Separating the WHEAT from the Chaff: An Empirical Design for Geo-Replicated State Machines

João Sousa and Alysson Bessani
LaSIGE, Faculdade de Ciências, Universidade de Lisboa, Portugal

***Abstract*—State machine replication is a fundamental technique for implementing consistent fault-tolerant services. In the last years, several protocols have been proposed for improving the latency of this technique when the replicas are deployed in geographically-dispersed locations. In this work we evaluate some representative optimizations proposed in the literature by implementing them on an open-source state machine replication library and running the experiments in geographically-diverse PlanetLab nodes and Amazon EC2 regions. Interestingly, our results show that some optimizations widely used for improving the latency of geo-replicated state machines do not bring significant benefits, while others – not yet considered in this context – are very effective. Based on this evaluation, we propose WHEAT, a configurable crash and Byzantine fault-tolerant state machine replication library that uses the optimizations we observed as most effective in reducing SMR latency. WHEAT employs novel voting assignment schemes that, by using few additional spare replicas, enables the system to make progress without needing to access a majority of replicas. Our evaluation shows that a WHEAT system deployed in several Amazon EC2 regions presents a median latency up to 56% lower than a "normal" SMR protocol.**

I. Introduction

State machine replication (SMR) [20], [26] is a well-known technique to implement fault-tolerant services. The basic idea is to have an arbitrary number of clients issuing requests to a set of replicas in such a way that: (1) all correct replicas execute the same sequence of requests; and (2) clients receive a reply for their requests despite the failure of a fraction of these replicas. This technique is adopted by many modern distributed systems, ranging from cluster-based coordination services (e.g., Chubby [7] and Zookeeper [17]) to geo-replicated databases (e.g., Spanner [10]).

The core of a replicated state machine is an agreement protocol (e.g., Paxos [21]) used to establish a total order in the messages delivered to the replicas. This protocol, which usually requires multiple communication steps, is responsible for a significant latency overhead when SMR is employed for geo-replication. To mitigate this problem, many SMR protocols have been proposed for wide area networks (WANs) (e.g., [1], [23], [24], [29]). These WAN SMR protocols employ optimizations to reduce latency, usually by decreasing the number of communication steps across the WAN. All these protocols were evaluated in real, emulated or simulated environments, showing the proposed optimizations were indeed effective in decreasing the protocol latency.

However, even though such evaluations generally use comparable methodologies, they do not use the same experimental environments and codebase across independent works. This lack of a common ground makes it hard to not only compare results across distinct papers, but also to assess which optimizations are actually effective in practice. This is aggravated by the fact that these evaluations tend to compare SMR protocols in an holistic manner and generally do not compare individual optimizations.

In this paper we present an extensive evaluation of several latency-related optimizations scattered across the literature (both for local data centers and geo-replication) using the same testbeds, methodology and codebase (see §III). More specifically, we selected a subset of optimizations for decreasing the latency of strongly-consistent geo-replicated systems, implemented them in the BFT-SMaRt replication library [6] (see §II) and deployed the experiments in the PlanetLab testbed and in the Amazon EC2 cloud. During the course of this evaluation, we obtained some unexpected results. The most notorious example is related with the use of multiple leaders – a widely accepted optimization used by several WAN-optimized protocols such as Mencius [23] and EPaxos [24]. Specifically, our results indicate that this optimization does not bring significant latency reduction just by itself; instead, we observed that using a fixed leader in a fast replica is more effective (and simpler) strategy to reduce latency. Moreover, we also found that adding a few more replicas to the system without increasing the size of the quorums required by the protocol may lead to significant latency improvements. These results shed light on which optimizations are really effective for improving the latency of geo-replicated state machines, and *constitute the first contribution of this paper*.

The aforementioned results showcasing the benefit of having extra replicas without necessarily increasing the quorum sizes required by the system led to *the second contribution of the paper*: two novel vote (weight) assignment schemes designed to preserve (CFT and BFT) SMR protocol correctness while also allowing the emergence of quorums of variable size (see §IV-B). By allowing quorums of different sizes, it is possible to avoid the need of accessing a majority of replicas – a requirement of many SMR protocols. We introduce two vote assignment schemes (for CFT [21] and BFT [8] SMR) and show that they enable the formation of safe and minimal quorums without endangering the consistency and availability of the underlying quorum system [22]. To the best of our knowledge, this is the first work that incorporates the idea of assigning different votes for different replicas (i.e., weighted replication) [14], [15], [25] in replicated state machines.

Our *third and final contribution* is the design, implementation and evaluation of WHEAT (WeigHt-Enabled Active replicaTion), a flexible WAN-optimized SMR protocol developed by extending BFT-SMaRt with the most effective optimizations (according to our experiments) and our vote assignment schemes (see §IV-A). The evaluation of WHEAT –

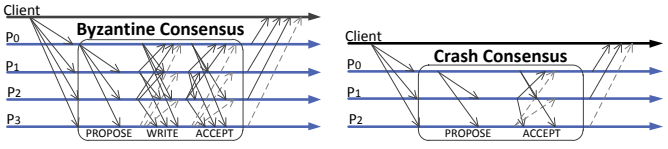
conducted in Amazon EC2 (see §IV-C) – shows that this protocol could outperform BFT-SMaRt by up to 56% in terms of latency. WHEAT was designed to operate both under crash and Byzantine faults. To the best of our knowledge, WHEAT is the first SMR protocol that is both optimized for geo-replication and capable of withstanding general Byzantine faults; Mencius [23] and EPaxos [24] tolerate only crash faults while BFT protocols like EBAWA [29] or Steward [1] requires either each replica to have a trusted component that can only fail by crash, or only tolerate Byzantine faults within a site (i.e., do not tolerate compromised sites), respectively.

II. State Machine Replication & BFT-SMaRt

In the SMR model [20], [26] an arbitrary number of clients send requests to a set of servers, which hosts replicas of a stateful service that updates its state after processing those requests. The goal of this technique is to make the state at each replica evolve in a consistent way, resulting in a service which is accurately replicated across all replicas. Since the service state was already updated by the time clients receive a reply from the service, this technique is able to offer *strong consistency*, i.e., linearizability [16]. To enforce this behavior, it is necessary that: (1) client requests are delivered to the replicas via total order broadcast; (2) replicas start their execution in the same state; and (3) replicas modify their state in a deterministic way.

All the experimental work done in this paper is based on the BFT-SMaRt open-source library [6]. BFT-SMaRt implements a modular SMR protocol on top of a Byzantine consensus algorithm [27]. Under favourable network conditions and the absence of faulty replicas BFT-SMaRt executes the message pattern depicted in Figure 1(a), which is similar to the PBFT protocol [8].

Clients send their requests to all replicas, triggering the execution of the consensus protocol. Each consensus instance i begins with one replica – the *leader* – proposing a batch of requests to be decided within that consensus. This is done by sending a PROPOSE message containing the aforementioned batch to the other replicas. All replicas that receive the PROPOSE message verify if its sender is the leader and if the batch proposed is valid. If this is the case, they register the batch being proposed and send a WRITE message to all other replicas containing a cryptographic hash of the proposed batch. If a replica receives $\lceil \frac{n+f+1}{2} \rceil$ WRITE messages with the same hash, it sends an ACCEPT message to all other replicas containing this hash. If some replica receives $\lceil \frac{n+f+1}{2} \rceil$ ACCEPT messages for the same hash, it deliver its correspondent batch as the decision for its respective consensus instance.



(a) BFT message pattern. (b) CFT message pattern.

Fig. 1. BFT-SMaRt message pattern during fault-free executions.

This is the message pattern that is executed if the leader is correct and the system is synchronous. If these conditions do not hold, the protocol needs to elect a new leader and force all replicas to converge to the same consensus execution. This mechanism is dubbed *synchronization phase* and is described in detail in [27].

As mentioned before, BFT-SMaRt can also be configured for crash fault tolerance (CFT). In this case it implements a Paxos-like message pattern [21], illustrated in Figure 1(b). The main differences are that the CFT protocol does not require WRITE messages, waits only for $\lceil \frac{n+1}{2} \rceil$ ACCEPT messages and requires a simple majority of non-faulty replicas to preserve correctness (the BFT protocol requires that more than three quarters of the replicas remain correct).

III. Experiments

In this section we present the experiments conducted to assess the effectiveness of certain optimizations proposed for SMR in wide area networks [8], [19], [21], [23], [24], [29], [30] and quorum systems [15], [25]. Before presenting our results, we describe some general aspects of our methodology.

A. Methodology

The considered hypotheses were evaluated by implementing the associated optimizations in BFT-SMaRt's code and executing it simultaneously with the original protocol. Our experiments focus on measuring latency instead of throughput, in particular the median and 90th percentile latency perceived by clients. This is due to the fact that throughput can be effectively improved by adding more resources (CPU, memory, faster disks) to replicas or by using better links, whereas geo-replication latency will always be affected by the speed of light limit and perturbations caused by bandwidth sharing.

During the experiments, clients were equally distributed across all hosts, i.e., a BFT-SMaRt replica and a BFT-SMaRt client were deployed at each host that executed the protocol. Similarly to other works (e.g., [17], [24]), each client invokes 1kB requests and receive 1kB replies from the replicas, which run a *null* service. Requests were sent to the replicas every 2 seconds, and each client writes its observed latency into a log file. This setup enabled us to retrieve results that are gathered under similar network conditions without saturating either the CPU or memory of the hosts used.

The experiments in which we evaluated optimizations to the SMR protocol were conducted mostly in PlanetLab,[1] a distributed testbed dedicated to computer networking and distributed systems research. We chose PlanetLab because this testbed is known for displaying unpredictable latency spikes and highly loaded nodes [12]. These conditions allow us to evaluate the optimizations within unfavourable conditions.

Since our experiments are designed to evaluate solely the client latency in fault-free executions, we only report executions in which all hosts were online. However, since PlanetLab's host are regularly restarted and sometimes become unreachable, we could seldom execute each experiment during the same amount of time [12]. Therefore, we had to launch multiple executions for the same experiment, so that within each execution there would be a period in which all hosts

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

were online. In any case, all experiments reported in this paper consider at least 24 hours of measurements.

All experiments were configured to tolerate a single faulty replica. Each experiment was executed using between three to five hosts spread through Europe. The unavailability of nodes already mentioned led us to use a total of eight hosts through all experiments (see Table I).

Country	City	Hostname
Poland	Wroclaw	planetlab1.ci.pwr.wroc.pl
England	London	planetlab-1.imperial.ac.uk
Spain	Madrid	planetlab2.dit.upm.es
Germany	Munich	planetlab2.lkn.ei.tum.de
Portugal	Aveiro	planet1.servers.ua.pt
Norway	Oslo	planetlab1.ifi.uio.no
France	Nancy	host4-plb.loria.fr
Finland	Helsinki	planetlab-1.research.netlab.hut.fi
Italy	Rome	planet-lab-node1.netgroup.uniroma2.it

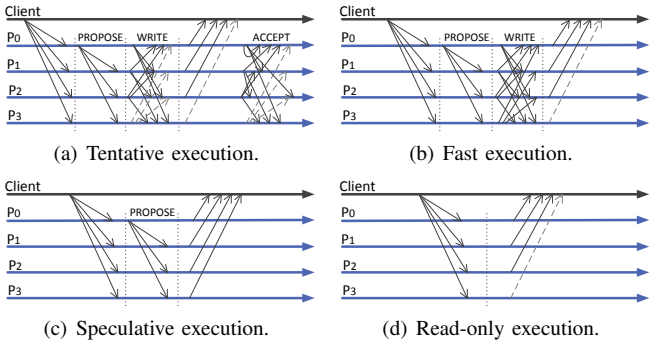
TABLE I. HOSTS USED IN PLANETLAB EXPERIMENTS

To validate our results in a global scale, two of the experiments described in the paper were executed on Amazon EC2,[2] using *t1.micro* instances distributed among five different regions. We used the same methodology described for the PlanetLab experiments.

B. Number of Communication Steps

The purpose of our first experiment is to observe how the client latency is affected by the number of communication steps performed by the SMR protocol. More precisely, we wanted to observe how efficient *read-only*, *tentative*, *speculative* and *fast* executions are in a WAN. The first two optimizations are proposed in PBFT [8], whereas the other two optimization are used by Zyzzyva [19] and Paxos at War [30] respectively. Since these optimizations target Byzantine-resilient protocols, we only evaluate them in BFT mode.

The message pattern for each of these optimizations is illustrated in Figure 2. Figure 1(a) displays BFT-SMART's standard message pattern (which is similar to PBFT's). Figure 2(a) displays the message pattern for tentative executions. This optimization consists of delivering client requests right after finishing the WRITE phase, thus executing the ACCEPT phase asynchronously. This optimization comes at the cost of (1) potentially needing to perform a rollback on the application state if there is a leader change, and (2) forcing clients to wait for $\lceil \frac{n+f+1}{2} \rceil$ messages from replicas (instead of $f+1$) [8]. Figure 2(b) displays the message pattern for fast executions. This optimization consists of delivering client requests right after gathering $\lceil \frac{n+3f+1}{2} \rceil$ WRITE messages (before the ACCEPT phase finishes). If such amount of WRITE messages arrive fast enough, the protocol can safely bypass the ACCEPT phase. Figure 2(c) displays the message pattern for speculative executions. This optimization enables the protocol to finish executions directly after the PROPOSE message is received in the replicas, as long as the clients are able to gather replies from all the replicas within a pre-established time window. If the clients are not able to gather all the replies within such time window, at least one additional round-trip message exchange is required to commit the requests. Figure 2(d) displays the message pattern for read-only executions. This optimization enables clients to obtain a response from the service in two communication steps. However, it can only be used to read the state from the service. Similarly to tentative executions, this optimization also demands that clients gather $\lceil \frac{n+f+1}{2} \rceil$ messages from replicas, *even for non-read-only operations*, to ensure linearizability [8].



(a) Tentative execution. (b) Fast execution.

(c) Speculative execution. (d) Read-only execution.

Fig. 2. Evaluated message patterns.

Setting: We created three BFT-SMART variants to evaluate fast, tentative and speculative executions (read-only executions were already supported). This experiment was deployed across hosts located in Nancy (leader), Wroclaw, Helsinki and Rome.

Results: The values for the median and 90th percentile latency for each client are shown in Figure 3. All evaluated optimizations exhibited latency reduction across all clients, with read-only executions finishing the protocol execution significantly faster than any of the other optimizations (i.e., 90th percentile latency from 43% to 63% smaller than in standard executions). Moreover, speculative executions also displayed significant latency reduction, reaching a 90th percentile latency 35% lower than standard execution. In the same way, tentative and fast executions also manage to reach a lower median and 90th percentile than standard executions, albeit with more modest differences. Furthermore, whereas fast executions displayed a latency decrease of about 10%, tentative executions managed to reduce latency by almost 20% (when compared to standard executions).

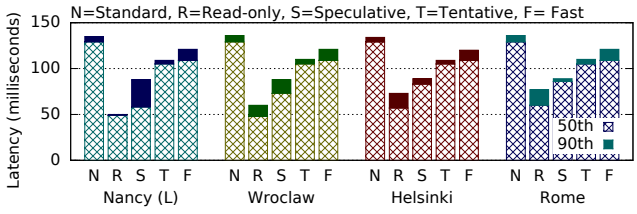


Fig. 3. Client latencies' 50th/90th percentile for each type of execution.

Main conclusion: The lowest latency displayed by read-only executions were to be expected, since they bypass all three communications steps executed between sending requests and gathering replies. Since speculative executions require the PROPOSE phase, they show higher latency than read-only executions. The advantage of tentative executions over fast executions can be explained by the fact that the latter require gathering WRITE messages from all four replicas, whereas the former only need it from three.

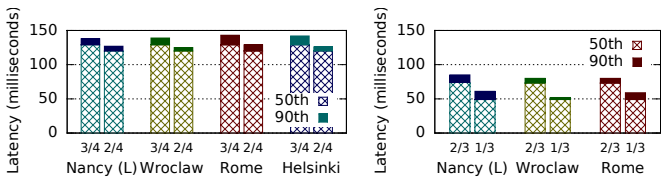
[2] http://aws.amazon.com/ec2/.

C. Number of Replies

In this experiment we intended to observe how the amount of replies required by clients affects the operation latency. By default, BFT-SMaRt clients wait for $\lceil \frac{n+f+1}{2} \rceil$ (BFT) or $\lceil \frac{n+1}{2} \rceil$ (CFT) replies from replicas in order to ensure linearizability. However, this number of replies is required due to the use of read-only executions [8]: if this optimization were not supported, $f+1$ matching replies (BFT) or 1 (CFT) replies would suffice.

Setting: We created a variant of a BFT-SMaRt client that waited only for $f+1$ (BFT) or 1 (CFT) replies, thus satisfying only sequential consistency (similarly to Zookeeper [17]) if the read-only optimization is employed. This experiment was deployed on hosts located in Nancy (leader), Wroclaw, Helsinki and Rome. The modified clients waited for two out of four replica replies (or one out of three in CFT), while the original version waited for the usual three out of four (two out of three in CFT). CFT experiments did not require the Helsinki's host.

Results: The values for the median and 90th percentile latency for each client are shown in Figure 4. It can be observed that both the original and modified protocols present very similar performance in BFT mode. On the other hand, the optimization was quite effective in the CFT mode. For the 90th percentile, this optimization showed an improvement from 8% to 11% in BFT mode and from 26% to 36% in CFT mode.



(a) BFT Mode. (b) CFT Mode.

Fig. 4. Client latencies' 50th/90th percentile for different numbers of replies.

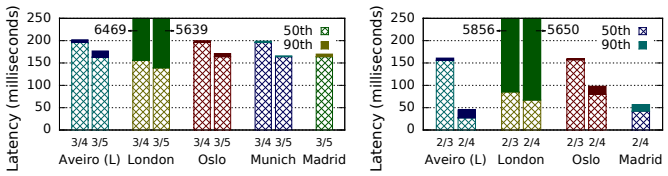
Main conclusion: The lower latency displayed when the protocol requires less replies was to be expected, but such reduction was more significant in CFT mode. This can be explained by the fact that the BFT mode employed one more replica and required one more reply when compared to CFT.

D. Quorum Size

This experiment is motivated by the works of Gifford [15] and Pâris [25], which use voting schemes with additional hosts to improve the availability of quorum protocols. As described in §II, BFT-SMaRt's clients and replicas always wait for $\lceil \frac{n+f+1}{2} \rceil$ messages from other replicas to advance to the next communication step (or $\lceil \frac{n+1}{2} \rceil$ in CFT mode). More precisely, BFT-SMaRt waits for *dissemination Byzantine quorums* [22] if operating in BFT mode and majority quorums [14] if operating in CFT mode. During this experiment, we enable the system to make progress without waiting for the aforementioned quorum types if spare replicas are present. Notice that this optimization, which is not employed in any SMR protocol, might lead to safety violations (discussed below).

Setting: We modified BFT-SMaRt to make replicas wait for only $2f+1$ (resp. $f+1$) messages in each phase of the BFT (resp. CFT) protocol, independently from the total number of replicas n.[3] This experiment was deployed on hosts located in Aveiro (leader), London, Oslo, Munich and Madrid. The original BFT-SMaRt was configured to execute across four replicas (three in CFT mode) and the modified version was configured to execute in five (four in CFT mode). The extra replica needed for executing the modified version was placed in Madrid, both for BFT and CFT mode. Experiments for CFT mode did not require the use of Munich's host. Since the modified version waits only for three out of five (3/5) messages (or 2/4 messages in CFT mode), both versions of BFT-SMaRt will wait for the same number of messages, even though the optimized versions use one additional replica.

Results: The values for the median and 90th percentile latency for each client are shown in Figure 5. The results show that the modified protocols – which used one extra replica – exhibited lower latency than the original protocols. This difference is more discernible in the CFT mode for two reasons. First, the ratio between the quorum size and the number of replicas (2/4) is smaller than the BFT case (3/5). Second, it did not use London's host (which observed a much worse 90th percentile latency than others). It can be observed that in the 90th percentile, the optimizations showed an improvement of 12%-17% in the BFT mode and 4%-72% in CFT mode, depending on the location of clients.



(a) BFT Mode. (b) CFT Mode.

Fig. 5. Client latencies' 50th/90th percentile with different quorum sizes.

Main conclusion: The modified version BFT-SMaRt was able to experience lower latency because it was given more choice: since both versions still waited for the same number of messages in each communication step, the slowest replica was replaced by the extra replica hosted in Madrid, thus decreasing the observed latency of the modified version. In normal protocols this benefit would be smaller, since the quorum size would normally increase with n.

Even though the use of additional replicas decreases the protocol latency, this kind of optimization cannot be directly applied to existing protocols without impairing their correctness. Limiting the amount of messages to $2f+1$ (or $f+1$) regardless of the total number of replicas available n does not guarantee the formation of intersecting quorums, which are required to ensure safety in both BFT and CFT modes [8], [21]. For example, in the CFT mode, our setup of $n = 4$ and $f = 1$ did not ensure majority quorums, which could had lead to safety violations. In order to preserve correctness, it is necessary to force any combination of $2f+1$ (or $f+1$) replicas to intersect in at least one correct server. In §IV-B we present mechanism that ensures this property and allows the use of this optimization in SMR systems.

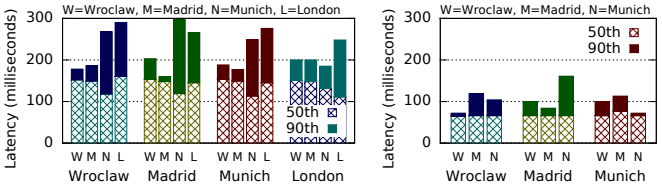
[3] If the original BFT-SMaRt were deployed in five hosts, the quorums would be comprised of four hosts (in the case of BFT mode).

E. Leader Location

The goal of our last experiment is to observe how much the leader's location can affect the client latency. This experiment is motivated by the fact that Mencius [23], EBAWA [29] and EPaxos [24] use different techniques to make each client use its closest (or co-located) replica as the leader for its operations. The rationale behind these techniques is to make client-leader communication faster, bringing down the end-to-end SMR latency (see Figure 1).

Setting: We deployed BFT-SMaRt in PlanetLab and conducted several experiments considering different replicas assuming the role of the leader. The hosts used were located in Wroclaw, Madrid, Munich and London (not used in CFT mode). Moreover, the experiment was repeated across Amazon EC2, using replicas in Ireland, Oregon, São Paulo and Sydney regions (Sydney was only used in BFT mode).

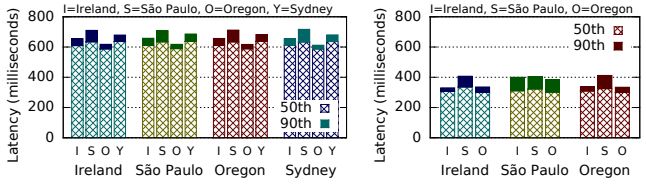
Results: Before launching this experiment, we expected that, for any client, its latency would be the lowest when its co-located replica were the protocol's leader. However, as seen in Figure 6, the median and 90th percentiles of the latency observed by the different clients do not change significantly when the leader location changes. In particular, the 90th percentile latency is, in general, lower when the leader was either in Madrid (BFT) or Wroclaw (CFT).



(a) BFT Mode. (b) CFT Mode.

Fig. 6. Client latencies' 50th/90th percentile when the leader is placed across PlanetLab hosts.

Since these results appeared to contradict the intuition of [23], [24], [29], we repeated this experiment in Amazon EC2, to find if this phenomenon is due to our choice of testbed. Figure 7 shows the results observed in each Amazon EC2 region. As with the PlanetLab results, the latency observed by the different clients do not present any significant change as we change the leader location. However, having the leader in Oregon results in a lower 90th percentile for all clients, both for BFT and CFT modes.



(a) BFT Mode. (b) CFT Mode.

Fig. 7. Client latencies' 50th/90th percentile when the leader is placed across Amazon EC2 regions.

Main conclusion: Since the obtained results depict a similar trend in the two different testbeds, we can assert that co-locating clients with the leader does not necessarily improve the latency of replicated state machines. On the other hand, placing the leader in the host with better connectivity with the remaining replicas can yield more consistent improvements. More precisely, the benefit of reaching the leader faster is not as important as hosting the leader in the replica with faster links with others.

F. Discussion

The results presented in §III-B indicate that, as expected, bypassing communication steps reduces client latency in BFT SMR protocols. However, even though read-only (resp. speculative) executions are up to 63% (resp. 35%) faster than standard executions, the benefits of tentative and fast executions are not so impressive: about 20% and 10%, respectively. The difference, as explained before, is due to the fact that fast executions requires larger quorums than tentative execution, which requires waiting for more messages (that can be slow in an heterogeneous environment such as a WAN). In the end, tentative execution matches the theoretically expected benefits: by avoiding 20% of the communication steps (see Figure 2), we did reduce latency to approximately 20%.

The results of §III-C and §III-D shows that decreasing the ratio between the number of expected messages and the total number of replicas can decrease latency significantly, especially for CFT replication. More specifically, clients that wait less replies had a 90th percentile latency improvement of up to 36% (resp. 11%) in CFT (resp. BFT) mode; and adding more replicas to the system while maintaining the same quorum size brings improvements of up to 72% (resp. 17%) in CFT (resp. BFT) mode. These results are mainly due to the performance-heterogeneity of hosts and links in real wide area networks: if the latency between all replicas were similar and network delivery variance were small, the observed improvements would be much more modest. Furthermore, they are in accordance with other studies showing that using smaller quorums may bring better latency than decreasing the number of communication steps (e.g., [18]).

The results of §III-E indicates that having the leader close to a client will not significantly reduce the SMR latency for this client. This result is unexpected since several protocols implement mechanisms such as rotating coordinator [23], [29] and multiple proposers [24] to make each client submit its requests to the closest replica. We found two main explanations for this apparent contradiction. First, the heterogeneity of real environments such as PlanetLab and Amazon EC2 make optimizations for reducing latency less effective. In fact, the authors of Mencius acknowledge that the protocol achieves lower latency than Paxos only in networks with small latency variances [23]. Second, in CFT mode, BFT-SMaRt clients wait for replies from a majority of replicas to ensure linearizability due to the use of the read-only optimization. EPaxos, Mencius and Paxos clients wait only for a single reply from the leader. This means that client-leader co-location in these protocols potentially reduce the latency in two communication steps, while in BFT-SMaRt this reduction is in only one (clients still need to wait for at least one additional reply). Consequently, having a client co-located with the leader should decrease the number of communication steps 25% in CFT mode and 20% in BFT mode, while in Mencius and EPaxos such theoretical improvement can reach 50%. Moreover, its worth to point out that these benefits appear only in favorable

conditions. For example, EPaxos presents almost the same latency of Paxos when under high request interference [24].

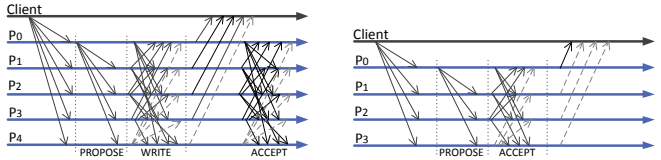
As a final remark, it is worth noting that our results show that having a leader in a well-connected replica brings, in general, more benefits than having clients co-located with leaders. For instance, we observed that latency was usually lower when the leader replica was hosted in Madrid, rather than when the leader replica was placed in the same location as a particular client. In the same line, adding faster replicas to the system may significantly improve latency, as shown in §III-D. For example, the addition of Madrid to the set of replicas decreased the 90th percentile latency in Oslo and Aveiro by 39% and 72%, respectively (CFT mode). More generally, these results highlight the fact that *not all replicas are the same in geo-replication* (at least in the considered testbeds) and that both the leader location and quorum formation rules must take into account the characteristics of the sites being used.

IV. The WHEAT Protocol

This section describes WHEAT, a WAN-optimized SMR protocol implemented on top of BFT-SMaRt. We start by discussing the WAN optimizations employed in the protocol and then introduce two novel vote assignment schemes to use smaller quorums without endangering the correctness of the protocol. We conclude the section with an evaluation of WHEAT in Amazon EC2.

A. Deriving the protocol

WHEAT employs the optimizations that were most effective in improving the latency of SMR in WANs. The selected optimizations (discussed below) reduce the number of communication steps, the number of replies that clients wait and the ratio between the quorum size and the total number of replicas. Since the results of client-leader co-location were not so expressive, and given that its implementation would require substantial changes in the base SMR protocol (which is already complex enough [6], [9]), we rejected this optimization and followed the fixed leader approach. As with BFT-SMaRt, WHEAT can be used in BFT or CFT modes, implementing the message patterns illustrated in Figure 8.



(a) BFT message pattern. (b) CFT message pattern.

Fig. 8. WHEAT's message pattern for $f = 1$ and one additional replica.

Reducing the number of communication steps: In BFT mode, WHEAT employs the read-only and tentative execution optimizations introduced in PBFT [8]. The reason to support tentative executions instead of fast or speculative executions is as follows: (1) during our experiments, tentative executions displayed slightly better latency than fast executions (i.e., they had a lower 90th percentile); (2) speculative executions are useful in environments were the network is predictable and stable, which we cannot expect in many geo-distributed settings. If such conditions are not met by the network (i.e, not delivering replies from all replicas within the required time window), clients need to trigger the commit phase and force the protocol to execute five communications steps [19]; and (3) tentative executions do not require modifications to the synchronization phase of BFT-SMaRt. Fast executions would required modifications to account for cases where a value was decided solely with $\lceil \frac{n+3f+1}{2} \rceil$ WRITE messages, whereas the rollback operation can be triggered using the state transfer protocol already implemented in BFT-SMaRt [6]. Furthermore, usage of speculative executions would demand the complete re-implementation of the original protocol, to account for the several corner cases necessary to preserve correctness under this type of executions, such as the aforementioned commit phase. Another advantage of tentative executions is that ACCEPT messages can be piggybacked in the next PROPOSE or WRITE messages, similarly to PBFT [8].

Reducing the number of replies a client waits: In BFT mode, the use of read-only and tentative executions lead WHEAT clients to always gather responses from a Byzantine quorum of replicas, i.e., at least $\lceil \frac{n+f+1}{2} \rceil$ replies. This means that it is impossible to enforce the optimization evaluated in §III-C without giving up linearizability [16]. However, single-reply read-only executions can still be used in the CFT mode as long as clients always contact the leader replica.[4] Consequently, in CFT mode WHEAT clients only need to wait for one reply (from any replica during write operations and from the leader during read-only operations).

Reducing the ratio between the quorum size and the number of replicas: As observed in §III-D, it is possible to significantly decrease latency by adding more replicas to the system, as long as the quorums used in the protocol remain with the same size. Both the Byzantine and crash variants of WHEAT are designed to exploit this phenomenon by modifying the quorum requirements of the protocol. However, to avoid breaking the safety properties of traditional SMR protocols (e.g., [8], [21], [29]), we need to introduce a mechanism to secure the formation of intersecting quorums of variable size. In the next section we introduce a voting scheme that preserves this requirement.

B. Vote Assignment Schemes

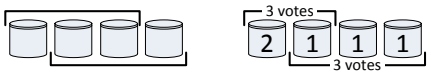
Our novel voting assignment schemes integrate the classical ideas of weighted replication [14], [15], [25] to SMR protocols. The goal is to extend quorum-based SMR protocols to (1) rely primarily on the fastest replicas present in the system, and (2) preserve its original safety and liveness properties.

The most important guarantees that quorum-based protocols need to preserve are (1) all possible quorums overlap in some correct replica and (2) even with up to f failed replicas, there is always some quorum available in the system. In CFT protocols like Paxos [21], quorums must overlap in at least one replica. Such intersection is enforced by accessing a simple majority of replicas during each communication step of a protocol. More specifically, protocols access $\lceil \frac{n+1}{2} \rceil$ replicas out of $n \geq 2f + 1$. BFT protocols like PBFT [8], on the other hand, usually employ disseminating Byzantine quorums [22] with at least $f + 1$ replicas in the intersection. In this case,

[4] It is also necessary to use temporal leases on the client, since the leader can be demoted at any point.

protocols access $\lceil \frac{n+f+1}{2} \rceil$ replicas out of $n \geq 3f + 1$. With this strategy, adding a single extra replica to the system results in higher latency, since any possible quorum becomes larger in size – unlike the weighted quorums strategy we present below.

The fundamental observation that we make, is that accessing a majority of replicas guarantees the aforementioned intersection, but that this is not the only way to secure such intersection. More specifically, if n is greater than $2f + 1$ (in CFT mode), it is possible to distribute weights across replicas in such way that a majority is not always required to (correctly) make progress. As an example, consider the quorums illustrated in Figure 9 (with one extra replica in the system). Whereas in Figure 9(a) the intersection is obtained by strictly accessing a majority of replicas, in Figure 9(b) we see that we can still obtain an intersection with a variable number of replicas (since we can obtain a sum of 3 votes by either accessing 2 or 3 replicas). In particular, if the replica with weight 2 is successfully probed, the protocol can finish a communication step with a quorum comprised by only half of the replicas. Otherwise, a quorum comprised by all replicas with weight 1 is necessary to make progress. Notice that for this distribution to be effective, it is necessary to attribute weight 2 to the fastest replica in the system.



(a) Classical (always a majority of replicas).

(b) Weighted (from $f + 1$ to $n - f$ replicas).

Fig. 9. Quorum formation when $f = 1$ and $n = 4$ (CFT mode).

We now generalize the weight distribution proposed in Figure 9(b) to account for other values of f. The objective is to assign certain numbers of votes (i.e., weights) to each replica in accordance with their connectivity/performance. This vote assignment must be done carefully to ensure that minimal quorums composed by faster replicas will be used under normal conditions (i.e., when the faster replicas are indeed faster) and larger, yet available quorums can be used to ensure that up to f faulty replicas are tolerated (despite their weights).

Let Q_v be the minimum number of votes that a quorum of replicas must hold to guarantee that quorums overlap by at least one correct replica. A quorum is said to be *safe and minimal* (or just minimal) if it is comprised by only $f + 1$ replicas that together hold Q_v votes. This quorum size is minimal because if f or less replicas were considered a quorum, other intersecting quorums would require *more than* $n - f$ replicas. These quorums will not be available when there are f faulty replicas in the system. This means that having quorums with less than $f + 1$ replicas implies giving up consistency or availability, as described in classical quorum definitions [22]. In a BFT system, for the same reasons, a minimal quorum must be comprised of $2f + 1$ replicas.

Using the above definitions, we consider vote distribution schemes that satisfy the following properties:

- **Safe minimality:** There exists at least one minimal quorum in the system.
- **Availability:** There is always a quorum available in the system that holds Q_v votes.
- **Consistency:** All quorums that hold Q_v votes intersect by at least one correct replica.

In the following, we describe vote assignment schemes for CFT and BFT modes that satisfy these properties.

CFT vote distribution: To calculate the vote distribution under CFT mode, we start by introducing the parameter Δ, which represents the number of extra replicas available in the system. Thus, n can be calculated using Δ as follows:

$$n = 2f + 1 + \Delta \tag{1}$$

We now introduce two additional variables. N_v represents the sum of the number of votes $\sum V_i$ that are attributed to each replica i. F_v is the maximum number of votes that can be dismissed in the system. Having these parameters, we can apply the standard quorum rules to the votes instead of the replicas. Hence, N_v is calculated as follows:

$$N_v = \sum V_i = 2F_v + 1 \tag{2}$$

As an example, consider Figure 9(b): the sum of all votes adds up to 5, which represents an *abstract quorum system* comprised by 5 hosts capable of withstanding 2 faults. Therefore, for this case, $N_v = 5$ and $F_v = 2$.

Since Δ and f are the input parameters, we need to (1) find a relation between Δ and f and values for N_v, F_v and V_i; (2) use those variables to force the emergence of *replica quorums* that intersect by one replica. More precisely, votes must be distributed in such a way that once $Q_v = F_v + 1$ votes are gathered, quorums always overlap by at least one correct replica.

If we assume that only two possible values can be assigned to replicas (e.g., a binary vote distribution), as in Figure 9(b), we can introduce variables V_{max} and V_{min}. However, we need to find how many replicas are assigned V_{max} and V_{min}. Let u be the number of replicas holding V_{max} votes and, consequently, $n-u$ the number of replicas holding V_{min}. Since the sum of all votes must be equal to N_v, we have:

$$N_v = 2F_v + 1 = uV_{max} + (n - u)V_{min} \tag{3}$$

We see that in this example, $V_{max} = 2$, $V_{min} = 1$ and $Q_v = F_v + 1 = 3$. We can observe two cases where 3 votes can be obtained: either by (1) accessing the single V_{max} replica and one of the V_{min} replicas, or (2) accessing all V_{min} replicas. Notice that in both cases, the same number of votes is dismissed, but not the same number of replicas; in case (1) two replicas are ignored, but in case (2) only one single replica is left unprobed. Also note that the number of votes dismissed is 2 – which happens to be the value of F_v (as we pointed out previously). This indicates that F_v has a direct relation to V_{max} and V_{min}. Given this observation, we generalize Figure 9(b) scenario to represent any Δ and f:

$$F_v = (\Delta + f)V_{min} = fV_{max} \tag{4}$$

We derive the relation between V_{max} and V_{min} as follows:

$$V_{max} = \frac{(\Delta + f)}{f} V_{min} \tag{5}$$

If we assume $V_{min} = 1$, equations (4) and (5) become:

$$F_v = \Delta + f \tag{6}$$

$$V_{max} = \frac{\Delta + f}{f} = 1 + \frac{\Delta}{f} \tag{7}$$

Having now more refined formulas for F_v, V_{max} and V_{min}, we can return to equation (3) and obtain the value of u:

$$2(\Delta + f) + 1 = u(1 + \frac{\Delta}{f}) + (n - u) \Rightarrow u = f \tag{8}$$

Knowing that $u = f$, still by equation (3), there must be f replicas holding V_{max} votes and $n - f$ replicas holding 1 vote (since $V_{min} = 1$). We thus have our CFT vote assignment scheme: equations (6) and (7) give us the values for F_v and V_{max} respectively, all in function of Δ and f.

The main benefit of this scheme is that if all the f replicas holding V_{max} are probed faster than any other, then just one of the $\Delta + f + 1$ other replicas holding V_{min} votes will be disregarded (like the two-replica quorum of Figure 9(b)). However, in the worst case, if f replicas holding V_{max} votes fail (or are slow), then all replicas with V_{min} votes will be accessed instead (as the three-replica quorum of Figure 9(b)).

CFT proof of correctness: In the following we briefly outline the proof that our vote assignment scheme satisfies the three properties described before. The full proof is available in the extended version of this paper [28].

Safe minimality: Let S_{max} to be the subset of f replicas that hold V_{max} votes each. By equation (4) these f replicas will add up to F_v votes. A safe and minimal quorum can be built using S_{max} plus one additional replica holding $V_{min} = 1$ votes, making a quorum with $Q_v = F_v + 1$ votes. □

Availability: Let S_{min} be the subset of $n - f$ replicas holding V_{min} votes each. In the worst case (maximum number of votes lost due to faults), when all the f replicas holding V_{max} fail, there will be still the $n - f = \Delta + f + 1$ replicas from S_{min}. According to equation (6), $\Delta + f$ account already for F_v votes. With the additional replica from S_{min}, we reach the required $Q_v = F_v + 1$ votes to form a quorum, even with the fV_{max} votes lost. Furthermore, any other combination of V_{max} and V_{min} replicas will always contain at least Q_v votes (as long there are at least $f+1$ replicas), since they will either be a minimal quorum (as proved before), a S_{min} quorum, or a hybrid of both (which will result in $Q_v \geq F_v + 1$). □

Consistency: Any quorum overlaps by at least one correct replica for the following reason: since a minimal quorum contains Q_v votes (as proved before), it must contain at least one V_{min} replica, which in turn must be a member of the S_{min} quorum (which also contains Q_v votes, as proved before). Since any other allowed combination of V_{max} and V_{min} replicas will either be a superset of a minimal or S_{min} quorum, they will also intersect by one replica (or more). □

BFT assignment: The reasoning here is similar to the CFT scheme, but with the following differences. First, equations (1) and (2) become $n = 3f + 1 + \Delta$ and $N_v = \sum V_i = 3F_v + 1$, respectively. These equations still lead to the same values of F_v and V_{max}, but u becomes $2f$ instead of f. This forces the system to have $2f$ replicas holding V_{max} and $\Delta + f + 1$ replicas holding one vote (V_{min}). Moreover, it is necessary to gather $2F_v + 1$ votes on each quorum, which makes $Q_v = 2F_v + 1$. Finally, a minimal quorum must be comprised by $2f + 1$ replicas instead of $f + 1$. The proof of correctness for the BFT voting assignment scheme and refer the reader to the extended version of the paper [28].

Additional resilience benefits: Our voting assignment schemes present two additional benefits in terms of resilience. First, it allows the system to tolerate more than f *crash faults* in certain scenarios. For instance, in Figure 9(b), two of the V_{min} replicas could fail by crash and the protocol would still make progress without violating safety. However, this is not the case if one of two failed replicas holds V_{max} votes. Secondly, when any of the fastest replicas is detected as slow or unavailable, BFT-SMART's reconfiguration protocol [6] can be used to redistribute votes, so that other replicas take the place of the ones that are no longer the fastest. Notice that this approach is better than using BFT-SMART's reconfiguration protocol to replace unavailable replicas. Such replacement would require a state transfer, which can be a slow operation for large state sizes and limited wide-area links. For example, a 4GB-state will take more than fifty minutes to be transferred in a 10 Mbps network (which is better than most links between EC2 regions). With our approach, the extra replicas are already active and up-to-date in the system, so the reconfiguration takes approximately the time to execute a "normal" SMR operation. Finally, it is worth mentioning that in the event that the systems experiences a period of high load, it is possible that the minimal quorum becomes overloaded and unable to reply faster than other quorums, thus forcing the system to make progress with different quorums. Nonetheless, any SMR protocol based on quorum systems is subject to this issue.

C. Implementation and Evaluation

We implemented WHEAT by extending BFT-SMART for supporting the chosen optimizations (§IV-A) and considering replicas with different number of votes (§IV-B). Most of the modifications to the code took into account the vote assignment schemes that calculate the quorums used in the protocol.

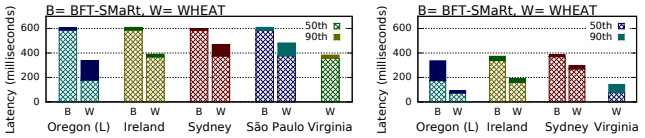
We evaluated WHEAT by running a set of experiments in Amazon EC2 and comparing the results with the original BFT-SMART system. As in the EC2 experiments reported in §III-E, we use sites on Ireland, Oregon, Sydney and São Paulo (only in BFT mode) for BFT-SMART using also Virginia as the additional replica of WHEAT. This means that the original version of BFT-SMART employed 4 replicas in BFT mode (resp. 3 in CFT mode) whereas WHEAT employed 5 replicas in BFT mode (resp. 4 in CFT mode), with two of these replicas in North America. In the BFT mode, the following parameters were employed (obtained through the voting schemes described previously): $N_v = 7$, $F_v = 2$, $V_{max} = 2$ for the replicas in Oregon and Virginia. In the CFT mode, the configuration was $N_v = 5$, $F_v = 2$, $V_{max} = 2$ for the replica in Virginia. We attributed the V_{max} values to the these sites because they were the ones with better connectivity to others, as shown in Table II.

The median and 90th percentile latencies for each client location and protocol is presented in Figure 10. By employing the selected optimizations (§IV-A) and using an additional replica in Virginia without increasing the quorum requirements

Sites	Ireland	São Paulo	Oregon	Sydney	Virginia
Ireland	0	211 ± 10	171 ± 11	340 ± 11	88 ± 10
São Paulo	208 ± 14	0	217 ± 19	359 ± 4	123 ± 3
Oregon	171 ± 14	217 ± 11	0	205 ± 7	70 ± 12
Sydney	336 ± 26	359 ± 4	205 ± 10	0	255 ± 12
Virginia	88 ± 10	123 ± 4	71 ± 13	256 ± 5	0

TABLE II. AVERAGE *roundtrip* LATENCY AND STANDARD DEVIATION (MILLISECONDS) BETWEEN AMAZON EC2 REGIONS AS MEASURED DURING A 24 HOUR-PERIOD.

(i.e., three and two replicas for BFT and CFT, respectively), WHEAT achieves, when compared to BFT-SMART, a 90th percentile latency improvement between 21% and 44% (BFT) and between 23% and 73% (CFT). Interestingly, the client in the leader region (Oregon) observed significant improvements, with median latency values matching the roundtrip times between Oregon and Ireland (BFT mode) or Virginia (CFT mode). This is a consequence of the fact that this client is co-located with the leader *in the most well-connected site of the system*. Moreover, upon considering all clients' measurements together, we found that WHEAT improved the global 90th percentile by 35% (BFT) and 28% (CFT). The global median improvement is even higher: 37% in BFT and 56% in CFT.



(a) BFT Mode. (b) CFT Mode.

Fig. 10. 50th/90th percentile latencies observed by BFT-SMART and WHEAT clients in different regions of Amazon EC2.

The improvements shown in this experiment should be taken with a bit of salt since they may be due the use of an additional site with a good roundtrip latency with other replicas (see Table II). If the new replica used in WHEAT were added on an hypothetical Amazon EC2 region "moon" (instead of Virginia), with a higher roundtrip latency with all other sites, the WHEAT results would be less impressive since the faster quorums will be the same of BFT-SMART. The only benefits will be due to the other optimizations (tentative executions for BFT and single-reply for CFT) implemented in the system. Nonetheless, our results illustrate the fact that in a real geo-replication setup there are significant benefits in assign different values to different replicas. Furthermore, even with the required algorithmic support, it is important to choose the location of the spare replicas employed in WHEAT, to ensure the minimal quorums will bring significant benefits.

A note on throughput: WHEAT aims to improve geo-replication latency, and thus all of its optimizations target this performance metric. However, the fact it uses Δ more replicas than BFT-SMART, implies it might achieve a slight lower peak throughput than the original system. This happens because more replicas lead to more message transmissions, which results in higher CPU and network bandwidth utilization. More precisely, each consensus instance on BFT-SMART requires the exchange of $3f + 18f^2$ (resp. $2f + 4f^2$) messages in BFT mode (resp. CFT mode), whereas in WHEAT it requires $3f + \Delta + 2(3f + \Delta)^2$ (resp. $2f + \Delta + (2f + \Delta)^2$) message exchanges. Although undesirable, this drawback will only affect a saturated system, which is rarely the case in production environments. Moreover, as discussed in §III-A, throughput can be improved by increasing CPU and network resources, while latency can only be addressed by better protocols.

V. RELATED WORK

The criticality of modern internet-scale services have created the need for geo-replication protocols for disaster tolerance, including whole-datacenter failures (e.g., [10]). Following this trend, several works proposed strongly consistent WAN SMR protocols like [1], [23], [24], [29].

Steward [1] is a hierarchical Byzantine fault-tolerant protocol for geographically dispersed multi-site systems. It employs a hybrid algorithm in the sense that it runs a BFT agreement protocol within each site, and a lightweight, crash fault-tolerant protocol across sites. Even though Steward is able to perform well in WANs (when compared with PBFT [8]), that comes at the cost of a very complex protocol (over ten specialized algorithms that run within and among sites) that demands plenty of resources (e.g., each site requires at least 4 replicas). Although we advocate the use of additional replicas for improving latency in WHEAT, our protocol is not radically different from "normal" protocols, requiring no specialized subprotocols or a specific number of replicas on a site.

Mencius [23] is an SMR protocol derived from Paxos [21] also optimized to execute in WANs. Like Paxos, it can survive up to f crashed replicas out of at least $2f + 1$. Replicas take turns as the leader and propose client requests in their turns. Clients send requests to the replicas in their sites, which are submitted for ordering when the replicas become the leader. EBAWA [29] is a Byzantine-resilient SMR protocol optimized for WANs. It considers a hybrid fault model in which each replica uses a local trusted/trustworthy service (that cannot be compromised) to provide tolerance to up to f Byzantine faults using only $2f + 1$ replicas. Similarly to Mencius, it uses a rotating leader to allow clients to send requests to the replicas that are close to them. Egalitarian Paxos (EPaxos) [24] is a recent SMR protocol also derived from Paxos and designed to execute in WANs. Unlike most SMR protocols inspired by Paxos, EPaxos does not rely on a single designated leader for ordering operations. Instead, it enables clients to choose which replica should propose their operations, and employs a mechanism which resolves conflicts between interfering operations. Differently from Mencius, EBAWA and EPaxos, WHEAT does not employ any mechanism to make clients use their closer replicas as leaders/coordinators/proposers. Our decision to avoid this optimization comes from observing that having a leader in the same site as the client gives less benefits in terms of latency than using the fastest replica as the leader.

Weighted replication was originally proposed by Gifford [15], and then revisited by Garcia-Molina [14] and Pâris [25]. While Gifford made all hosts hold a copy of the state with distinct voting weights, Pâris made a distinction between hosts that hold a copy of the state and hosts that do not hold such copy, but still participate in the voting process (thus acting solely as witnesses). More recent works confirmed the usefulness of these ideas also for performance by showing that adding few servers to a group of replicas can significantly improve the access latency of majority quorums [5], and the same kind of technique is being used in practical systems to improve tolerance to slow servers [11]. By contrast,

Garcia-Molina addresses the idea of weighted replication in [14] for *coterie systems*, which later evolved into the classic quorum systems without including vote distribution. Unlike our approach, none of these works target geo-replication: [25] and [14] are strictly theoretical contributions and [15] considers a local datacenter. To the best of our knowledge, we present the first vote assignment scheme that unpacks a weight distribution in function of the expected number of faults and the amount of spare replicas available in the system.

There are empirical studies which evaluate the availability of quorum systems (e.g., [2], [5]), the latency of distributed algorithms over the internet (e.g., [4]) and the performance of different total order broadcast protocols – a fundamental building block for SMR – over a WAN (e.g., [3], [13]). Our experiments have a different goal: instead of evaluating the performance of distinct protocols, we compare geo-replication-related optimizations employed by different protocols, but implemented in the same codebase, to validate the effectiveness of these optimizations in real WANs.

VI. Conclusion

In this paper we revisited some optimizations proposed in the literature for improving the latency of SMR protocols in wide area networks. More concretely, we implemented such optimizations in an open-source SMR library and compared its latency with a non-optimized version in the PlanetLab testbed and Amazon EC2 cloud to assess which of these optimizations bring significant benefits. Our results indicated that removing communication steps and demanding less replies from replicas lead to latency reductions of up to 20%, depending on the hosts and fault model. Surprisingly, using the closer replica as the leader held less benefits than what was expected. These results guided our design for WHEAT, an SMR protocol optimized for geo-replication that can be configured either for crash-only or Byzantine fault tolerance. WHEAT was implemented by extending BFT-SMaRt with the optimizations we observed as most effective and implementing novel vote assignment schemes for efficient quorum usage. Our evaluation showed gains of up to 56% for certain configurations, when compared with the unmodified BFT-SMaRt.

Acknowledgments: This work was partially supported by the EC through projects SUPERCLOUDS (H2020-ICT-2014-1) and by FCT via the Multiannual Program (LaSIGE).

References

[1] Y. Amir, C. Danilov, D. Dolev, J. Kirsch, J. Lane, C. Nita-Rotaru, J. Olsen, and D. Zage. Steward: Scaling Byzantine fault-tolerant replication to wide area networks. *IEEE Transactions on Dependable and Secure Computing*, 7(1):80–93, 2010.

[2] Y. Amir and A. Wool. Evaluating quorum systems over the internet. In *Proceedings of the 26th Annual International Symposium on Fault-Tolerant Computing*, 1996.

[3] T. Anker, D. Dolev, G. Greenman, and I. Shnayderman. Evaluating total order algorithms in WAN. In *In Proceedings of the International Workshop on Large-Scale Group Communication*, 2003.

[4] O. Bakr and I. Keidar. Evaluating the running time of a communication round over the internet. In *Proceedings of the 21st Annual Symposium on Principles of Distributed Computing*, 2002.

[5] O. Bakr and I. Keidar. On the performance of quorum replication on the internet. Technical Report UCB/EECS-2008-141, EECS Department, University of California, Berkeley, 2008.

[6] A. Bessani, J. Sousa, and E. Alchieri. State machine replication for the masses with BFT-SMART. In *Proc. of the 44th Annual IEEE/IFIP International Conf. on Dependable Systems and Networks*, 2014.

[7] M. Burrows. The chubby lock service for loosely-coupled distributed systems. In *Proceedings of the 7th Symposium on Operating Systems Design and Implementation*, 2006.

[8] M. Castro and B. Liskov. Practical Byzantine fault tolerance and proactive recovery. *ACM Transactions Computer Systems*, 20(4):398–461, 2002.

[9] T. Chandra, R. Griesemer, and J. Redstone. Paxos made live - an engineering perspective (2006 invited talk). In *Proceedings of the 26th ACM Symposium on Principles of Distributed Computing*, 2007.

[10] J. C. Corbett et. al. Spanner: Google's globally distributed database. *ACM Transactions on Computer Systems*, 31(3):8:1–8:22, 2013.

[11] J. Dean and L. A. Barroso. The tail at scale. *Communications of the ACM*, 56:74–80, 2013.

[12] E. Duarte, T. Garrett, L. Bona, R. Carmo, and A. Züge. Finding stable cliques of planetlab nodes. In *Proceedings of the 40th IEEE/IFIP International Conference on Dependable Systems and Networks*, 2010.

[13] R. Ekwall and A. Schiper. Modeling and validating the performance of atomic broadcast algorithms in high latency networks. In *Euro-Par 2007 Parallel Processing*, 2007.

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

[15] D. Gifford. Weighted voting for replicated data. In *Proceedings of the 7th ACM Symposium on Operating Systems Principles*, 1979.

[16] M. Herlihy and J. M. Wing. Linearizability: A correctness condition for concurrent objects. *ACM Transactions on Programing Languages and Systems*, 12(3):463–492, 1990.

[17] P. Hunt, M. Konar, F. Junqueira, and B. Reed. Zookeeper: Wait-free coordination for internet-scale services. In *Proceedings of the Usenix Annual Technical Conference*, 2010.

[18] F. Junqueira, Y. Mao, and K. Marzullo. Classic Paxos vs Fast Paxos: Caveat emptor. In *Proceedings of the Workshop on Hot Topics in System Dependability*, 2007.

[19] R. Kotla, L. Alvisi, M. Dahlin, A. Clement, and E. Wong. Zyzzyva: Speculative Byzantine fault tolerance. *ACM Transactions on Computer Systems*, 27(4):45–58, 2009.

[20] L. Lamport. Time, clocks, and the ordering of events in a distributed system. *Communications of the ACM*, 21(7):558–565, 1978.

[21] L. Lamport. The part-time parliament. *ACM Transactions Computer Systems*, 16(2):133–169, 1998.

[22] D. Malkhi and M. Reiter. Byzantine quorum systems. *Distributed Computing*, 11(4):203–213, 1998.

[23] Y. Mao, F. P. Junqueira, and K. Marzullo. Mencius: building efficient replicated state machines for WANs. In *Proceedings of the 8th USENIX Conference on Operating systems design and implementation*, 2008.

[24] I. Moraru, D. G. Andersen, and M. Kaminsky. There is more consensus in egalitarian parliaments. In *Proceedings of 24th ACM Symposium on Operating Systems Principles*, 2013.

[25] J. Pâris. Voting with witnesses: A consistency scheme for replicated files. In *In Proceedings of the 6th International Conference on Distributed Computing Systems*, 1986.

[26] F. Schneider. Implementing fault-tolerant service using the state machine aproach: A tutorial. *ACM Computing Surveys*, 22(4):299–319, 1990.

[27] J. Sousa and A. Bessani. From Byzantine consensus to BFT state machine replication: A latency-optimal transformation. In *Proceedings of the 9th European Dependable Computing Conference*, 2012.

[28] J. Sousa and A. Bessani. Separating the WHEAT from the chaff: An empirical design for geo-replicated state machines (extended version). Technical Report TR 2015-04, Department of Informatics, Faculty of Sciences of the University of Lisboa, July 2015.

[29] G. Veronese, M. Correia, A. Bessani, and L. C. Lung. EBAWA: Efficient Byzantine agreement for wide-area networks. In *12th IEEE International High Assurance Systems Engineering Symposium*, 2010.

[30] P. Zielinski. Paxos at war. Technical Report UCAM-CL-TR-593, University of Cambridge, Computer Laboratory, 2004.