contain the discovered $maxTS$ ($maxTSmsg$).

Then the following key predicate is used to decide on the return value:
$\exists \alpha \in [1, V+1]\ \exists MS \subseteq maxTSmsg\ s.t.\ |MS| \geq S - \alpha t \wedge |\cap_{m \in MS}\ m.seen| \geq \alpha$

The above predicate is derived from the observation that for any two read operations ρ_1 and ρ_2 that witness the same $maxTS$ and computed $maxTSmsg_1$ and $maxTSmsg_2$ respectively, the difference $||maxTSmsg_1| - |maxTSmsg_2||$ is less than or equal to t. If the predicate is true or if $maxPS = maxTS$ the reader returns $maxTS$ otherwise it returns $maxTS - 1$. If a reader observes that $|\cap_{m \in MS}\ m.seen| = \alpha$ or less than $2t+1$ messages containing $maxPS$ are discovered in the system, then it performs a second communication round before returning $maxTS$.

To associate the timestamps with the values we maintain a triple $\langle ts, v_{ts}, v_{ts-1}\rangle$, where ts is the current timestamp, v_{ts} the value written with this timestamp, and v_{ts-1} the value written with the previous timestamp. We prove the correctness (atomicity) of the new implementation. Note that SF is not a straightforward extension of [2]. The introduction of virtual nodes raises new challenges such as ensuring consistency within groups so that atomicity is not violated by processes sharing the same virtual identifier.
Impossibility and MWMR model. We consider two families of algorithms, one that does not use reader grouping mechanisms and the other that assumes grouping mechanisms such as our algorithm. For both families we show that there is no semifast atomic implementation when $\frac{S}{t} - 2$ or more virtual identifiers (virtual nodes) exist in the system. The idea behind the proof is to show by contradiction that if $V \geq \frac{S}{t} - 2$, then the fast behavior of the system violates atomicity, and as a result property (4) of the semifast definition cannot hold. Additionally it is shown that any semifast algorithm must inform no less than $3t+1$ server processes during a second communication round — otherwise either property (3) of the semifast definition does not hold, or atomicity is violated.

Examining the applicability of the result in the multiple writer multiple reader (MWMR) model, we show that there does not exist semifast atomic implementations even in the case of 2 writers, 2 readers and $t = 1$. Our proof assumes that the read operations are allowed to perform one or more communication rounds.
Simulations. We simulated our SWMR implementation using the NS2 network simulator and we obtained preliminary results demonstrating that only a small fraction of read operations need to perform a second communication round. Specifically, under reasonable execution conditions in our simulations no more than 10% of the read operations required a second round. Our simulations assume that the readers are uniformly distributed among the virtual nodes. Furthermore we assume both stochastic and static environments and several plausible frequencies on the invocation of read and write operations.
Acknowledgment: We thank Rachid Guerraoui and Peter Musial for their helpful comments.

References

1. H. Attiya, A. Bar-Noy, and D. Dolev. Sharing memory robustly in message passing systems. *J. of the ACM*, 42(1):124–142, 1996.
2. P. Dutta, R. Guerraoui, R. R. Levy, and A. Chakraborty. How fast can a distributed atomic read be? In *Proceedings of the twenty-third annual ACM symposium on Principles of distributed computing*, pages 236–245. ACM Press, 2004.