Latency-Aware Leader Selection for Geo-Replicated Byzantine Fault-Tolerant Systems

Michael Eischer and Tobias Distler
Friedrich-Alexander University Erlangen-Nürnberg (FAU)
Email: {eischer,distler}@cs.fau.de

***Abstract*—In a geo-replicated setting, the response time of a leader-based Byzantine fault-tolerant (BFT) protocol often differs significantly depending on which of the replicas in the system is currently acting as leader. Identifying a single optimal leader position in general is impossible due to workload characteristics usually varying over the course of the day. As a consequence, the approach used in many existing BFT replication protocols, which assign the leader role in a static manner and only change the leader in case of suspected or detect faulty behavior, results in unnecessarily high latency in wide-area environments.**

In this paper we address this problem with ARCHER, a latency-aware mechanism to select the leader of a geo-replicated BFT system based on end-to-end response times measured by clients. To prevent faulty replicas from gaining an unfair advantage by sending protocol messages early, ARCHER relies on a hash-chain-based approach that enables clients to detect if a protocol phase has been skipped. In addition, ARCHER offers means to tolerate incorrect latency values reported by faulty clients and can also be extended to solve other selection problems such as the placement of active and passive replicas in resource-efficient BFT systems.

I. Introduction

Byzantine fault-tolerant (BFT) systems based on state-machine replication [1], [2], [3], [4], [5], [6], [7], [8], [9], [10] allocate special responsibilities to one of their replicas, the *leader*. In particular, for each incoming client request, the leader must initiate a BFT agreement protocol by proposing the request to the other replicas in the system, the *followers*. If the leader fails to meet its responsibilities, the followers cooperate to reassign the leader role to another replica to ensure progress.

To select the current leader, many existing BFT agreement protocols [1], [5], [6], [7], [8], [9], [10] rely on a static scheme that initially appoints the replica with the lowest id as leader, and if necessary reassigns the leader role in a round-robin fashion among replicas. In a local-area setting, in which all replicas are placed in close proximity to each other, this approach is feasible as the specific location of the leader usually does not affect overall system response time. This is mainly due to the fact that in such an environment all distances between a client and each replica are essentially equal. However, the same does not apply to wide-area settings where replicas reside at different geographical sites, for example, to prevent them from all being affected by a single data-center outage. Here, with client-to-replica and replica-to-replica communication delays differing, the latency a client observes may significantly vary depending on which replica currently acts as leader. Consequently, existing location-unaware leader selection approaches typically result in unnecessarily high response-time overheads.

In general, there are two important reasons for why selecting a latency-optimal leader is inherently difficult in wide-area environments: (1) Having determined a leader that minimizes response time for one client does not automatically mean that the same leader is also the best choice for clients at other locations. As a result, this makes it necessary to take the entire client workload into account when selecting a leader. Some BFT protocols try to mitigate this problem by continuously rotating the leader role among replicas and thereby allowing clients to always send their requests to the nearest replica [2], [4], [7], [10], [11]. However, experiments show that in practice this technique does not offer additional latency benefits compared with a fixed, well-selected leader [8], which is why in this work we focus on single-leader protocols. (2) Due to the set of active clients typically varying over time (e.g., as a consequence of workload shifts during the global day/night cycle), in wide-area settings there is usually no specific leader location that is optimal in all cases. This means that the traditional approach of only changing a leader if it fails to behave correctly is unsuitable to meet the demands created by the varying workloads of a geo-replicated system.

In this paper, we address these problems with ARCHER, a latency-aware mechanism that enables a geo-replicated BFT system to optimize the selection of its leader replica based on end-to-end response-time information collected and provided by clients. To adapt to changing workload conditions, clients periodically evaluate different leader candidates by measuring the response times of special probe messages that pass through the entire BFT protocol without modifying any state. By deterministically combining the latency information of different clients, correct replicas are then able to make consistent decisions on which replica in the system to select as leader.

ARCHER aims at minimizing the p-th percentile of the end-to-end latency observed by correct clients; the value for p is configurable and its maximum depends on the fraction of faulty clients that need to be tolerated in the worst case. Apart from offering resilience against faulty clients, our mechanism also mitigates the impact faulty replicas can have on the outcome of the selection process. During response-time measurements, ARCHER for example protects each probe message with a hash chain that enables a client to verify whether the reply to its probe message is actually the result of a complete BFT-protocol execution. As a consequence, it is impossible for faulty replicas to successfully skip protocol phases in an effort to force correct clients into reporting low response times.

This is the authors' version of an article presented at the *1st Workshop on Byzantine Consensus and Resilient Blockchains (BCRB '18)*, Luxembourg City, Luxembourg, 25 June 2018 and published in the *Proceedings of the 48th International Conference on Dependable Systems and Networks Workshops (DSN-W '18)*.

In particular, this paper makes the following contributions: (1) It presents ARCHER, a latency-aware mechanism to dynamically select the leader in geo-replicated BFT systems. (2) It shows the flexibility of ARCHER by explaining how to extend the approach to also support the placement of active and passive replicas in resource-efficient BFT systems [5], [9]. (3) It evaluates ARCHER for a key-value store whose replicas are distributed across multiple Amazon EC2 [12] regions.

II. Background and Problem Statement

In this section, we provide background on leader-based BFT systems and motivate why the state-of-the-art approach to select a leader has drawbacks in geo-replicated environments.

Background. Agreement-based BFT systems [1], [2], [4], [5], [8], [9] ensure consistency by establishing a global total order on client requests before executing them. For this purpose, such systems rely on multi-phase agreement protocols in which, having received a request, a leader replica proposes the request to its follower replicas. Figure 1 presents two examples for leader-based BFT replication protocols, PBFT [1] and RePBFT [9], which both use a total of $3f + 1$ replicas to tolerate f faults and require a client to wait for $f+1$ matching replies from different replicas to accept a result as correct. While in PBFT all replicas in the system actively participate in the agreement process as well as the execution of requests, in RePBFT only $2f + 1$ replicas are active during normal-case operation; to save resources, the other f replicas remain passive and serve as standbys to be activated in case of faults.

With all replicas residing in close proximity to each other, in a local-area environment the overall response time of a BFT system in general does not depend on the question which of the replicas currently serves as leader. However, as Figure 2 illustrates by the example of PBFT, the same does not apply to wide-area environments, where latency may vary highly due to differences in client-to-replica as well as replica-to-replica communication delays. In such settings, for the same client and replica locations, the position of the leader can have a significant impact on when a replica enters the next agreement phase (as indicated by the bold black lines in Figure 2), and consequently on when a client is able to obtain enough matching replies for a stable result. For resource-efficient protocols such as RePBFT [5], [9], similar observations can be made not only for the selection of the leader but also with regard to the decision which of the replicas to set passive.

Existing Approaches. Not focusing on geo replication, many BFT systems [5], [6], [7], [8], [9], [10] apply PBFT's technique of initially selecting the leader as the replica with the lowest id in the system. Furthermore, they only reassign the leader role to another replica if the current leader is deemed faulty. Using the same method in a geo-replicated setting is very likely to result in unnecessarily high response times, caused by non-optimal leader placement. Apart from that, this approach also prevents a BFT system from supporting use cases where the location from which clients predominantly issue requests varies over the course of the day, for example, due to access patterns being correlated to users' office hours.

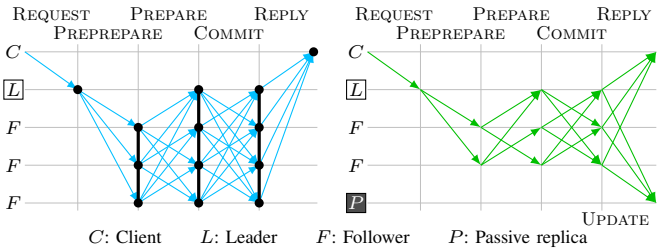


Figure 1. PBFT (left) and RePBFT (right, normal case) in a local-area setting.

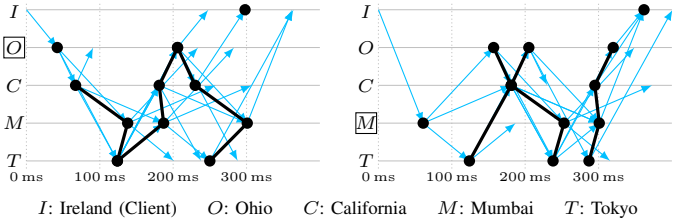


Figure 2. PBFT in a geo-replicated setting with different leaders.

Some BFT protocols circumvent the problem of having to select a leader by frequently reassigning the leader role among replicas [2], [4], [7], [10], [11]. While this approach can be beneficial if at all times requests are issued from a variety of geographical locations, it comes with additional overhead when the workload primarily stems from the same location, as replicas in such case are forced to actively skip their turns. Furthermore, recent experiments have shown that in real-world settings a rotating leader does not offer latency improvements over a single leader that resides at a suitable location [8].

Problem Statement. Considering the drawbacks of existing approaches in the context of wide-area environments, there is a need for a latency-aware mechanism to select the leader replica of a geo-replicated BFT system. In particular, such a selection mechanism should provide the following properties:

- *Effectiveness*: The selection process should identify the replica that, when serving as leader, offers the lowest overall response time for the current system workload.
- *Consistency*: To prevent inconsistencies between correct replicas, all correct replicas must select the same leader.
- *Resilience*: A faulty replica or client should not be able to manipulate the decision process in such a way that eventually an unsuitable replica is selected as leader.
- *Adaptiveness*: The selection mechanism should enable a BFT system to dynamically reassign the leader role to another replica in case a change in workload characteristics allows the other replica to offer lower response times.
- *Flexibility*: The mechanism should not only provide support for leader selection, but also be able to solve related problems such as the placement of passive replicas.

Meeting these requirements in the potential presence of unreliable clients and replicas is especially challenging, as some tasks, for example measuring the response time observed by a specific client, cannot be easily reproduced or verified from a remote location. As a consequence, the selection mechanism must be designed to tolerate partly inaccurate information.

III. ARCHER

In this section, we present details on our latency-aware leader-selection technique ARCHER. Furthermore, we discuss how the method can be extended to also select passive replicas.

A. System Model

ARCHER targets leader-based BFT systems whose clients and replicas are distributed across different geographical locations, resulting in response times to be dominated by communication delays. Our approach is not limited to a particular BFT agreement protocol, however, for clarity in the following we focus our discussion on ARCHER's integration with PBFT and RePBFT, because these are the two replication protocols currently supported by our prototype. Both protocols require clients and replicas to authenticate the messages they send using MAC authenticators [1]. For this purpose, each sender of a message shares an individual secret key with each receiver.

Up to f of the $3f + 1$ replicas and a subset of clients may be faulty, this also includes arbitrary, malicious behavior such as trying to manipulate the selection process. For typical ARCHER use cases, we expect the fraction of faulty clients to be small based on the rationale that, if a large part of a system's clients are faulty, optimizing latency is probably not the most important problem that needs to be solved.

B. General Approach

ARCHER enables a geo-replicated BFT system to minimize the p-th percentile of the end-to-end response times observed by correct clients c; p is a configurable value and also has an impact on the fraction of faulty clients $b = 1 - c < c \cdot (1 - p)$ the selection mechanism is able to tolerate. That is, ARCHER for example is resilient against up to about 9% faulty clients when it has been configured to optimize the 90th response-time percentile for correct clients. If the fraction of faulty clients exceeds b, a non-optimal configuration may be chosen. However, even in such case the safety of the BFT system remains unaffected as system safety does not depend on ARCHER, but on the underlying BFT replication protocol.

ARCHER applies a two-step approach to make selection decisions in a consistent and resilient manner. In the first step, clients rely on probe messages to measure system response times for different leaders. Similar to regular requests, probes pass through the entire replication protocol, but unlike requests they do not have any effect on agreement or application state. During its way through the protocol, a probe collects information on the duration of different protocol phases and the involvement of replicas. To prevent faulty replicas from being able to unfairly improve their chances of becoming leader by skipping protocol phases, ARCHER protects probes using a hash-chain-based approach that later allows clients to verify whether their probes have been handled correctly.

In the second step, clients make the knowledge that they have obtained via probes available to the system by attaching it to their regular requests. As requests are agreed on, ARCHER this way can ensure that all correct replicas use the same inputs and therefore make consistent selection decisions.

C. Latency-Aware Leader Selection

Below, we present details on the two steps performed by ARCHER to select a leader based on end-to-end latency information measured and provided by a BFT system's clients.

Measuring Latencies. To evaluate the impact of different leaders on overall system response times, ARCHER clients send probe messages to all replicas and measure for each probe the time it takes to collect $f+1$ matching replies. This process is repeated periodically to update the latency information and thereby enable ARCHER to adapt to changing network and workload conditions. Upon receiving a probe, a replica starts a new BFT agreement protocol instance for the probe and acts as leader for this instance; this is independent of whether the replica currently is the actual leader or a follower. While the probe passes through the protocol, it is strictly separated from the regular requests to avoid any interference. Furthermore, when a probe commits on a replica, the replica sends a corresponding reply to the client, without modifying the application state. Executing the entire BFT protocol for probes ensures that the response times measured by clients closely resemble those for regular requests in the geo-replicated BFT system, which are dominated by communication delays between sites.

During measurements, ARCHER prevents faulty replicas from gaining an advantage by manipulating the process in their favor. In general, there are two main strategies a faulty replica might pursue in an effort to affect the results: (1) A faulty replica may increase response times by deliberately dropping or delaying probes. For ARCHER, this does not pose a problem as a faulty replica this way only reduces its own impact on the agreement process, and because the effective delay is limited to the point where the protocol progress no longer depends on the faulty replica. To further ensure that faulty replicas are not able to slow down the system by delaying regular requests while properly processing probes, the underlying protocol should monitor request-processing latency. (2) A faulty replica can try to improve the chances of a particular leader (potentially the replica itself) being selected by skipping protocol phases and, for example, immediately sending a reply. ARCHER addresses this problem by enabling clients to detect such scenarios with the help of hash-chain-based proofs that all replicas must include in their replies to probes, as illustrated in Figure 3.

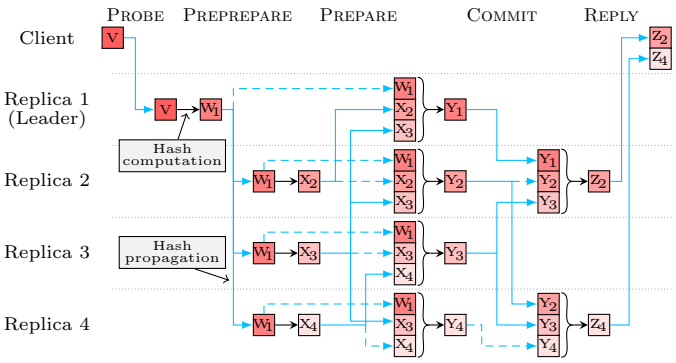


Figure 3. Example hash tree for a PBFT instance (irrelevant parts omitted); hashes are computed using input hash(es) and respective client–replica secret.

The idea behind this approach is to force replicas to prove that their probe reply is the result of a correct BFT-protocol execution. For this purpose, whenever a client sends a probe it also includes a newly-generated random hash (i.e., V in Figure 3) in the message. In every protocol phase, each replica is then required to compute and forward a new hash (HMAC) of its own that proves that the replica waited until it has actually received the messages necessary to complete the current phase. To send a commit, a replica in PBFT for example needs to have the leader's pre-prepare message and $2f$ prepare messages from different followers. Therefore, in the scenario in Figure 3, Replica 4 for example computes its commit HMAC Y_4 over a concatenation of the pre-prepare's hash W_1 and the hashes X_3 and X_4 contained in prepares. For each HMAC computation, a replica includes the secret it shares with the client that initiated the probe. This ensures that replicas cannot forge the hashes of other replicas and consequently makes it impossible for a faulty replica to produce a valid hash without first having received enough messages from the previous protocol phase.

Apart from the determined HMAC, a replica also forwards the replica ids that correspond to the input hashes on which the HMAC is based (e.g., 1, 3, and 4 in the example discussed above). This way, each probe reply not only contains the resulting hash but also a list of replica ids representing the path through the agreement protocol the probe has taken. Knowing the secrets it shares with each replica, this information enables a client to verify that a probe has been handled correctly based on the reply's hash. To do so, starting with the initial random hash from the probe, a client reproduces the protocol path as stated in the probe reply. A client only accepts a probe reply if the reproduced hash matches the hash contained in the reply.

If a client receives less than $f+1$ valid probe replies within a predefined period of time (e.g., because of X_3 and thus all subsequent hashes being incorrect in Figure 3), the client restarts its measurement by submitting a new probe with a new random hash. However, this time, in advance, it computes the intermediate hashes for the protocol paths that previously failed and includes knowledge about the intermediate hashes in the probe, thereby allowing replicas to already detect and drop faulty messages during protocol execution. To enable a replica to verify intermediate hashes without revealing them, a probe in such case contains hashes of intermediate HMACs. Due to the potentially large number of possible protocol paths, a client only computes intermediate hashes for the paths that have failed so far and if necessary repeats this procedure with additional paths until it obtains enough valid probe replies.

Selecting a Leader. Relying on ARCHER for leader selection, replicas maintain a database with the end-to-end response times measured by clients (see Figure 4). To consistently update this database on all correct replicas, clients attach their measurement results to regular requests, and replicas only adopt a new value when the corresponding request has been committed. Using the database, replicas periodically reevaluate the current choice of leader every u database updates; u is a configurable value. Furthermore, to account for varying network and workload conditions, replicas deterministically discard database entries that have not been updated for a predefined number of reevaluation intervals. In summary, by guaranteeing that replicas operate on the same database state and make decisions at the same logical point in time, ARCHER ensures that all correct replicas select the same leader. The overhead for handling the probes depends on their frequency, the number of clients, and the amount of regular requests.

	Leader 1	Leader 2	Leader 3	Leader 4
Client 1	297 ms	297 ms	360 ms	**376 ms**
Client 2	345 ms	**363 ms**	479 ms	402 ms
Client 3	359 ms	312 ms	**398 ms**	318 ms
Client 4	**355 ms**	366 ms	288 ms	288 ms
90th percentile	355 ms	363 ms	398 ms	376 ms

Figure 4. Example for the response-time database maintained by replicas.

When a reevaluation is due, each replica determines a reference response-time value for each leader candidate by applying the c-th percentile over all client entries, with c being the minimum fraction of correct clients; Figure 4 illustrates this step for an example with $c = 90$. Using the c-th percentile as reference for two reasons enables ARCHER to minimize the influence faulty clients can have by reporting incorrect latency values: (1) It ensures that faulty clients cannot force an arbitrarily high reference value in order to prevent a particular replica from becoming leader. (2) In cases where faulty clients try to favor a certain replica by incorrectly reporting low values, the approach guarantees that the reference value is still at least as high as the targeted p-th response-time percentile for all correct clients (see Section III-B). This is true as long as the fraction of faulty clients suffices $b = 1 - c < c \cdot (1 - p)$; that is, it must be smaller than the fraction of correct clients that experience response times above the p-th percentile.

Having determined reference response times for all leader candidates, a replica selects the candidate with the lowest reference value as new leader, using replica ids as tie breaker. If the new leader differs from the current leader, the replica then advises its agreement protocol to initiate a view change and to switch into a new view in which the selected candidate acts as leader. Relying on the BFT protocol's view-change mechanism to switch the leader ensures liveness even if the new leader later turns out to be faulty. To prevent a system from quickly changing between different views, ARCHER blacklists leaders whose views have been abandoned due to suspected or detected faulty behavior. As leader changes can only be proposed every u database updates, the number of additional view changes triggered by ARCHER is limited, thus maintaining the liveness guarantees of the underlying protocol.

D. Latency-Aware Selection of Passive Replicas

In the following, we describe how the mechanism presented in Section III-C can be extended to also select passive replicas.

Measuring Latencies. When also taking passive replication into account, the number of possible leader and passive-replica combinations grows to a point where measuring them individually is no longer practical. As illustrated in Figure 5, to select passive replicas ARCHER therefore applies an approach that first measures relative protocol-message arrival times when all

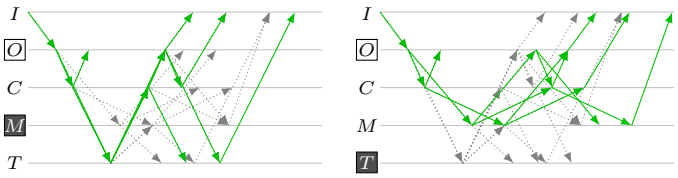


Figure 5. RePBFT protocol execution for different passive replicas.

replicas are active (dotted gray lines) and then calculates the differences that would result from specific replicas being passive (solid green lines). For this purpose, each replica creates a local timestamp when it obtains the pre-prepare message for an ARCHER probe, and in the following uses this timestamp as baseline to measure and record the relative arrival times and sender replica ids of subsequent prepare and commit messages. When the replica has received all agreement messages from the probe's protocol instance (or after a timeout), the replica sends the collected information to the client in a dedicated message. The client then checks whether the order of the reported arrival times matches the verified protocol path announced in the corresponding probe reply and, if this is the case, forwards the values to replicas, attached to one of its regular requests.

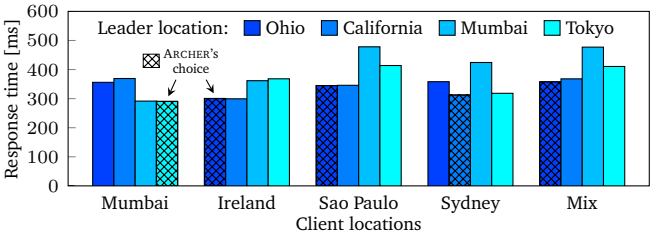
Selecting Passive Replicas. Based on the provided relative arrival times representing a protocol execution with only active replicas, each replica is able to assess the impact of passive replicas. As depicted in Figure 5, a replica to this end assumes a subset of replicas to be passive and virtually eliminates all their agreement messages. In addition, the replica reconstructs how the protocol execution would have progressed without these messages, thereby virtually postponing messages of protocol phases that in such case would have started later. By repeating this procedure for all combinations of leaders and passive replicas, each replica is able to extend its response-time database (see Section III-C) by additional columns for different system configurations. Using the same method as for leader selection, this allows all correct replicas to compute reference response times for each system configuration and to consistently decide on which of the replicas to set passive.

IV. EVALUATION

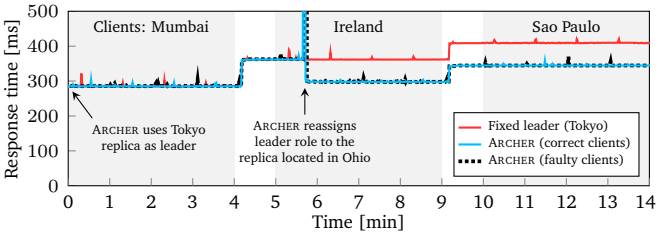
In this section, we present evaluation results from our ARCHER prototype which implements a geo-replicated key-value store and runs on Amazon EC2 [12]. For all experiments, we place the replicas in Ohio, California, Mumbai, and Tokyo, while the locations of clients vary between Mumbai, Ireland, Sao Paulo, and Sydney. Furthermore, we use a total of 60 clients and configure ARCHER to optimize the 90th percentile of the response times observed by correct clients.

Leader Selection. In our first experiment, we evaluate ARCHER's ability to minimize the response times of PBFT for a given workload. To this end, we measure latencies for all possible leader configurations and compare the results to the configuration chosen by ARCHER. In total, we examine five workloads: four scenarios in which all requests originate from only one client location each, and one scenario ("Mix") where the clients are uniformly distributed across all client locations. Figure 6a presents the measurement results of this experiment and the respective configuration favored by ARCHER (▨). The numbers confirm that the response times of a geo-replicated BFT system can differ significantly depending on where its leader replica is located. With all clients in Sao Paulo, overall latency for example increases by as much as 39% when the leader is in Mumbai compared with the leader being in Ohio. Besides, our results also show that there is no single replica that as leader provides optimal response times in each case, thereby confirming the need for a mechanism to dynamically select a leader. Relying on ARCHER to solve this problem leads to optimal response times for all evaluated workloads.

Adaptation to Varying Workloads. In our second experiment, we evaluate ARCHER in the presence of a varying workload where the locations from which client requests are issued change over time. For comparison, we repeat the experiment with a fixed leader in Tokyo, which is optimal for the workload conditions that exist at the beginning of the experiment when all clients are located in Mumbai. As shown in Figure 6b, as long as requests originate in the same location, both approaches achieve similar response times. However, when the client workload starts to gradually shift from Mumbai to Ireland four minutes into the experiment, and especially one minute later when the process has completed, the fixed leader in Tokyo no longer offers the lowest response times possible. In contrast, based on the probe-measurement results provided by clients ARCHER at this point learns that placing the leader in Ohio improves performance, and consequently adapts the system by initiating a view change. As a result of the leader switch, the 90th percentile of end-to-end response times decreases by about 18% compared with the fixed-leader approach. When the client workload shifts again nine minutes into the experiment, this time from Ireland to Sao Paulo, ARCHER identifies Ohio to still be the optimal leader location and therefore keeps the current system configuration.



(a) 90th percentile of response times for different client and leader locations.



(b) 90th percentile of response times for a dynamically varying workload.

Figure 6. Selection of a leader replica with ARCHER.

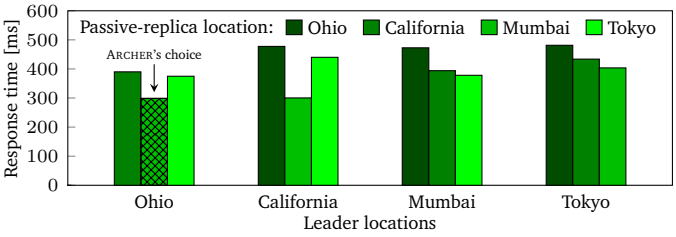


Figure 7. Selection of a combination of leader and passive replica.

Resilience Against Faulty Clients. To investigate the influence of faulty clients on ARCHER, we modify the previous experiment and add 5 clients located in Mumbai that report manipulated probe-measurement results to replicas in an effort to keep the leader in Tokyo, that is, at the location that is optimal for these clients. As depicted in Figure 6b, with ARCHER being configured to optimize the 90th response-time percentile, 8% of active clients are unable to force a leader selection that would be disadvantageous for correct clients.

Passive-Replica Selection. In the final experiment, our goal is to select a RePBFT system configuration for a scenario in which clients are located in Ireland. ARCHER for this setting places the leader in Ohio and the passive replica in Mumbai. Having measured the response times for all possible combinations of leader and passive-replica locations (see Figure 7), we can confirm that the configuration selected by ARCHER indeed minimizes response times for the targeted use-case scenario.

V. Related Work

While most BFT protocols only switch their leader in case of faults [1], [5], [6], [8], [9], [10], some protocols also reassign the leader role during normal-case operation, for example, to balance load across replicas [2], [7], [10], to improve system resilience against slow replicas [3], or to minimize communication delays between clients and the leader [4], [11]. Similar to ARCHER, these BFT protocols exploit the fact that the single-leader requirement only pertains to each agreement-protocol instance individually and that consequently the leader role may be reassigned between instances. ARCHER builds on this principle by dynamically switching to a new leader if, for example as result of a change in workload conditions, the selected replica offers lower end-to-end response times.

Apart from single-leader protocols, there are BFT protocols that make use of multiple leader replicas at a time. RBFT [6], for example, reduces the degrading impact a slow leader can have on performance by executing multiple agreement-protocol instances for the same client request in parallel; for each instance, a different replica acts as leader. Omada [13] partitions the agreement of client requests to balance load across heterogeneous servers and appoints one leader for each partition. Both of these protocols face the task of selecting their leaders from the group of all replicas and would therefore benefit from applying ARCHER in geo-replicated settings.

Steward [14] tolerates arbitrary faults in wide-area settings by combining Byzantine fault-tolerant replica groups at different sites with a leader-based crash-tolerant agreement protocol. The problem of finding suitable leaders in crash-tolerant systems has, for example, been investigated by Liu and Vukolić [15]. They presented an approach to select and dynamically adapt a set of leaders based on measured round-trip latency between replicas and an enumeration of all possible leader sets. Although effective, this technique cannot be directly applied to BFT systems as measuring and sharing inter-replica communication delays is not straightforward in the presence of Byzantine faults. The same applies to leaderless crash-tolerant system architectures in which all replicas can propose client requests and consistency for interfering requests is ensured by conflict-resolution sub protocols [16], [17].

VI. Conclusion

ARCHER optimizes end-to-end latency in geo-replicated BFT systems by selecting a suitable leader and/or set of passive replicas. Periodically repeating this process consequently allows a system to support use cases with varying client locations. ARCHER offers resilience against both faulty replicas as well as faulty clients, for example, protecting probe messages with hash chains to ensure that faulty replicas cannot force correct clients into promoting non-optimal selection decisions.

Acknowledgments: This work was partially supported by the German Research Council (DFG) under grant no. DI 2097/1-2 ("REFIT").

References

[1] M. Castro and B. Liskov, "Practical Byzantine fault tolerance," in *Proc. of OSDI '99*, 1999.
[2] G. S. Veronese, M. Correia, A. N. Bessani, and L. C. Lung, "Spin one's wheels? Byzantine fault tolerance with a spinning primary," in *Proc. of SRDS '09*, 2009.
[3] A. Clement, E. Wong, L. Alvisi, M. Dahlin, and M. Marchetti, "Making Byzantine fault tolerant systems tolerate Byzantine faults," in *Proc. of NSDI '09*, 2009.
[4] G. S. Veronese, M. Correia, A. N. Bessani, and L. C. Lung, "EBAWA: Efficient Byzantine agreement for wide-area networks," in *Proc. of HASE '10*, 2010.
[5] R. Kapitza, J. Behl, C. Cachin, T. Distler, S. Kuhnle, S. V. Mohammadi, W. Schröder-Preikschat, and K. Stengel, "CheapBFT: Resource-efficient Byzantine fault tolerance," in *Proc. of EuroSys '12*, 2012.
[6] P.-L. Aublin, S. B. Mokhtar, and V. Quéma, "RBFT: Redundant Byzantine fault tolerance," in *Proc. of ICDCS '13*, 2013.
[7] J. Behl, T. Distler, and R. Kapitza, "Consensus-oriented parallelization: How to earn your first million," in *Proc. of Middleware '15*, 2015.
[8] J. Sousa and A. Bessani, "Separating the WHEAT from the chaff: An empirical design for geo-replicated state machines," in *Proc. of SRDS '15*, 2015.
[9] T. Distler, C. Cachin, and R. Kapitza, "Resource-efficient Byzantine fault tolerance," *IEEE Trans. on Comput.*, vol. 65, no. 9, pp. 2807–2819, 2016.
[10] J. Behl, T. Distler, and R. Kapitza, "Hybrids on steroids: SGX-based high performance BFT," in *Proc. of EuroSys '17*, 2017.
[11] Y. Mao, F. P. Junqueira, and K. Marzullo, "Towards low latency state machine replication for uncivil wide-area networks," in *Proc. of HotDep '09*, 2009.
[12] Amazon EC2, https://aws.amazon.com/ec2/.
[13] M. Eischer and T. Distler, "Scalable Byzantine fault tolerance on heterogeneous servers," in *Proc. of EDCC '17*, 2017.
[14] 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 Trans. on Dependable and Secure Computing*, vol. 7, no. 1, pp. 80–93, 2010.
[15] S. Liu and M. Vukolić, "Leader set selection for low-latency geo-replicated state machine," *IEEE Trans. on Parallel and Distributed Systems*, vol. 28, no. 7, pp. 1933–1946, 2017.
[16] I. Moraru, D. G. Andersen, and M. Kaminsky, "There is more consensus in egalitarian parliaments," in *Proc. of SOSP '13*, 2013.
[17] B. Arun, S. Peluso, R. Palmieri, G. Losa, and B. Ravindran, "Speeding up consensus by chasing fast decisions," in *Proc. of DSN '17*, 2017.