On the Practicality of Atomic MWMR Register Implementations

Nicolas Nicolaou
University of Cyprus
nicolasn@cs.ucy.ac.cy

Chryssis Georgiou
University of Cyprus
chryssis@cs.ucy.ac.cy

***Abstract*—In this work we conduct an experimental performance evaluation of four MWMR atomic register implementations: SFW from [8], APRX-SFW and CWFR from [11], and SIMPLE (the generalization of [5] in the MWMR environment). We implement the algorithms on NS2, a single processor simulator, and on PlanetLab, a planetary-scale real-time network platform. Due to its simplistic nature, SIMPLE requires two communication round-trips per read or write operation, but almost no local computation. The rest of the algorithms are (to this writing) the only to allow *single* round read and write operations but require *non-trivial* computational demands. We compare these algorithms with SIMPLE and amongst each other to study the trade-offs between communication delay and local computation. Our results shed new light on the practicality of atomic MWMR register implementations.**

***Keywords*-distributed storage; atomic registers; computation vs communication; experimental performance evaluation;**

I. INTRODUCTION

Motivation and prior work: Frequent hardware failures increased the popularity of reliable distributed storage systems as a medium for data availability and survivability; such systems are becoming increasingly important in data centers and other enterprise settings [3]. Traditional approaches achieve reliability by using a centralized controller to handle the replication of data over a redundant array of independent disks (RAID). However, the centralized controller, which resides on a single location and is usually connected to the network via a single interface, constitutes a single point of failure and compromises data accessibility [6].

A distributed storage system overcomes this problem by replicating the data in geographically dispersed nodes, and ensuring data availability even in cases of complete site disasters. Distribution of data introduces, however, the problem of preserving data consistency between the replicas. Atomicity is the strongest consistency guarantee and provides the illusion that operations on the distributed storage are invoked sequentially, even though they can be invoked concurrently. The challenge of providing atomic consistency becomes even greater in the presence of asynchrony and failures. In order to address this challenge and provably identify the trade-offs between consistency, reliability and efficiency, researchers over the last two decades have focused on the simplest form of consistent distributed storage: *an atomic read/write register*; for a recent survey see [4]. Atomic read/write register implementations can be used directly to build distributed file systems (e.g., [9]) and also as building blocks for more complex distributed storage systems; for examples of storage systems built on register implementations see [10], [16], [15].

This work is supported by the Cyprus Research Promotion Foundation's grant ΠΕΝΕΚ/0609/31 and the European Regional Development Fund.

In this work we consider Multiple-Writer, Multiple-Reader (MWMR) atomic registers over an asynchronous, crash-prone, message-passing setting. In such settings the register is replicated among a set of replica hosts (or servers) to provide fault-tolerance and availability. Read and write operations are implemented as communication protocols that ensure atomic consistency.

The efficiency of register implementations is normally measured in terms of the latency of read and write operations. Two factors affect operation latency: (a) computation, and (b) communication delays. An operation communicates with servers to read or write the register value. This involves at least a single communication round-trip, or *round*, i.e., messages from the invoking process to some servers and then the replies from these servers to the invoking process. Previous works focused on reducing the number of rounds required by each operation. Dutta et al. [7] developed the first single-writer/multi-reader (SWMR) algorithm, where all operations complete in a single round. Such operations are called *fast*. The authors showed that fast operations are possible only if the number of readers in the system is constrained with respect to the number of servers. They also showed that it is impossible to have MWMR register implementations where *all* operations are fast. To remove the constraint on the number of readers, Georgiou et al. [14] introduced *semifast* implementations where at most one complete two-round read operation is allowed per write operation. They also showed that semifast MWMR register implementations are impossible.

Algorithm SFW, developed by Englert et al. [8], was the first to allow both reads and writes to be fast in the MWMR setting. The algorithm used *quorum systems*, sets of intersecting subsets of servers, to handle server failures. To decide whether an operation could terminate after its first round, the algorithm employed two *predicates*, one for the write and one for read operations.

A later work by Georgiou et al. [11] identified two weaknesses of algorithm SFW with respect to its practicality: (1) the computation required for calculating the predicates is NP-complete, and (2) fast operations are possible only when every *five* or more quorums have a non-empty intersection. To tackle these issues the authors introduced two new algorithms. The first algorithm, called APRX-SFW, proposed a polynomial $\log$-approximation solution for the computation of the predicates in SFW. This would allow faster computation of the predicates while potentially increasing the number of two-round operations. To tackle the second weakness of SFW, the authors presented algorithm CWFR that uses *Quorum Views* [13], client-side decision tools, to allow some fast *read* operations without additional constraints on the underlying quorum system. Write operations in this implementation always take two rounds to complete. *See [12] for more discussion on related work.*

All the above algorithms have been rigorously proven to guarantee atomic consistency and theoretical analyses provide clues for their efficiency. However, little work has been done to study the practicality and effectiveness of these approaches over realistic, planetary-scale adverse environments. The efficiency of these approaches would greatly affect any distributed file or storage system built on top of them. Furthermore, as the effect of the performance metrics is orthogonal one may wonder: "When shall we prefer reduced communication for increased computation and when vice-versa?"

Contributions and results: In this work we attempt to provide empirical evidence on the efficiency and practicality of MWMR atomic register implementations and give an answer to the above question. More precisely, we experimentally evaluate the performance of algorithms SFW, APRX-SFW, and CWFR as they are the only known algorithms to this writing to allow single round writes and reads in the MWMR model. To observe the benefits of reducing the number of communication rounds per operation, we compare our findings with the performance of algorithm SIMPLE, an all two-round algorithm. To test the *efficiency* of the algorithms we first implement them on NS2 [2], and to test their *practicality* we deploy them on PlanetLab [1], a planetary scale real-time network platform. In more detail:

(a) To test *efficiency*, we simulate the algorithms on the NS2 [2] network simulator. The controlled environment of NS2 allows us to test the performance of the algorithms under different environmental conditions. In particular, the operation latency of the algorithms is tested under the following simulation scenarios: (1) Variable number of readers/writers/servers, (2) Deployment of different quorum constructions, and (3) Variable network delay.

Our obtained results on NS2 confirm the high computation demands of SFW over APRX-SFW as theoretically proven in [11]. In addition, they suggest that the computation burden needed by APRX-SFW and CWFR is lower than the communication cost of a second communication round in most scenarios, by comparing the two algorithms with algorithm SIMPLE. In terms of scalability, it appears that every algorithm suffers a performance degradation as the number of participants is increasing. Finally, we observe that long network delays promote algorithms with high computational demands and fewer communication rounds (such as algorithm APRX-SFW). In general we can say that the simulation promotes CWFR as the most efficient algorithm, in most of the scenarios.

(b) To test *practicality*, we implement and deploy the algorithms on PlanetLab [1], an overlay network infrastructure composed of machines that are located throughout the globe. Given the adverse and unpredictable conditions of this real-time system, we measure and compare the operation latency of the algorithms under two different families of service scenarios: (1) Variable number of readers/writers/servers, (2) Deployment of different quorum constructions. In our implementations communication was established via TCP/IP and the C/C++ programming language and sockets were used for interfacing with TCP/IP.

Our findings also suggest that the computation burden of algorithms CWFR and APRX-SFW is lower than the communication cost of the second round required by algorithm SIMPLE in most of the scenarios. More precisely, computation does not appear to have a great impact on the performance of the algorithms. This is partly due to the fact that both CWFR and APRX-SFW exhibit a percentage of slow operations under 20%. Also, unlike NS2, there are a number of machines executing our protocols and thus computation is no longer performed by a single machine. In terms of scalability, we still observe a degradation on the performance of the algorithms as the number of participants increases. In addition, we observe that the intersection degree of the quorum system can play a decisive factor as it affects in a large degree the performance of APRX-SFW. The higher the intersection degree the more reads/writes can be fast. Even though this is true, the average latency achieved by APRX-SFW in environments with small intersection degree is not much higher than the latency of the competition. In general we can say that PlanetLab promotes APRX-SFW as the most practical algorithm, in most of the scenarios.

II. MODEL AND DEFINITIONS

We consider the asynchronous message-passing model. There are three distinct finite sets of crash-prone processes: a set of readers $\mathcal{R}$, a set of writers $\mathcal{W}$, and a set of servers $\mathcal{S}$. The identifiers of all processes are unique and comparable. Communication among the processes is accomplished via reliable communication channels.

Servers and quorums: Servers are arranged into intersecting sets, or *quorums*, that together form a quorum system $\mathbb{Q}$. 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 $\bigcap_{Q\in\mathcal{A}} Q \neq \emptyset$. We call n the *intersection degree* of $\mathbb{Q}$. 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 $2 \leq k \leq n$.

Processes may fail by crashing. A process i is *faulty* in an execution if i crashes in the execution (once a process crashes, it does not recover); 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 study atomic read/write register implementations, where the register is replicated at servers. Reader/writer p *invokes* a read(resp. write) operation when it receives a read(resp. write) request. An operation terminates with the corresponding acknowledgment, called the *response* of the process. Finally, an operation π is *incomplete* when the invocation of π does not have the associated response; otherwise 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.

We say that an operation (read or write) π_1 *precedes* another operation π_2, or π_2 *succeeds* π_1, if the response for π_1 precedes in real time the invocation of π_2. Two operations are *concurrent* if neither precedes the other.

Atomicity provides the illusion that operations are performed sequentially even though they may be performed concurrently. Notice that every process in the system must determine the same *partial* sequence of operations.

Efficiency and Fastness: We measure the efficiency of an atomic register implementation in terms of *computation* and *communication round-trips* or *rounds*. A round is defined as follows [7], [14], [13]: 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. We use quorum systems and ***tags*** to impose an ordering on, the values written to the register replicas. A tag contains a timestamp (counter) and the writer's identifier.

III. Algorithm Overview

We now overview the algorithms we evaluate. A more comprehensive description of the algorithms appears in [12]. We assume that the algorithms use quorum systems and follow the failure model presented in Section II. Thus, termination is guaranteed as long as each read and write operation waits for replies from a single quorum. To order the written values the algorithms use (tag, value) pairs.

Algorithm SIMPLE is a generalization of the algorithm presented by Attiya et al. [5] for the SWMR model. The algorithm uses two round read and write protocols. During the first round, read and write operations query a quorum of servers for the maximum tag. In the second round a write operation generates a new tag and propagates the new tag along with the value to be written to a quorum of servers. A read operation propagates the maximum tag-value pair it discovered in its first round to a quorum of servers, before returning the value associated with the maximum tag.

Algorithm SFW [8] introduces the server side ordering technique which moves partial responsibility for the ordering of the written values to the servers. Thus, servers increase their local tag whenever they receive a write request. In SFW, reads and writes may perform 1 or 2 communication rounds. During their first round they query a quorum of servers for the latest tags introduced in the system. The messages received are used to evaluate a write predicate on the writer side and a read predicate on the reader side. If the predicate is satisfied then the write (resp. read) completes in a single communication round. Otherwise they perform a second communication round. During this round the write operation propagates the maximum discovered tag with the value to be written. A read operation propagates the maximum tag-value pair discovered in the first round. Algorithm APRX-SFW [11] improves the computational complexity of the predicates by approximating their validation. The read and write protocols remain the same as in SFW.

Algorithm CWFR [11] allows 2 round writes and 1 or 2 round reads on top of general quorum constructions. The write protocol is identical to the that of SIMPLE. The read protocol queries a quorum of servers during its first round and tries to detect the latest potentially completed write operation among the replies. To do so, CWFR utilizes a generalized form of Quorum Views [13] for the MWMR model. If the detection of such write is possible then the read returns that write's value in a single round. Otherwise, the read performs a second round and propagates the maximum tag-value pair it discovered during its first round.

Table I accumulates the theoretical communication and computation burdens of the four algorithms. Columns WR and RR show how many rounds are required per write and read operation respectively. Columns RC and WC present the computation required by each algorithm and the last column the technique the algorithm incorporates to decide on the values read/written on the atomic register.

IV. Experimentation Platform

This section presents the testbed we considered for the two experimentation environments. For both NS2 [2] and Planetlab [1] we describe the parameters we use in our implementations along with our execution scenarios.

The general testbed of our experiments consists of a set of writer, reader, and server nodes. Servers are organized in majority quorums. As discussed in [8], assuming $|\mathcal{S}|$ servers out of which f can crash, we can construct an $(\frac{|\mathcal{S}|}{f} - 1)$-wise quorum system $\mathbb{Q}$. Each quorum $Q \in \mathbb{Q}$ has size $|Q| = |\mathcal{S}| - f$. The processes are not aware of f but they are aware of the quorum system which is generated *a priori* and is distributed to each node in the service.

A. NS2 Simulator

Communication between the nodes is established via bidirectional links, with 1Mb bandwidth, latency of $10ms$, and DropTail queue; this resembles an average connection in real environments. To model local asynchrony, each node sends messages after a random delay between 0 and 0.3 sec. From pretests we run, we observed that larger delays did

Algorithm	WR	RR	RC	WC	Decision Tool
SIMPLE	2	2	$O(\|\mathcal{S}\|)$	$O(\|\mathcal{S}\|)$	Highest Tag
SFW	1 or 2	1 or 2	$O(2^{\|\mathbb{Q}\|-1})$	$O(2^{\|\mathbb{Q}\|-1})$	Predicates
APRX-SFW	1 or 2	1 or 2	$O(\|\mathcal{W}\|\|\mathcal{S}\|^2\|\mathbb{Q}\|)$	$O(\|\mathcal{S}\|^2\|\mathbb{Q}\|)$	Predicate Approximation
CWFR	2	1 or 2	$O(\|\mathcal{S}\|\|\mathbb{Q}\|)$	$O(\|\mathcal{S}\|)$	Quorum Views / Highest Tag

Table I
THEORETICAL COMPARISON OF THE FOUR ALGORITHMS.

not affect the trend of our results but rather the simulation termination time. So we used the largest value at which our simulations were terminating in a reasonable time (around a week). We ran NS2 in Ubuntu, on a Centrino 1.8GHz processor. The average of 5 samples per scenario provided the stated operation latencies.

We use the positive time parameters $rInt = 4sec$ and $wInt = 4sec$ to model operation frequency. Readers and writers pick a uniformly random time between $[0 \ldots rInt]$ and $[0 \ldots wInt]$, respectively, to invoke their next read (resp. write) operation. We allow each participant to perform up to 25 operations (this totals to 500-4000 operations in the system). As testing the performance of the algorithms under high concurrency was our main goal, we decided to allow more operations per participant than allowing more than 80 participants in the service; larger participation was delaying the simulation without providing any further insight on the comparison of the algorithms.

B. Planetlab Configuration

PlanetLab is an overlay network infrastructure, composed of 1075 machines at 525 locations worldwide. As opposed to NS2, Planetlab provides no control over the network components and execution sequence of the algorithms.

Our implementations were written in C++ programming language and communication between the nodes was established via C sockets and TCP/IP. For our experiments we used 20 physical machines (see [12] for a description of these machines). Each machine was used to host one or more processes. All the servers were hosted on the first 10 machines in a round robin fashion. For instance, if the number of servers were 15 then the first 5 machines were hosting two server instances. The first 10 machines were chosen to split our servers geographically throughout the globe. The readers and the writers ran on all 20 machines starting from the 10^{th} machine and following a round robin technique. Thus, some of the first 10 machines could run a server and a client at the same time.

We use the time parameters $rInt = 10sec$ and $wInt = 10sec$ to model operation frequency. Readers and writers pick a uniformly at random time between $[0 \ldots rInt]$ and $[0 \ldots wInt]$, respectively, to invoke their next read (resp. write) operation. Each reader and writer chooses a new timeout every time their latest operation is completed. For our experiments we allow each participant to perform up to 20 operations (this totals to 400-3000 operations).

Difficulties with Planetlab: The hectic environment in Planetlab introduced the following difficulties.

Multiplexing: The traditional forking approach for the client-server communication would have introduced new processes in the system (children). This may have affected the correctness of the algorithms. Instead, the multiplexing technique allowed the server to individually communicate with every client via a non-blocking socket connection without affecting correctness.

Resource Limits: PlanetLab offers limited resources for each experiment. Also, the master machine that starts the experiments has limitations on the number of processes that can be initiated concurrently. For these reasons we bounded the number of readers and writers to 40.

Sampling: The extremely adverse environment of Planetlab affected the results of the algorithms when their runs were separated by large time intervals. Running the algorihtms in an alternating fashion overcame this problem.

Weighted Average Time: Reader and writer failures affected the computation of the average latency time. We obtained a weighted average by dividing the total operation latency over the total number of terminated processes for all 5 sample runs of an algorithm.

C. Scenarios

The scenarios were designed to test (i) scalability of the algorithms as the number of readers, writers and servers increases, (ii) the relation between quorum system deployment and operation latency, and (iii) whether network delays may favor the algorithms that minimize the communication rounds (only on NS2). In particular we consider the following parameters:

Number of Participants: We run every test with 10, 20, 40, and 80 readers and writers on NS2 and 10, 20, 30, 40 readers and writers on Planetlab. To test the scalability of the algorithms with respect to the number of replicas in the system we ran all of the above tests with 10, 15, 20, and 25 servers. Such tests highlight whether an algorithm is affected by the number of participants in the system, and subsequently by the increasing number of concurrent read and write operations. The more the servers on the other hand, the more concurrent values may coexist in the service. So, algorithms like APRX-SFW and CWFR, who examine all the discovered values and do not rely on the maximum value, may suffer from local computation delays.

Quorum System Construction: Majority quorums can provide quorum systems with high intersection degree. As the number of servers $|\mathcal{S}|$ varies between 10, 15, 20, and 25, we ran the tests for two different failure values, i.e. $f = 1$ and $f = 2$. This affects our environment in two ways: (i) we get quorum systems with different quorum intersection degrees, and (ii) we obtain quorum systems with different number of quorum members. Changes on the quorum constructions help us evaluate how the algorithms handle various intersection degrees and quorum systems of various memberships.

Network Latency: In a controlled environment like NS2 it is interesting to examine what is the impact of the network latency on the overall performance of the algorithms. In this scenario (only applied in NS2) we examined whether higher network latencies may favor algorithms (like CWFR and APRX-SFW) that, although they have high computation demands, they allow single round operations. For this scenario we change the latency of the network from 10ms to 500ms and we deploy a 6-wise quorum construction.

V. RESULTS AND ANALYSIS

We now discuss our experimentation results. Our discussion is accompanied by sample plots for each scenario in Figures 1 and 2. The complete list of plots appears in [12].

Before we proceed to compare the three algorithms under investigation we established a proof-of-concept comparison between algorithms SFW and APRX-SFW on NS2. Examining the latency of the two algorithms, including both communication and computation costs, provides evidence of the heavy computational burden of algorithm SFW. In particular, we obtained the following numbers for the average read latency: (i) $|\mathcal{S}| = 15$, SFW $RL \approx 10s$, APRX-SFW $RL \approx 2s$, and (ii) $|\mathcal{S}| = 20$, SFW $RL \approx 1200s$, APRX-SFW $RL \approx 2s$. It appears that the average read latency of algorithm SFW grows exponentially, with respect to the number of servers (and thus quorum members) in the deployed quorum system. On the other hand the average latency of read operations in APRX-SFW grows very slowly. In addition, the number of slow operations in APRX-SFW support the fact that algorithm APRX-SFW implements a $\log |\mathcal{S}|$-approximation of SFW. These results demonstrate the performance benefit of using algorithm APRX-SFW over algorithm SFW. (More details in [12].)

We now compare algorithm APRX-SFW with algorithm CWFR. To examine the impact of the computation on the operation latency, we compare both algorithms to algorithm SIMPLE. Recall that algorithm SIMPLE requires insignificant computation. Thus, the latency of an operation in SIMPLE directly reflects four communication delays (i.e., two rounds).

A. NS2 Results

In the next paragraphs we present how the read and write operation latency is affected by the scenarios we discussed in Section IV-C as those were simulated on NS2.

Variable Participation: For this scenario we tested the scalability of the algorithms when the number of readers, writers, and servers changes. Sample plots for this scenario appear in Figures 1a(i) and 1a(ii). These plots present the average latency of read operations when the number of writers is 10 and 80.

From our results follows that the number of writers affects the latency of read and write operations in APRX-SFW significantly, while they do not have a significant impact on the performance of the other two algorithms. As the number of writers grows we observe that the latency of operations for APRX-SFW increases. Such effect is not caused by the number of reader participants.

As can be seen in Figure 1a(i) both CWFR and APRX-SFW maintain an average read latency of 1.8s when no more than 20 writers exist in the service. That is true for most of the server participation scenarios and leaves the latency of read operations for the two algorithms below the latency of read operations in SIMPLE. That suggests that the computation burden does not exceed the latency added by a second communication round. Once the number of writers grows larger than 40 (Figure 1a(ii)) the average read latency of APRX-SFW is higher than the average latency of SIMPLE, even though it allows single round operations. This suggests that the computation takes longer than the second communication round. That was expected, as in APRX-SFW each read operation may examine as many tags as the number of writers. Unlike APRX-SFW, algorithm CWFR seems to sustain the number of slow reads below 30% and the average read latency under 2s in all scenarios. The only case where APRX-SFW over-performs CWFR is when we deploy a quorum system with a large intersection degree (20-wise). Although APRX-SFW allows less slow reads than CWFR, the computation demands of APRX-SFW forced the average read latency of the algorithm to be identical to the average read latency of CWFR.

The only algorithm that allows single round write operations is APRX-SFW. The number of writers and servers however, introduce high computation demands for the write operations. As a result, despite the fast writes, the average write latency of APRX-SFW can be higher than the average write latency of the other two algorithms. Note here that unlike the read operations, writes can be fast in APRX-SFW only when the write predicate holds. So in a 4-wise quorum we observed that every write operation in APRX-SFW performs two communication rounds since the predicate never holds.

Quorum Construction: Figure 1b plots the average read (1b(i)) and write (1b(ii)) latency with respect to the number of quorum members in the quorum system.

We observe that incrementing the number of servers, and thus cardinality of the quorum system, reduces the percentage of slow reads for both APRX-SFW and CWFR. Read latency (Figure 1b(i)) on the other hand is not proportional to the reduction on the amount of slow operations. Both algorithms APRX-SFW and CWFR experience an incrementing trend on the latency of the read operations as the number of servers increases. Notice that the read latency in SIMPLE also follows an increasing trend even though every read operation requires two communication rounds to complete. This suggests that as the servers increase in number, each reader needs to exchange more messages, and hence the communication delay is worse. The latency of read operations in CWFR is not affected greatly as the number of servers changes. As a result, CWFR appears to maintain a

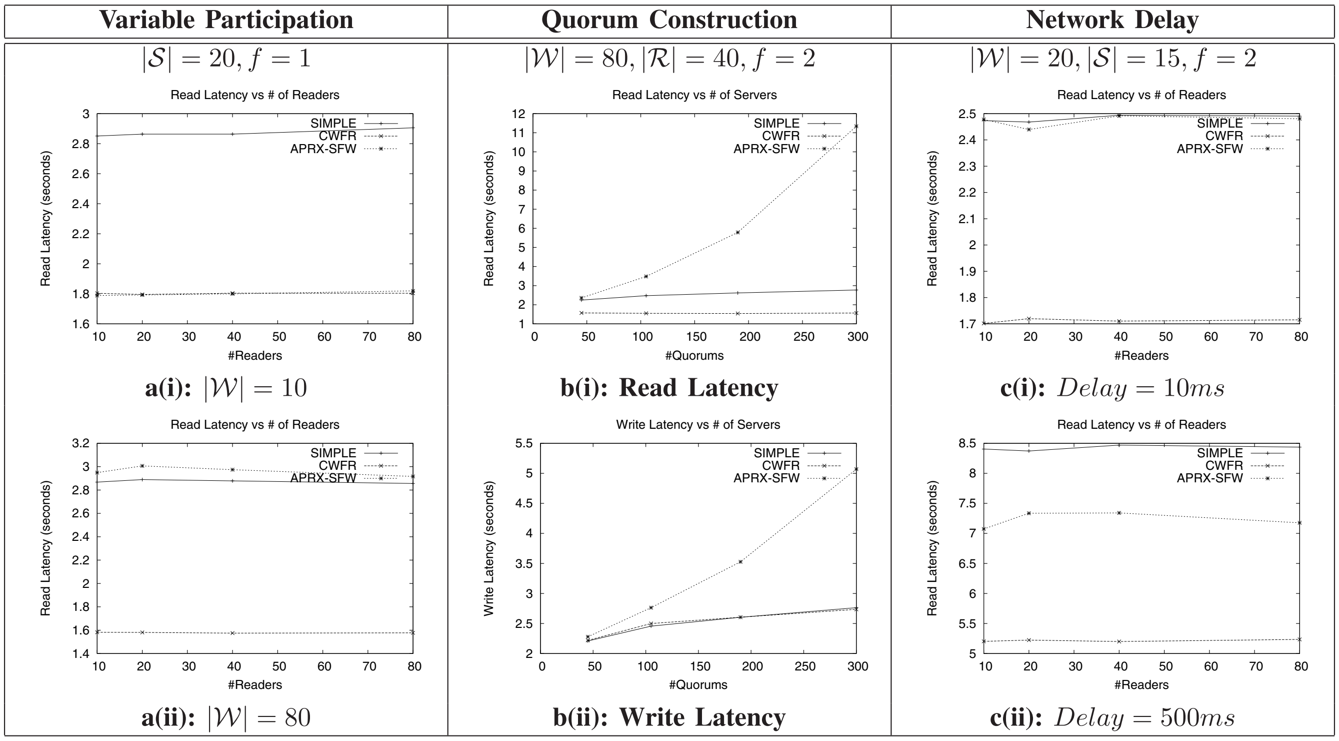

Figure 1. NS2 simulation plots

read latency close to 1.5 sec in every scenario. With this read latency CWFR over-performs algorithm SIMPLE in every scenario as the latter maintains a read latency between 2.5 and 3 sec. Unlike CWFR, algorithm APRX-SFW experiences a more aggressive change on the latency of read operations. The read latency in APRX-SFW is affected by both the number of quorums in the system, and the number of writers in the system. As a result the latency of read operations in APRX-SFW in conditions with a large number of quorum members and writers may exceed the read latency of SIMPLE up to 5 times. The reason for such performance is that every read operation in APRX-SFW the reader examines the tags assigned to every writer and for every tag runs the approximation algorithm on a number of quorums in the system. The more the writers and the quorums in the system, the more time the read operation takes to complete.

Similar observations can be made for the write operations. Observe that although both CWFR and SIMPLE require two communication rounds per write operation, the write latency in these algorithms is affected negatively by the increase of the number of quorums in the system. Also note that the read and write latency of SIMPLE is almost identical. This proves the fact that the computation demands in either operation is also identical. As for APRX-SFW, the increase on the number of servers reduces the amount of slow write operations but increases the average write latency. Comparing with the latency of read operations it appears that although APRX-SFW may allow more fast read operations than writes under the same conditions, the average read latency is higher than the average write latency. The simple explanation for this behavior lie on the evaluation of the read and write predicates. Each reader needs to examine the latest tags assigned to every writer in the system whereas each writer only examines the tags assigned to its own write operation.

Network Latency: During our last scenario we considered increasing the latency of the network infrastructure from 10ms to 500ms. With this scenario we want to examine whether in slow networks is more preferable to minimize the amount of rounds, even if that means higher computation demands. We considered just a single setting for this scenario where the number of servers is 15, the maximum number of failures is 2.

In order to establish meaningful conclusions we need to compare the outcomes of the operation performance of this scenario with the respective scenario where the latency is 10ms. We notice that the largest network delay reduces the amount of slow reads for both CWFR and APRX-SFW. In addition the delay indeed helps APRX-SFW to perform better than SIMPLE in scenarios where APRX-SFW was performing identical or worse than SIMPLE when the delay was 10ms. This can be seen in Figures 1c(i) and 1c(ii). As we can see in Figure 1c(i) when the delay was 10ms, the average read latency of APRX-SFW was aligning with the average read latency of SIMPLE. When we increased the network delay to 500ms (Figure 1c(ii)) the average read latency of APRX-SFW was remarkably smaller than the average latency of SIMPLE under the same participation and failure conditions. Similar observations can be made for the write operations.

So we can safely conclude that the network delay can be one of the decisive factors on which algorithm is suitable for a particular application.

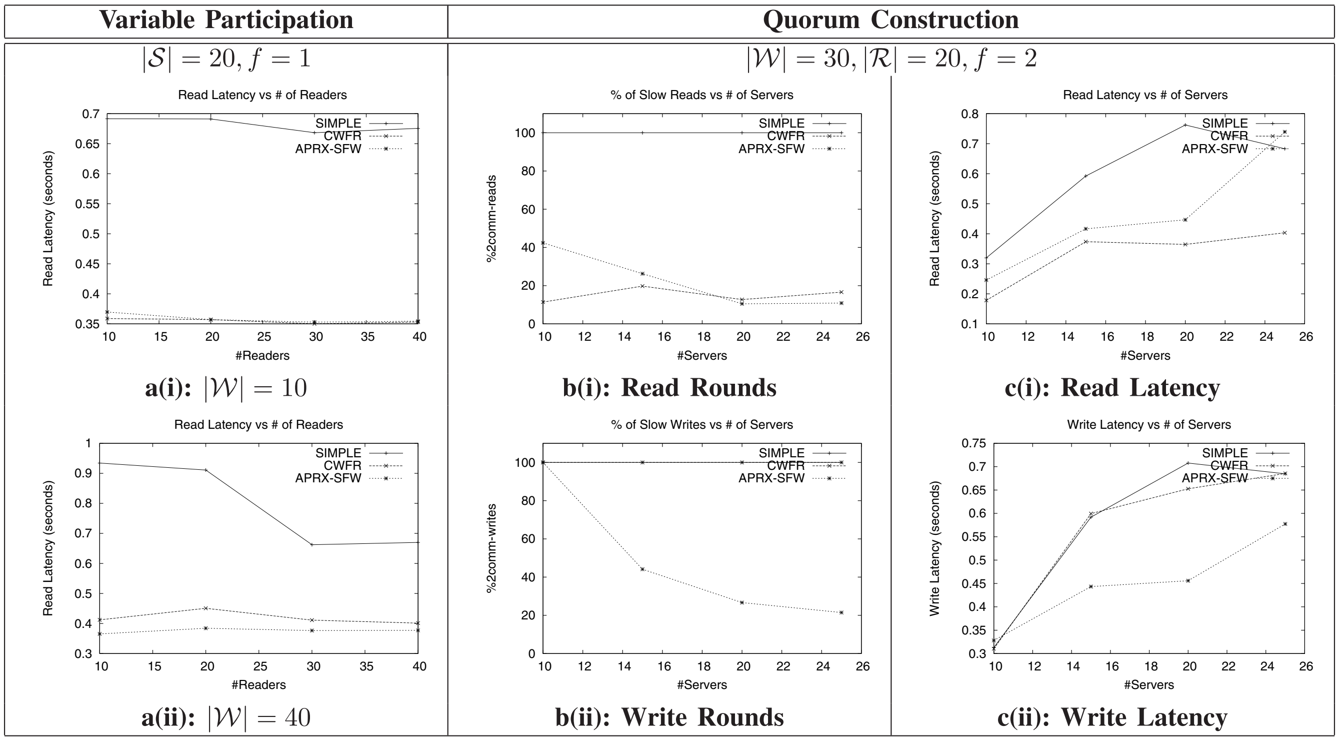

Figure 2. PlanetLab plots

B. PlanetLab results

Here we discuss our results obtained by implementing the first two scenarios of Section IV-C in PlanetLab. PlanetLab machines often go offline unexpectedly preventing some scenarios to finish and collect their data. As those cases are not frequent we ignore them in our conclusions below.

Variable Participation: Figures 2a(i) and 2a(ii) present an example on how the average read latency is affected by the number of writers. For both plots, we use a 19-wise quorum system ($|\mathcal{S}| = 20, f = 1$). and we vary the number of readers. Figure 2a(i) presents a run with 10 writers and Figure 2a(ii) the corresponding run with 40 writers.

Our results in PlanetLab, unlike the results in NS2, show that APRX-SFW over-performs CWFR in most of the cases. This makes it evident that the adverse environment of a real-time system diminishes the high concurrency between read and write operations. Notice that APRX-SFW and CWFR require higher computation demands when multiple writers try concurrently to change the value of the register. Computation is decreased when writes are consecutive. Thus, low concurrency favors algorithms with high computation and low communication demands.

As in NS2, the number of readers has little effect on the average read latency for both APRX-SFW and CWFR. On the other hand, as can also be seen in Figure 2a the read latency of the two algorithms increases as the number of writers grows. Unlike our findings in NS2, the number of writers has a greater impact on the performance of CWFR rather on APRX-SFW. This is another evidence that write operations do not overlap due to the adverse environment, and thus APRX-SFW can discover a valid tag by examining a subset (and not all) of the candidate tags. Both CWFR and APRX-SFW require fewer than 20% of reads to be slow almost in all scenarios. The only scenario where the percentage of slow reads for APRX-SFW rise above 50% is when we deploy a 4-wise quorum system. This demonstrate the dependence of the predicates on the intersection degree of the quorum system used. We discuss this relation in the next paragraph. The large percentage of fast read operations keeps the overall latency of read operations for the two algorithms below the latency of read operations of SIMPLE; this suggests that the computation burden does not exceed the latency added by a second communication round.

Similar to the read operations, the average write latency is only affected by the number of writers and servers in the system. The only algorithm that allows single round write operations is APRX-SFW. We observe that the percentage of slow writes increases as the number of writers increases but it decreases as the number of servers increases. This is expected as more writers result in more write collisions and thus more tags propagated concurrently in the system. It is very interesting that the average write latency of APRX-SFW is below the write latency of the algorithms that require two communication rounds. This shows that the computation burden of APRX-SFW does not exceed the time of the second communication round. Note here that unlike the read operations, writes can be fast in APRX-SFW only when the write predicate holds. So when deploying a 4-wise quorum system that does not allow for the write predicate to hold every write operation in APRX-SFW performs two communication rounds. In such case we observed that the write latency of APRX-SFW was almost identical to the delay of the other two algorithms; this is another evidence that the computation burden of APRX-SFW does not affect the latency of writes by a high margin in real systems.

Quorum Construction: Figures 2b and 2c illustrate an example of the performance of read and write operations (communication rounds and latency) with respect to the number of servers (and thus quorum members in the quorum system).

From the plots it is clear that APRX-SFW is affected by the number of quorums and in extent by the intersection degree of the quorum system. In particular, we observe that the cardinality of the quorum system reduces the percentage of slow reads for APRX-SFW (see Figure 2b(i)). On the other hand, the number of slow reads in CWFR does not seem to be affected. The distinction between APRX-SFW and CWFR is depicted nicely when the line of the slow read percentage of APRX-SFW crosses below the line of CWFR when the intersection degree grows to $\frac{20}{2} - 1 = 9$, and remains lower for larger values of the intersection degree. Operation latency on the other hand is not proportional to the reduction on the amount of slow operations. Both algorithms APRX-SFW and CWFR experience an incrementing trend on the average read latency as the number of servers increases. As in NS2, the read latency in SIMPLE also follows an increasing trend, suggesting an increase on the communication cost.

Similar observations can be made for the write operations. Observe that although both CWFR and SIMPLE require two communication rounds per write operation, the write latency in these algorithms is affected negatively by the increase of the number of quorums in the system. As mentioned before this is an evidence of the higher communication demands when we increase the number of servers. We also note that the average write latency in SIMPLE is almost identical to the average read latency of the same algorithm. This proves the fact that the computation demands in either operation is also identical. As for APRX-SFW we observe that the increase on the number of servers reduces the amount of slow write operations, more than what we observed during the NS2 simulation. Also we observe that the average write latency of APRX-SFW increases as the number of quorums increases in the system. Interestingly however, unlike the findings in NS2, the latency of writes remains in most cases below the competition. Comparing with the latency of read operations it appears that although APRX-SFW may allow more fast read operations than writes under the same conditions, the average latency of each read is almost the same as the average write latency.An example of this behavior can be seen in Figures 2b and 2c. In this example slow reads can drop as low as 10% (Figure 2b(i)) and the average read latency (Figure 2c(i)) climbs to an average of 0.5s (the 0.7s point appears to be an outlier). On the other hand no less than 20% slow writes are allowed (Figure 2b(ii)) and the average write latency climbs just above 0.5s (Figure 2c(ii)). The simple explanation for this behavior lies on the evaluation of the read and write predicates: each reader needs to examine the latest tags assigned to every writer in the system whereas each writer only examines the tags assigns to its own write operation.

VI. CONCLUSIONS

In this work we have implemented and compared the MWMR atomic register algorithms SIMPLE, APRX-SFW, and CWFR on the NS2 simulator and on PlanetLab. Combining our findings in both experimentation environments, we may generally conclude that in most runs, algorithms APRX-SFW and CWFR perform better than algorithm SIMPLE. This suggests that the additional computation incurred in these two algorithms does not exceed the delay associated with a second communication round. Furthermore, depending on the server configuration used and the frequency of read and write operations, we draw a clear line when algorithm CWFR should be preferred over algorithm APRX-SFW and vice-versa.

REFERENCES

[1] PlanetLab. `http://www.planet-lab.org`.

[2] NS2 network simulator. http://www.isi.edu/nsnam/ns/.

[3] M. K. Aguilera, I. Keidary, D. Malkhi, J.-P. Martin, and A. Shraery. Reconfiguring replicated atomic storage: A tutorial. *Bulletin of the EATCS*, 102:84–081, 2010.

[4] H. Attiya. Robust simulation of shared memory: 20 years after. *Bulletin of the EATCS*, 100:99–114, 2010.

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

[6] G. Chockler, I. Keidar, R. Guerraoui, and M. Vukolic. Reliable distributed storage. *IEEE Computer*, 2008.

[7] P. Dutta, R. Guerraoui, R. R. Levy, and A. Chakraborty. How fast can a distributed atomic read be? In *Proc. of PODC 2004*.

[8] B. Englert, C. Georgiou, P. M. Musial, N. Nicolaou, and A. A. Shvartsman. On the efficiency of atomic multi-reader, multi-writer distributed memory. In *Proc. of OPODIS 2009*.

[9] R. Fan and N. Lynch. Efficient replication of large data objects. In *Proc, of DISC 2003*, pp. 75–91.

[10] E. Gafni and L. Lamport. Disk paxos. *Distributed Computing*, 16(1):1–20, 2003.

[11] C. Georgiou, N. Nicolaou, A. Russel, and A. A. Shvartsman. Towards feasible implementations of low-latency multi-writer atomic registers. In *Proc. of NCA 2011*, pp. 75–82.

[12] C. Georgiou and N. C. Nicolaou. On the practicality of atomic MWMR register implementations. In arXiv:1111.2693v2.

[13] C. Georgiou, N. C. Nicolaou, and A. A. Shvartsman. On the robustness of (semi) fast quorum-based implementations of atomic shared memory. In *Proc. of DISC 2008*, pp. 289–304.

[14] C. Georgiou, N. Nicolaou, and A. A. Shvartsman. Fault-tolerant semifast implementations of atomic read/write registers. *J. Parallel. and Distributed Comp.*, 69(1):62–79, 2009.

[15] A. Shraer, J.-P. Martin, D. Malkhi, and I. Keidar. Data-centric reconfiguration with network-attached disks. In *Proc. of LADIS 2010*.

[16] S. Yasushi, F. Svend, V. Alistair, M. Arif, and S. Susan. FAB: building distributed enterprise disk arrays from commodity components. In *Proc. of ASPLOS 2004*, pp. 48–58.