Brief Announcement: Oh-RAM! One and a Half Round Read/Write Atomic Memory *

Theophanis Hadjistasi † Nicolas Nicolaou ‡ Alexander A. Schwarzmann †

ABSTRACT

Emulating atomic read/write shared objects in a message-passing system is a fundamental problem in distributed computing. Considering that network communication is the most expensive resource, efficiency is measured first of all in terms of the communication needed to implement read and write operations. It is well known that two communication round-trip phases involving in total four message exchanges are sufficient to implemented atomic operations. In this work we present a comprehensive treatment of the question of when and how it is possible to implement atomic memory where read and write operations complete in three message exchanges, i.e., we aim for *One and half Round Atomic Memory*, hence the name Oh-RAM! We present algorithms that allow operations to complete in three communication exchanges without imposing any constraints on the number of readers and writers. We present an implementation for the single-writer/multiple-reader (SWMR) setting, where reads complete in *three* communication exchanges and writes complete in *two* exchanges. Then we pose the question of whether it is possible to implement multiple-writer/multiple-reader (MWMR) memory where operations complete in at most three communication exchanges. We answer this question in the negative by showing that an atomic memory implementation is impossible if both read and write operations take *three* communication exchanges, even when assuming two writers, two readers, and a single replica server failure. Motivated by this impossibility result, we provide a MWMR atomic memory implementation where reads involve *three* and writes involve *four* communication exchanges. In light of our impossibility result these algorithms are optimal in terms of the number of communication exchanges.

†University of Connecticut, Storrs CT, USA. Email: theophanis.hadjistasi@uconn.edu, aas@engr.uconn.edu
‡IMDEA Networks Institute, Madrid, Spain. Email: nicolas.nicolaou@imdea.org

*The work of the second author is fully supported by FP7-PEOPLE-2013-IEF grant ATOMICDFS No:629088.

PODC'16, July 25–28, 2016, Chicago, IL, USA.

ACM ISBN 978-1-4503-3964-3/16/07.
DOI: http://dx.doi.org/10.1145/2933057.2933073

1. INTRODUCTION

For more than two decades a series of solutions sought to provide efficient implementations of the shared memory abstraction for distributed settings where crash-prone asynchronous processors communicate by exchanging messages. To cope with processor failures, distributed object implementations use *redundancy* by replicating the object at multiple network locations (replica servers). Replication introduces the problem of consistency due to the fact that read and write operations may access different object replicas, some of which may contain obsolete object values. Atomicity [7] (or linearizability [6]) is the most intuitive consistency semantic for read/write objects as it provides the illusion of a single-copy object that serializes all accesses such that each read operation returns the value of the latest preceding write operation.

The seminal work of Attiya, Bar-Noy, and Dolev [1] provided the first algorithm, colloquially referred to as ABD, that implements single-writer/multiple-reader (SWMR) atomic objects. The ordering of operations is accomplished with the help of logical *timestamps* associated with each value. Here each operation is guaranteed to terminate as long as some majority of replica servers do not crash. Each write operation takes one communication round-trip phase, or round, involving *two* message exchanges and each read operation takes two rounds involving in total *four* message exchanges. Subsequently, [8] showed how to implement multi-writer/multiple-reader (MWMR) atomic memory where both read and write operations involve two communication round trips involving in total four message exchanges.

The work by Dutta et al. [2] introduced a SWMR implementation where both reads and writes involve a single round consisting of *two* communication exchanges. It was shown that this is possible only when the number of readers r is bounded with respect to the number of servers s and the server failures f, specifically by $r < \frac{s}{f} - 2$, and when there is only a single writer in the system. An interesting observation made in [2] is that atomic memory may be implemented (using a max/min technique) so that each read and write operation complete in *three* communication exchanges. The authors however did not elaborate further on the inherent limitations that such a technique may impose on the distributed system.

Subsequent works, e.g., [4], focused in relaxing the bound on the number of readers and writers in the service by proposing hybrid approaches where some operations complete in *one* and others in *two* rounds. In addition, [3] provides tight bounds on the number of rounds that *read* and *write* operations require in the MWMR model.

Contributions. We address the gap between one-round and two-round implementations by considering implementations that take "one and a half rounds," i.e., where operations complete in *three* message exchanges, and that do not impose constraints on the number of readers. Table 1 summarizes our complexity results. Additional details follow.

Algorithm	Read Ex.	Write Ex.	Read Com.	Write Com.
ABD (SWMR) [1]	4	2	$4\|\mathcal{S}\|$	$2\|\mathcal{S}\|$
Oh-SAM	3	2	$\|\mathcal{S}\|^2 + 2\|\mathcal{S}\|$	$2\|\mathcal{S}\|$
ABD (MWMR) [1]	4	4	$4\|\mathcal{S}\|$	$4\|\mathcal{S}\|$
Oh-MAM	3	4	$\|\mathcal{S}\|^2 + 2\|\mathcal{S}\|$	$4\|\mathcal{S}\|$

Table 1: Summary of complexity results.

(1) We present a new SWMR algorithm for atomic objects in the asynchronous message-passing model with processor crashes. The write operation takes *two* communication exchanges and it is similar to the write operation of ABD. Read operations take *three* communication exchanges. A key idea of the algorithm is that the reader returns the value that is associated with the *minimum* timestamp (cf. the observation in [2]). The read operations in this algorithm are optimal in terms of communication exchanges.

(2) A major part of our contribution is to show the impossibility of MWMR implementations where both write and read operations take *three* communication exchanges. We show that atomicity is violated even in a system that consists of two readers, two writers, and is subject to a single server failure.

(3) Motivated by the impossibility result, we revise the SWMR algorithm to yield a MWMR algorithm. The existence of multiple writers complicate the write operations, and in the new algorithm writes take *four* communication exchanges (cf. [8]). Read operations complete again in *three* communication exchanges.

Both of our algorithms are *optimal* in terms of communication exchanges for the settings without constraints on the number of participants. The algorithms improve latency in a trade-off for communication complexity.

2. SYSTEM MODEL AND EFFICIENCY

The system consists of a collection of failure-prone, asynchronous processes with unique identifiers from a totally-ordered set $\mathcal{I}$. The set $\mathcal{I}$ consists of three disjoint sets: set $\mathcal{W}$ of writer identifiers, set $\mathcal{R}$ of reader identifiers, and a set $\mathcal{S}$ of replica servers identifiers with each *server* process maintaining a copy of the object. Processes communicate by exchanging messages via asynchronous point-to-point reliable channels; messages may be reordered in transit. For convenience we use *broadcast* as a shorthand for sending point-to-point messages to multiple destinations. Any subset of writers and readers, and any minority of the servers $\mathcal{S}$ may crash in any execution.

Efficiency of implementations is assessed in terms of (i) *message complexity*, i.e. the worst-case number of messages exchanged during an operation, and (ii) *operation latency*, that is determined by the *computation time* and the *communication delays*. Computation time accounts the computation steps that the algorithm performs in each read or write operation.

Communication delays are measured in terms of *communication exchanges*. The protocol implementing each operation involves a sequence of sends (or broadcasts) of typed messages and the corresponding receives. *Communication exchange* within an execution of an operation is the collection of sends and receives for a specific message type within the protocol. Traditional implementations, such as ABD, are structured in terms of *rounds*, where each round consists of two message exchanges; (i) a broadcast initiated by the process executing an operation; and (ii) a convergecast that consists of responses to the initiator.

3. SWMR ALGORITHM Oh-SAM

We first present the SWMR algorithm Oh-SAM: *One and a half Round Single writer Atomic Memory*. The write operation takes *two* communication exchanges (similarly to ABD). Read operations take *three* communication exchanges: (1) the reader sends message to servers, (2) each server that receives the request, *relays* the request to all servers, and (3) once a server receives the relay for a particular read from a majority of servers, it replies to the reader. The read completes once it collects a majority of these replies. A key idea of the algorithm is that the reader returns the value associated with the *minimum* timestamp.

Messages. There are five types of messages. The *readRequest* and *writeRequest* messages define a request from a process p to servers either for reading or writing respectively. The *readAck* and *writeAck* are messages sent from a server to a process p as an acknowledgement for a read or a write operation respectively. Servers relay a read request to all servers using *readRelay* messages.

Writer Protocol. Writer w increments its timestamp and broadcasts a *writeRequest* message to all servers $s \in \mathcal{S}$. It terminates when $|\mathcal{S}|/2+1$ *writeAck* messages are collected.

Read Protocol. Reader r creates a *readRequest* message and broadcasts it to all the servers. Once "fresh" messages are collected from a majority, then reader returns the value associated with the *minimum* timestamp among the replies.

Server Protocol. (1) Upon receiving *readRequest* message the server uses the information enclosed in the received message along with its local information to create a *readRelay* message which it broadcasts to all servers $s \in \mathcal{S}$.

(2) Upon receiving *readRelay* message server s compares the incoming with its local information and updates accordingly. Next, server s checks if the received *readRelay* message declares a new read operation and updates its local information. Finally, when the server collects *readRelay* messages from a majority of servers, then it creates a *readAck* message which it sends to the reader r.

(3) Upon receiving *writeRequest* message server s compares the incoming with its local information and updates accordingly. Finally, the server sends an acknowledgement message to the requesting writer w.

Correctness. We prove correctness of algorithm Oh-SAM by showing that *liveness* is satisfied with respect to our failure model, and *atomicity* is satisfied with respect to the order imposed by the *timestamps* used by each operation.

THEOREM 1. *Algorithm* Oh-SAM *implements an atomic SWMR read/write register.*

Complexity. The analysis of performance of Oh-SAM is straightforward and the results are given in Table 1.

4. IMPOSSIBILITY RESULT

Next we examine if it is possible to implement MWMR atomic read/write objects in an asynchronous, message-passing system with crash-prone processors where read and write operations take *three* communication exchanges. We

consider algorithms that implement a write operation invoked by process p according to the following three-phase scheme: (1) the invoker p sends a message to a set of servers; (2) each server that receives the message from p sends a certain relay message to a set of servers; and (3) once a server receives "enough" relay messages it replies to p. Each phase involves a communication exchange.

We briefly explain why it is reasonable to use such a three-phase scheme. First of all, the servers cannot know about a write operation unless writer contacts them, thus it must be the writer who initiates phase (1). Moreover, since asynchrony makes it impossible to distinguish slow servers from crashed servers, the writer must include all servers in this phase. In phase (3) it must be the servers who inform the writer about the status/completion of the write operation. Otherwise, either the third phase is unnecessary for the writer, or the writer will wait indefinitely. From the above reasoning, phase (2) must be the transitional phase for the servers to move from phase (1) to phase (3). Hence, phase (2) must facilitate the dissemination of the information regarding any write operation to the rest of the servers. Given this communication scheme we obtain the following result.

THEOREM 2. *It is not possible to obtain an atomic read/write register implementation, where all operations perform 3 communication exchanges, when* $|\mathcal{W}| = |\mathcal{R}| = 2$, $|\mathcal{S}| \geq 3$ *and* $f = 1$.

5. MWMR ALGORITHM Oh-MAM

Motivated by the impossibility result, we sought a solution that involves three or four communications exchanges per operation. We now present our MWMR algorithm Oh-MAM: *One and a half Round Multiple writer Atomic Memory.* Compared with the SWMR setting, we need to impose an ordering on the values that are concurrently written by multiple writers. Since writers may have the same local timestamp, we differentiate them by associating each value with a *tag* consisting of a pair of a timestamp *ts* and the *id* of the writer. We use lexicographic comparison to order tags (cf. [8]). The read protocol is identical to the SWMR setting, thus in Algorithm Oh-MAM we present only the protocols for the writer and server processes.

Messages. In this algorithm, two additional types of messages are taking place. A *discover* message defines a request from a process p to servers in order to discover the latest tag. Servers response is enclosed in a *discoverAck* message.

Writer Protocol. This protocol is similar to ABD (as given in [8]). Writer w broadcasts a *discover* message to all servers. It then waits to collect $|\mathcal{S}|/2 + 1$ *discoverAck* messages. Once the *discoverAck* messages are collected, writer w determines the maximum timestamp from the tags, increases it, creates a new *writeRequest* message and broadcasts it to all servers. It then waits for $|\mathcal{S}|/2 + 1$ *writeAck* messages to terminate.

Server Protocol. Servers react to messages from the readers exactly as in algorithm Oh-SAM. Here we describe server actions for *discover* and *writeRequest* messages.

(1) Upon receiving *discover* message server s attaches its local information in a new *discoverAck* message that it sends to writer w_i.

(2) Upon receiving *writeRequest* message the server compares the received with its local information and updates accordingly. Server acknowledges this operation to writer w_i by sending a *writeAck* message.

Correctness. We prove correctness of algorithm Oh-MAM by showing that *liveness* is satisfied with respect to our failure model, and *atomicity* is satisfied with respect to the lexicographical order imposed by the *tags* used by each operation.

THEOREM 3. *Algorithm* Oh-MAM *implements an atomic MWMR read/write register.*

Complexity. The performance of Oh-MAM appears in Table 1, in comparison to Oh-SAM and ABD (MWMR) [1].

6. CONCLUSIONS

In this work, we focused on the problem of emulating atomic read/write shared objects in message-passing setting using *three* communication exchanges, i.e., the equivalent of one-and-a-half traditional rounds. We presented an algorithm for the SWMR setting that allows each read operation to complete in *three* and each write operation in *two* communication exchanges. We showed that it is impossible to implement an MWMR atomic object when both read and write operations complete in *three* communication exchanges. Motivated by this, we presented the first algorithm for the MWMR setting that allows each read operation to complete in *three* and each write operation in *four* communication exchanges. In the full paper [5] we rigorously reasoned about the correctness of our algorithms. We note that the algorithms do not impose any constrains on the number of readers (SWMR and MWMR) or the writers (MWMR) participating in the service. Both algorithms are optimal in terms of communication exchanges when no bounds are imposed on participation.

7. REFERENCES

[1] ATTIYA, H., BAR-NOY, A., AND DOLEV, D. Sharing memory robustly in message passing systems. *Journal of the ACM 42(1)* (1996), 124–142.

[2] DUTTA, P., GUERRAOUI, R., LEVY, R. R., AND CHAKRABORTY, A. How fast can a distributed atomic read be? In *Proceedings of the 23rd ACM symposium on Principles of Distributed Computing (PODC)* (2004), pp. 236–245.

[3] ENGLERT, B., GEORGIOU, C., MUSIAL, P. M., NICOLAOU, N., AND SHVARTSMAN, A. A. On the efficiency of atomic multi-reader, multi-writer distributed memory. In *Proceedings 13th International Conference On Principle Of DIstributed Systems (OPODIS 09)* (2009), pp. 240–254.

[4] GEORGIOU, C., NICOLAOU, N. C., AND SHVARTSMAN, A. A. Fault-tolerant semifast implementations of atomic read/write registers. *Journal of Parallel and Distributed Computing 69*, 1 (2009), 62–79.

[5] HADJISTASI, T., NICOLAOU, N., AND SCHWARZMANN, A. A. Oh-ram! one and a half round read/write atomic memory. www.arxiv.com, 2016.

[6] HERLIHY, M. P., AND WING, J. M. Linearizability: a correctness condition for concurrent objects. *ACM Transactions on Programming Languages and Systems (TOPLAS) 12*, 3 (1990), 463–492.

[7] LAMPORT, L. How to make a multiprocessor computer that correctly executes multiprocess program. *IEEE Trans. Comput. 28*, 9 (1979), 690–691.

[8] LYNCH, N. A., AND SHVARTSMAN, A. A. Robust emulation of shared memory using dynamic quorum-acknowledged broadcasts. In *Proceedings of Symposium on Fault-Tolerant Computing* (1997), pp. 272–281.