Low-Latency Geo-Replicated State Machines with Guaranteed Writes

Michael Eischer, Benedikt Straßner, and Tobias Distler
Friedrich-Alexander University Erlangen-Nürnberg (FAU)

Abstract

When deployed in geo-distributed environments, existing state-machine replication protocols require at least one wide-area communication step for establishing a total order on client requests. For use cases in which clients are not interested in the actual result of a request, but just need a guarantee that the request will be processed eventually, this property usually incurs unnecessarily high response times. To address this problem we present Weave, a cloud-based geo-replication protocol that relies on replica groups in multiple geographic regions to efficiently assign stable sequence numbers to incoming requests. This approach enables Weave to offer *guaranteed writes* which in the absence of faults only wait for communication within a client's local replica group to produce an execution guarantee for a particular sequence number. Our experiments with a distributed queue and a replicated log show that guaranteed writes can significantly improve response times of geo-replicated applications.

CCS Concepts. • **Computer systems organization → Dependable and fault-tolerant systems and networks**; **Reliability**; **Cloud computing**; • **Software and its engineering → Consistency**.

Keywords. Geo-replication, state-machine replication, consistency, wide-area networks, cloud

ACM Reference Format:
Michael Eischer, Benedikt Straßner, and Tobias Distler. 2020. Low-Latency Geo-Replicated State Machines with Guaranteed Writes. In *7th Workshop on Principles and Practice of Consistency for Distributed Data (PaPoC '20), April 27, 2020, Heraklion, Greece.* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3380787.3393686

1 Introduction

Geo-distributed state-machine replication [30] enables systems to tolerate failures of entire data centers by maintaining consistent copies of an application's state at different geographic sites. Relying on traditional protocols such as Paxos [7, 16, 18, 19] for this purpose, one of the sites acts as leader and proposes an order on incoming client requests, which then needs to be confirmed by a majority of replicas before the requests can be executed. Although effective, this approach has two major drawbacks with regard to latency: (1) Due to replicas being distributed across different regions, the ordering of requests requires communication over wide-area networks and therefore usually results in high response times. (2) As all client requests must reach the leader before they can be ordered, the end-to-end latency experienced by clients may vary significantly depending on the geographic location of a client relative to the leader replica.

Existing solutions for mitigating these problems include the use of weighted quorums to increase the chance of a fast consensus [31] as well as the optimized selection of the leader location [12]. Alternatively, it is possible to design a system in such a way that multiple replicas share the responsibilities of the leader [10, 23], thereby allowing each client to submit requests to the replica closest to its own location. Unfortunately, these approaches still share one property: They all require at least one wide-area communication step before being able to guarantee the execution of a client request. In particular, this unnecessarily increases response times for use cases in which clients are not interested in the actual results of (write) requests, but only want to be sure that their requests are executed eventually. For example, if the replicated application implements a distributed message queue through which a set of producer clients communicate with a set of consumer clients [9], it is often sufficient for a producer to learn that the enqueue operation of a message will succeed in the future and the message therefore will not be lost. Another typical example of such a use case is a reliable geo-replicated log persistently storing the modifications to an application's state [27], especially if the associated application already can respond to the client once it has proof that a modification will eventually be included in the log.

In this paper we present Weave, a Paxos-based geo-replication protocol with multiple leaders that (in addition to regular writes) features *guaranteed writes* to address the problems discussed above. Unlike regular writes, guaranteed writes produce replies that do not contain the actual result of an operation, but instead represent a guarantee to the client that the request will later be processed at a specific sequence number. Leveraging the availability characteristics provided by modern cloud infrastructures such as Amazon EC2, Weave's novel architecture enables the protocol to perform guaranteed writes without involving wide-area communication. For this purpose, Weave models

PaPoC '20, April 27, 2020, Heraklion, Greece
© 2020 Copyright held by the owner/author(s). Publication rights licensed to ACM.
This is the author's version of the work. It is posted here for your personal use. Not for redistribution. The definitive Version of Record was published in *7th Workshop on Principles and Practice of Consistency for Distributed Data (PaPoC '20), April 27, 2020, Heraklion, Greece*, https://doi.org/10.1145/3380787.3393686.

a system as a collection of replica groups that is distributed across multiple geographic regions; within each region, the replicas of a group are hosted in separate cloud-provided fault domains ("availability zones" [2]). In combination with the use of optimized quorums [14], this setting in the normal case allows Weave to safely assign stable sequence numbers to guaranteed writes locally within each group, that is, without wide-area interaction with other groups.

For reads, Weave offers clients the flexibility to trade off latency for consistency [32] by choosing between different guarantees. Among other things, this includes the possibility to perform group-local reads that are ensured to reflect the effects of a previous guaranteed write made by the same or another client. To do so, in its read request a client states the sequence number returned by the guaranteed write in question, thereby instructing replicas to only invoke the read operation after having executed the corresponding write.

In particular this paper makes the following contributions: (1) It proposes guaranteed writes as a means for clients in geo-replicated systems to quickly get an execution guarantee for their requests. (2) It presents Weave, a multi-leader state-machine replication protocol that employs multiple replica groups and optimized quorum sizes to provide guaranteed writes. (3) It experimentally evaluates Weave for two cloud-based use cases: a distributed queue and a replicated log.

2 Background and Problem Statement

In this section, we provide background on geo-replicated state machines and motivate why state-of-the-art approaches incur a significant latency overhead for write requests when a client only needs an execution guarantee.

2.1 Background

State-machine replication [30] frequently serves as the basis to provide clients with a fault-tolerant service. It works by running a deterministic state machine on multiple servers, called replicas, which process client requests according to a total order, thereby progressing through the same application states. The replicas totally order requests by using a consensus protocol to assign them to slots with consecutive sequence numbers which define the execution order. This also means that a replica can execute a request only after it has received and processed all predecessors to ensure that all replicas execute the same set of requests in the same order.

One way to implement the consensus algorithm is by using a variant of the Paxos protocol [7, 16, 18, 19] as shown in Figure 1. As a first step the replicas elect a replica as leader which needs votes from a majority quorum of replicas. Afterwards the leader assigns requests to sequence numbers by sending Propose messages to all replicas. Each assignment must be confirmed by a majority quorum of replicas by replying with an Accept message to the leader. The leader's Propose message also implicitly counts as an Accept. Once

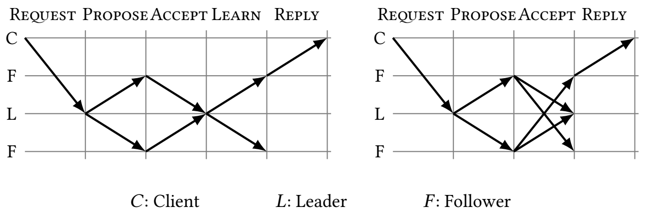

Figure 1. Message pattern of Paxos (left) and Paxos$_{\text{bc}}$ (right) to order a client request.

confirmed, the leader informs all replicas about the successful assignment. In case a leader is assumed to be faulty, for example due to not responding within a given timeout, the other replicas elect a new leader. The majority quorums guarantee that at least one replica that accepted a sequence number assignment is also part of the majority quorum required for the leader election thus ensuring that a successful assignment cannot change as a result of faults.

In the context of geo-replication, replicas are located in data centers in several regions spread across the globe to have replicas located near the clients. However, this has the downside that Paxos will require several wide-area communication steps that add significant latency to the request processing: As a first step, the client has to send its request to the leader potentially located in a different region. Afterwards, ordering the request incurs two further wide-area communication steps to a majority of replicas. And finally, the leader forwards the decision to the followers, which can then reply to the client afterwards. In total, it can take several hundred milliseconds before a request can be processed.

2.2 Existing approaches

In the following, we review several approaches that reduce the number of wide-area communication steps required in order to improve the request processing latency.

As illustrated on the right-hand side of Figure 1, a simple variation of Paxos, in the following referred to as Paxos$_{\text{bc}}$, removes one communication step [16, 19]: Replicas broadcast their Accept message to all other replicas instead of just sending it to the leader. Every replica can then execute the client request right after collecting Accepts from a majority quorum, allowing a replica near the client to send the reply.

Mencius [23] splits the leader role across all replicas by statically partitioning the sequence number space and assigning a part to each of them. A client sends its request to the nearest leader to avoid the first wide-area communication step. Each leader can then independently propose requests for its sequence numbers. However, the replicas may need to additionally coordinate over the wide-area network via Skip messages to tell other leaders to close gaps between sequence numbers, which can delay request execution.

Clock-RSM [10] avoids the latter problem with globally synchronized clocks whose timestamps are used for ordering requests instead of sequence numbers. The assignment of requests to timestamps is broadcast in a fault-tolerant way comparable to Paxos$_{bc}$. This communication also confirms that all proposals up to a certain timestamp were received and that all requests with an earlier timestamp can be executed. Clock-RSM reduces the processing latency down to two wide-area communication steps, but cannot remove the delays completely.

For read requests it is possible to completely avoid wide-area communication in case the client can handle the retrieval of slightly outdated data [8, 15, 32]. In that case, a replica can process a read request solely based on its current state thereby avoiding all wide-area communication.

The latter is also possible by splitting the application state into smaller objects which are then located in a data center near the client [1, 21, 26]. However, this has the downside that by handling objects separately there is no longer a single total order across replicas, making it unsuitable for replication of a single state machine. In addition, this requires running a full Paxos cluster in each region.

2.3 Problem Statement

The reviewed approaches suggest that we can confirm the execution of a (write) request to a client only after at least one wide-area communication step, which causes high latency. For operations such as adding messages to a queue or logging state modifications, the client typically is, however, only interested in the fact that the service will process the command. This led us to ask whether it is possible to provide the client with lower latency for such requests. In particular, such a system should provide the following properties:

- *Guaranteed writes:* Provide the client with an execution guarantee for its request without having to wait on wide-area communication.
- *Consistent:* Ensure consistency of the replicated state machines at all times, that is, process all write requests according to a single total order.
- *Flexible:* Provide the client with the ability to choose between reads with different consistency levels.
- *Resource efficient:* The system should use just enough replicas to provide the previous properties in the fault-free case and degrade gracefully in case of faults.

3 Weave

In this section we present Weave and detail how it organizes and coordinates its replicas to enable guaranteed writes.

3.1 System Model

Weave focuses on stateful applications that are replicated across data centers in several geographic regions worldwide, with clients being located in the same areas. We assume each client to know the region that is closest to itself from a latency perspective. The application is implemented as a deterministic state machine and offers *write* operations, which can modify application state, and *read* operations, that do not.

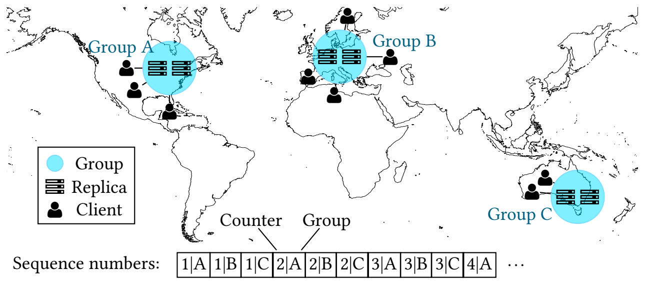

Figure 2. Weave system architecture

Replicas are assumed to only fail by crashing with a total of up to f faults. Recovery of a failed replica is possible as long as it ensures to not send conflicting messages (e.g. by maintaining a persistent log of previous messages). Each region hosts multiple replicas running in different data centers that are part of separate fault domains. Cloud providers offer this as availability zones, which represent data centers with independent power supplies and redundant network connections that are located several kilometers apart from each other [2]. This allows the replicas within a region to communicate with each other with low latency while at the same time minimizing the risk of correlated failures or network partitions of an entire region.

We assume the network communication to be asynchronous in general with synchronous periods during which messages are delivered with a bounded but unknown delay [11]. For measuring time intervals, we assume that replicas are equipped with real-time clocks whose frequency differs by less than one percent, a requirement which is fulfilled by typical real-time clocks [24].

3.2 General Approach

Weave enables replicas to quickly provide a client with guaranteed writes that will be executed at a certain sequence number while also offering the flexibility for the client to select the required consistency level for its read requests:

- *Linearizable* which requires waiting for one wide-area communication step to the farthest replica to guarantee a fully up-to-date reply.
- *Consistent Prefix with Bounded Staleness* which ensures that a replica has executed at least all requests up to a client-specified sequence number before replying.
- *Consistent Prefix* which just reads from the current state of the replica.

Weave places $f + 1$ replicas, a so-called *group*, in each of l regions yielding a total of $n = l \cdot (f + 1)$ replicas as shown in Figure 2. The sequence numbers are suffixed with a group index with each group being responsible for sequence

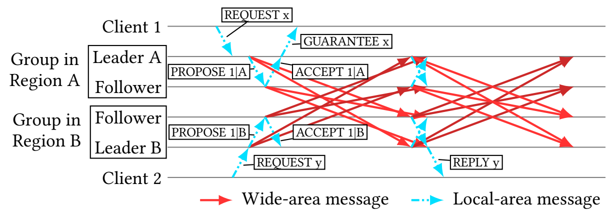

Figure 3. Messages exchanged between WEAVE replica groups to process a guaranteed write request x and a regular write request y.

numbers with their index. Each group runs an instance of Paxos_{bc} in which one replica of the group serves as leader and assigns requests to their sequence numbers while all other replicas from all regions are tasked with accepting the assignments. Based on work on the required quorum sizes for Paxos by Howard et al. [14] we use a small quorum size of $f+1$ in the normal case for accepting a proposal in exchange for a larger quorum of size $n-f$ for the much rarer leader election. This allows a group to assign sequence numbers to write requests without wide-area communication and to quickly provide the client with the corresponding guarantee.

Before a replica can execute a request at sequence number s it must wait until it received the requests for all lower sequence numbers. To avoid that the groups slow down each other, each group sends new proposals only in regular intervals while ensuring that they progress at a similar speed.

3.3 Reduced Quorum Sizes

Paxos traditionally uses majority quorums which grow with the number of replicas. As discovered by Howard et al. [14] the accept quorums do not need to overlap each other. It is sufficient that all accept quorums intersect with all leader election quorums. A PROPOSE message along with f ACCEPT messages is sufficient for assigning a sequence number to a request as this guarantees that in case of failures at least one correct replica remains that has the request. With a leader election quorum of size $n-f$ at least one of the $f+1$ accepting replicas is included in both quorums, thus ensuring that requests keep their sequence number across leader elections.

3.4 Replica Groups

Each of the l regions contains a group of $f+1$ replicas, which together with the accept quorums of size $f+1$ allows a group's leader to assign requests to sequence numbers without waiting for wide-area communication during normal operation, that is, the fault-free case. Clients in each region communicate directly with their region's leader thus ensuring that their requests and the associated replies are transmitted locally and therefore with low latency.

This structure enables *guaranteed writes*. As shown in Figure 3 for example the leader replica of group A can early on provide a guarantee to client 1 that its request x will be executed in the future. This is safe as even with the maximum of f faults at least one correct replica from the accept quorum of size $f+1$ remains, which knows the request assignment, and which would also be part of the $n-f$ quorum required for a leader election, thus ensuring that the request cannot be lost if a leader replica fails. Therefore the request for a guaranteed write will be executed eventually.

The sequence numbers are distributed equally onto all groups. A sequence number s consists of a counter c suffixed with a group index g: $s = c|g$. The total order is defined by the lexicographical order over the sequence numbers. Each group is responsible for all sequence numbers with its group index, which allows the leader of each group g to order requests without coordination with other groups. However, the execution of a request has to wait until all preceding sequence number slots were filled by all groups. For example as shown in Figure 3 the regular write request y of client 2 is executed after group B receives a sequence number assignment from group A.

In case the leader of a group has failed, one of the remaining replicas in the group will be elected as leader. The leaders of all other groups are unaffected by this change.

The replica groups and *guaranteed writes* are optimized for the normal case in which all replicas are working. In case of a failure the affected group needs help from other replicas to order a request which takes two wide-area communication steps. The PROPOSE message has to reach a replica in another region which then replies with the missing ACCEPT message to complete the ordering. This slow-down only affects the group with the faulty replica(s), all other groups still work at their normal speed. These groups can complete the ordering with ACCEPT messages from their own group.

3.5 Group Coordination

As a request can only be executed after all lower sequence number slots are filled, the groups need to coordinate to avoid blocking each other. WEAVE ensures that the leaders propose their assignment for a counter value c at roughly the same time. The mechanism consists of three parts:

a) Each leader only proposes a new assignment roughly every δ milliseconds. It contains either a batch of all requests received since the previous assignment, a single request or a special no-op request that is skipped during execution, in case no requests are available. δ should be chosen such that it is a few times smaller than the wide-area communication latency and thus only adds a small amount of additional latency to the execution of each request.

b) Each leader measures the communication latency to other leaders in regular intervals and uses it to estimate the one-way communication latency. For this each leader i sends PING messages containing a unique value u to all other leaders. Once a leader j receives a PING message it will at once respond with a PONG message. The leader i measures the time between sending the Ping and receiving the Pong message. We assume the one-way communication latency $d(i, j)$ between two replicas to be symmetrical and thus set it to half the minimum latency measured during the last 100 pings. The minimum is used to filter out interference like conflicting network traffic which can only increase the measured ping time, but not reduce it. The ping messages are piggybacked on the regular Propose and Accept message to avoid the message overhead.

c) Each leader estimates its current progress relative to all other leaders and adapts its proposal speed accordingly to stay in sync even with small differences in the leaders' clock speeds. When a Propose message with the sequence number $s = c_j|j$ from group j arrives at the leader of group i it is already $d(i, j)$ milliseconds old, which is the one-way communication latency between the groups' leaders. Based on this the leader of group i estimates its time offset $o(i, j)$ towards the leader of group j by assuming that the latter kept proposing requests in the meantime. A positive time offset means that group i is ahead of group j and vice versa for a negative offset. It is calculated as follows:

$$o(i, j) = c_i \cdot \delta - \big(c_j \cdot \delta + \min(t(i, j), 2\delta) + d(i, j)\big)$$

The formula calculates the time offset based on the difference between the current sequence counters of group i and j. For the latter the formula accounts for the time $t(i, j)$ that has passed since receiving c_j and the network delay $d(i, j)$. $t(i, j)$ is bounded to 2δ to ensure that the time offset starts to grow if the next Propose message is long overdue.

After proposing sequence number c_i the leader calculates the time offsets $o(i, j)$ to all other groups j and uses the largest value as reference point o_i to adapt its proposal interval δ_i for the next sequence number. It reacts to small differences in progress with small adjustments while making bigger adaptions to its proposal speed for large differences:

- $\delta_i = 0.5\delta$: Propose at twice the normal speed, if o_i is less than -10δ, as leader of group i lags far behind all other groups.
- $\delta_i = 0.95\delta$: Speed up slightly, if o_i is less than $-\delta/2$.
- $\delta_i = \delta$: Propose at normal speed, if the value of o_i is between $-\delta/2$ and $\delta/2$.
- $\delta_i = 1.05\delta$: Slow down a bit, if o_i is less than 20δ.
- $\delta_i = \infty$: Temporarily pause proposals, if o_i is larger than 20δ. This ensures that the groups cannot diverge too much if one of them is stuck, for example while electing a new leader. The proposal speed calculation must be repeated every 2δ milliseconds as long as this case applies or when new proposals arrive.

These adjustments cause groups which are behind to speed up their proposals to catch up with the other groups. Faster groups will also slightly slow down their proposal speed to help the slower groups catch up and to ensure that all groups propose the same sequence numbers at the same time.

3.6 Read Consistency Levels and Guaranteed Writes

For a read or write request to the application a client c sends a $\langle$Request, $c, t_c, o, m, a\rangle$ message to the leader of its nearest group. t_c is a client-specific request identifier, for example a counter that is increased for every request. The operation to execute is specified by o and the execution mode m can be either *linearizable read / write*, *guaranteed write* or *read with consistent prefix*. The first mode requires the leader to totally order the request and execute it afterwards, the second mode quickly returns an execution guarantee to the client and the last mode allows the leader to reply based on its local application state. A client can specify that the request must only be processed after the leader has reached a sequence number of at least a. For a read operation with bounded staleness the client sets a to the minimum expected sequence number. If consistent prefix is sufficient as consistency level, that is reading the state at an arbitrary point in the total order, the client just sets a to zero.

The replica groups described in Section 3.4 enable the leader of a group to quickly confirm a guaranteed write to the client. Once such a request was assigned to a sequence number s, the leader sends this guarantee in the form of a $\langle$Guarantee, $t_c, s\rangle$ message to the client. The client must then specify the returned sequence number s in later requests to ensure that these are executed after the guaranteed write and therefore provide sequential consistency.

After executing a regular read or write request the leader sends the result r in a $\langle$Reply, $t_c, r, s\rangle$ message to the client. It contains the request identifier t_c to enable the client to match the reply to the corresponding request. The leader also informs the client about the sequence number s of the latest totally ordered request after which o was processed.

4 Evaluation

In this section we experimentally evaluate the Weave prototype using a distributed queue and a replicated log as applications. We compare the results of Weave with Mencius$_{\text{bc}}$, which uses the Paxos broadcast optimization, and is the protocol implementation of Mencius providing the lowest latency [23]. In addition, we built an optimized version of Mencius$_{\text{bc}}$ called Mencius$^*_{\text{bc}}$, which we modified to provide guaranteed writes and reads with different consistency levels for a fair comparison with Weave. Mencius$^*_{\text{bc}}$ sends the write guarantee once a request was ordered, that is once the leader replica for the corresponding sequence number slot has collected a quorum of accept messages, which requires two wide-area communication steps. All systems are based on a common code base to ensure comparability. We leave the comparison with Clock-RSM to future work.

All replicas are spread across the three Amazon EC2 regions in Ohio, Frankfurt and Sydney and run in t3.micro instances (2 vCPUs, 1 GB RAM, Ubuntu 18.04.4 LTS and OpenJDK 11). We use a setup that tolerates one fault ($f = 1$).

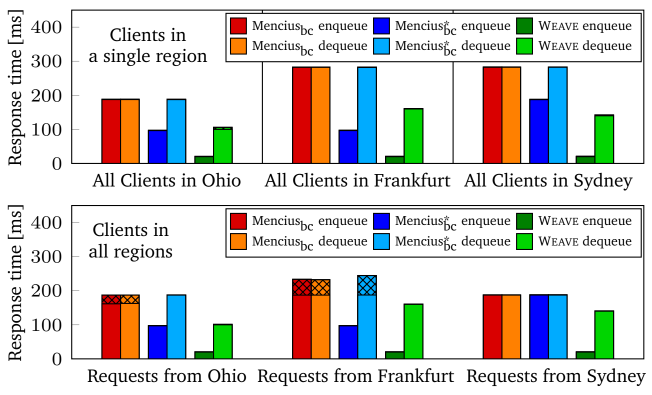

Figure 4. Median □ and 90th ▨ percentile of the response times for the distributed message queue.

Both Mencius variants comprise a single replica per region, whereas Weave relies on $f + 1 = 2$ replicas per region, each being located in a different availability zone (i.e., fault domain) to reduce the risk of correlated failures. Weave proposes new requests with an interval of $\delta = 20ms$.

The clients are co-located in the same regions but in a different availability zone than the replicas. All client instances of a region run in a single virtual machine. The clients are configured to issue their requests in a closed loop; that is, they send the next request immediately after getting a reply to the previous one. The request payload is set to 200 bytes.

4.1 Distributed Message Queue

In our first experiment, we evaluate the latency benefit of guaranteed writes and compare the results for Weave with those for Mencius. The clients representing a distributed application are split into producers and consumers, which coordinate via a distributed message queue. For this purpose, they enqueue and dequeue small messages. It is sufficient for the producers to learn that a message will be enqueued eventually, which is a textbook example for the use of guaranteed writes. The consumers on the other hand need the operation's result and thus have to issue regular write requests.

We evaluate four settings: three configurations with all clients located in each one of the three regions, and one configuration with clients equally spread across all regions. Each configuration uses a total of 60 client instances. The measured median and 90th percentile response times for the enqueue and dequeue operations are shown in Figure 4.

The dequeue operation, that is regular write requests, in $\text{Mencius}_{\text{bc}}$ and $\text{Mencius}^*_{\text{bc}}$ take between 187 and 283 ms to complete with a single active client location. With multiple client locations, the 90th percentile of $\text{Mencius}_{\text{bc}}$ improves to 233 ms. Weave on the other hand completes regular writes within 100 to 161 ms, outperforming Mencius by up to 142 ms for clients in Sydney. It also provides more stable response times with multiple active client locations than Mencius.

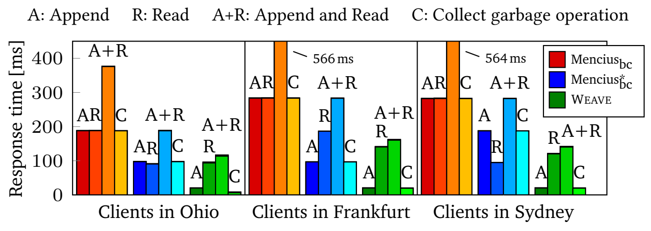

Figure 5. Median □ and 90th ▨ percentile response times for the replicated log.

Mencius requires at least two wide-area communication steps for coordination between leaders, whereas the replica groups and the continuous proposals of new sequence numbers enable Weave to only wait for one wide-area communication step and provide much lower response times.

For enqueue operations, which use guaranteed writes, Weave consistently achieves low response times below 21 ms. Even with the ability to process guaranteed writes, $\text{Mencius}^*_{\text{bc}}$ still needs 96 ms or more to issue an ordering guarantee. In other words, Weave's response time is more than 78% lower than the response time of $\text{Mencius}^*_{\text{bc}}$. This improvement is enabled by the replica groups of Weave which allow requests to commit without waiting for wide-area communication.

4.2 Replicated Log

The second experiment evaluates the latency benefit of combining guaranteed writes and reads with consistent prefix consistency to hide high execution latency. The scenario consists of an appender client which issues guaranteed writes to reliably *append* state updates to a replicated log. After ten updates it sends the sequence number w contained in the latest guaranteed write confirmation to a compactor client which retrieves, processes and garbage collects the state updates. For this it issues a *read* request with consistent prefix consistency that is marked for execution after sequence number w, allowing it to issue the read request while the corresponding log append request still waits for its execution. Afterwards the compactor client sends a guaranteed write to *garbage collect* old requests. We measure three workloads for which both clients are located in one of the three regions.

Figure 5 shows the response times for the individual operations. $\text{Mencius}_{\text{bc}}$ can only issue regular requests and it therefore takes more than 560 ms between the appender client issuing the write request and the collector receiving the result (A+R). With $\text{Mencius}^*_{\text{bc}}$ the response time decreases to between 188 and 284 ms as the appender client is able to notify the collector earlier on, which then waits until the state update is added to the log. Weave completes the append and read operation with a similar response time as is needed for just executing a single regular write. This takes within 113 to 162 ms depending on the client location, maintaining a large performance improvement compared to $\text{Mencius}^*_{\text{bc}}$.

5 Related Work

Fast Paxos [20] totally orders requests while allowing all replicas to directly propose sequence number assignments for their requests, thus avoiding the communication step to the leader needed in Paxos. This requires a larger *fast* quorum of size $\lceil \frac{3n}{4} \rceil$ to maintain safety. In case replicas propose different assignments additional communication steps are required to resolve the conflict. EPaxos [25] removes the leader role and instead allows each replica to directly propose requests. Commutative requests are ordered once they are accepted by a fast quorum of replicas, whereas conflicts are resolved using further communication steps. In WEAVE clients send their requests to the leader of their local replica group, thus avoiding the high latency associated with wide-area communication. As each replica group works on its own part of the sequence number space, no conflicts can arise.

Multi-Ring Paxos [5, 6] consists of multiple rings of replicas which independently order requests. An application can subscribe to multiple rings whose requests are then merged deterministically. Unlike WEAVE, which is optimized for low latency, Multi-Ring Paxos focuses on maximum throughput and requires all nodes, that is replicas and subscribers of a ring, to forward protocol messages along the ring. Especially in a geo-replicated setting with subscribers located in different regions, the latter increases the latency until a request is delivered to all subscribers. Merging requests from different rings requires that each ring proposes a fixed amount of sequence numbers per time interval. In Mult-Ring Paxos this is coordinated using globally synchronized clocks, whereas WEAVE only needs clocks running with a similar frequency.

CORFU [3] implements a distributed log by mapping sequence number ranges in a round robin manner onto multiple groups of storage nodes. The actual ordering happens on the storage nodes which enforce a write-once semantic in order to produce a totally ordered log. Conflicts between clients are avoided by using a central sequencer which assigns slots in the log. CORFU, in difference to WEAVE, is designed for local-area environments and would, when used in a wide-area environment, incur several wide-area communication steps for coordination via the central sequencer and for replication within a group resulting in high latency.

The use of replica groups is an established building block in the design of state-machine replication protocols. COP [4], SAREK [22] and Agora [29], for example, exploit replica groups to parallelize request agreement and enable a replicated system to effectively utilize multiple cores on each participating server. Omada [13] builds on this idea to support systems with heterogeneous servers by introducing groups with different weights. All mentioned protocols rely on full-fledged replica groups that contain enough replicas to handle both normal-case operation as well as fault tolerance within a replica group. In contrast, replica groups in WEAVE are designed to receive assistance from replicas outside of their group in case one or more members of their own group are faulty or slow. This enables WEAVE to operate with smaller groups and thereby improve resource efficiency.

MDCC [17] partitions its application state into objects and uses a generalized version of Fast Paxos to order updates and transactions involving multiple objects. The use of a fast quorum of replicas, however, leads to high latency as it requires communication with far away replicas. DPaxos [26] allows objects to be placed in a region near the client. Using the quorum optimization from FPaxos [14], state modifications are ordered within one or a small number of regions near the client. WPaxos [1] also uses optimized quorums and combines them with the ability for leader replicas to steal object ownership from other regions in order to adapt to workload changes. In comparison to WEAVE these approaches either require a large fast quorum or use a full set of at least $2f + 1$ replicas per region and only work for applications whose state can be partitioned.

Pando [33] which provides a wide-area optimized data storage with strong consistency, avoids the need for a fixed per-object leader replica for Paxos. Instead, for each write request which a replica receives, it tries to become leader for the accompanying object and orders the request afterwards. The transition from leader election to ordering is delegated to a centrally located replica, which combined with optimized quorums allows Pando to approach the latency of just executing the ordering step in the normal case while also avoiding the need to communicate with a leader in a possibly distant location. WEAVE has a leader replica in each region which also provides each client with a nearby leader.

SDPaxos [34] splits the request dissemination and the actual ordering into separate steps and combines them such that these require only a single round trip to a majority of replicas when tolerating up to two faults. The dissemination step is executed by every replica and therefore splits the transmission load whereas the more lightweight ordering is handled by a single leader replica. Canopus [28] is optimized for high throughput and forms groups consisting of nearby replicas, which each agree on an ordered sets of requests, which is then disseminated and merged into a single total order along an overlay tree. In WEAVE the load for distributing requests is split between the leader replicas.

6 Conclusion

In this paper we presented WEAVE, which provides guaranteed writes that enable clients to quickly get an execution guarantee for their requests. The latter is enabled by placing groups of $f+1$ replicas in each region, allowing them to order requests without waiting for wide-area communication.

Acknowledgments

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

References

[1] Ailidani Ailijiang, Aleksey Charapko, Murat Demirbas, and Tevfik Kosar. 2017. *WPaxos: Ruling the Archipelago with Fast Consensus*. Technical Report 2017-03. University at Buffalo, SUNY.

[2] Amazon EC2. 2020. Regions and Availability Zones. https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/using-regions-availability-zones.html.

[3] Mahesh Balakrishnan, Dahlia Malkhi, John D. Davis, Vijayan Prabhakaran, Michael Wei, and Ted Wobber. 2013. CORFU: A Distributed Shared Log. *ACM Transactions on Computer Systems* 31, 4, Article 10 (2013).

[4] Johannes Behl, Tobias Distler, and Rüdiger Kapitza. 2015. Consensus-Oriented Parallelization: How to Earn Your First Million. In *Proceedings of the 16th Middleware Conference (Middleware '15)*. 173–184.

[5] Samuel Benz, Leandro Pacheco de Sousa, and Fernando Pedone. 2016. Stretching Multi-Ring Paxos. In *Proceedings of the 31st Annual ACM Symposium on Applied Computing (SAC '16)*. 492–499.

[6] Samuel Benz, Parisa Jalili Marandi, Fernando Pedone, and Benoît Garbinato. 2014. Building Global and Scalable Systems with Atomic Multicast. In *Proceedings of the 15th International Middleware Conference (Middleware '14)*. 169–180.

[7] Tushar D. Chandra, Robert Griesemer, and Joshua Redstone. 2007. Paxos Made Live: An Engineering Perspective. In *Proceedings of the 26th Symposium on Principles of Distributed Computing (PODC '07)*. 398–407.

[8] Brian F. Cooper, Raghu Ramakrishnan, Utkarsh Srivastava, Adam Silberstein, Philip Bohannon, Hans-Arno Jacobsen, Nick Puz, Daniel Weaver, and Ramana Yerneni. 2008. PNUTS: Yahoo!'s Hosted Data Serving Platform. In *Proceedings of the 34th International Conference on Very Large Data Bases (VLDB '08)*. 1277–1288.

[9] Tobias Distler, Christopher Bahn, Alysson Bessani, Frank Fischer, and Flavio Junqueira. 2015. Extensible Distributed Coordination. In *Proceedings of the 10th European Conference on Computer Systems (EuroSys '15)*. 143–158.

[10] Jiaqing Du, Daniele Sciascia, Sameh Elnikety, Willy Zwaenepoel, and Fernando Pedone. 2014. Clock-RSM: Low-Latency Inter-datacenter State Machine Replication Using Loosely Synchronized Physical Clocks. In *Proceedings of the 44th International Conference on Dependable Systems and Networks (DSN '14)*. 343–354.

[11] Cynthia Dwork, Nancy Lynch, and Larry Stockmeyer. 1988. Consensus in the Presence of Partial Synchrony. *J. ACM* 35, 2 (1988), 288–323.

[12] Michael Eischer and Tobias Distler. 2018. Latency-Aware Leader Selection for Geo-Replicated Byzantine Fault-Tolerant Systems. In *Proceedings of the 1st Workshop on Byzantine Consensus and Resilient Blockchains (BCRB '18)*. 140–145.

[13] Michael Eischer and Tobias Distler. 2019. Scalable Byzantine Fault-tolerant State-Machine Replication on Heterogeneous Servers. *Computing* 101, 2 (2019), 97–118.

[14] Heidi Howard, Dahlia Malkhi, and Alexander Spiegelman. 2017. Flexible Paxos: Quorum Intersection Revisited. In *Proceedings of the 20th International Conference on Principles of Distributed Systems (OPODIS '16)*. Article 25.

[15] Patrick Hunt, Mahadev Konar, Flavio P. Junqueira, and Benjamin Reed. 2010. ZooKeeper: Wait-free Coordination for Internet-scale Systems. In *Proceedings of the 2010 USENIX Annual Technical Conference (USENIX ATC '10)*. 145–158.

[16] Jonathan Kirsch and Yair Amir. 2008. Paxos for System Builders: An Overview. In *Proceedings of the 2nd Workshop on Large-Scale Distributed Systems and Middleware (LADIS '08)*. Article 3.

[17] Tim Kraska, Gene Pang, Michael J. Franklin, Samuel Madden, and Alan Fekete. 2013. MDCC: Multi-Data Center Consistency. In *Proceedings of the 8th European Conference on Computer Systems (EuroSys '13)*. 113–126.

[18] Leslie Lamport. 1998. The Part-Time Parliament. *ACM Transactions on Computer Systems* 16, 2 (1998), 133–169.

[19] Leslie Lamport. 2001. Paxos made simple. *ACM Sigact News* 32, 4 (2001), 18–25.

[20] Leslie Lamport. 2006. Fast Paxos. *Distributed Computing* 19, 2 (2006), 79–103.

[21] Kfir Lev-Ari, Edward Bortnikov, Idit Keidar, and Alexander Shraer. 2016. Modular Composition of Coordination Services. In *Proceedings of the 2016 USENIX Annual Technical Conference (USENIX ATC '16)*. 251–264.

[22] Bijun Li, Wenbo Xu, Muhammad Zeeshan Abid, Tobias Distler, and Rüdiger Kapitza. 2016. SAREK: Optimistic Parallel Ordering in Byzantine Fault Tolerance. In *Proceedings of the 12th European Dependable Computing Conference (EDCC '16)*. 77–88.

[23] Yanhua Mao, Flavio P. Junqueira, and Keith Marzullo. 2008. Mencius: Building Efficient Replicated State Machines for WANs. In *Proceedings of the 8th USENIX Conference on Operating Systems Design and Implementation (OSDI '08)*. 369–384.

[24] Hicham Marouani and Michel R. Dagenais. 2008. Internal Clock Drift Estimation in Computer Clusters. *Journal of Computer Systems, Networks, and Communications* 2008, Article 9 (2008).

[25] Iulian Moraru, David G. Andersen, and Michael Kaminsky. 2013. There is More Consensus in Egalitarian Parliaments. In *Proceedings of the 24th Symposium on Operating Systems Principles (SOSP '13)*. 358–372.

[26] Faisal Nawab, Divyakant Agrawal, and Amr El Abbadi. 2018. DPaxos: Managing Data Closer to Users for Low-Latency and Mobile Applications. In *Proceedings of the 2018 International Conference on Management of Data (SIGMOD '18)*. 1221–1236.

[27] Diego Ongaro and John Ousterhout. 2014. In Search of an Understandable Consensus Algorithm. In *Proceedings of the 2014 USENIX Annual Technical Conference (ATC '14)*. 305–320.

[28] Sajjad Rizvi, Bernard Wong, and Srinivasan Keshav. 2017. Canopus: A Scalable and Massively Parallel Consensus Protocol. In *Proceedings of the 13th International Conference on Emerging Networking EXperiments and Technologies (CoNEXT '17)*. 426–438.

[29] Rainer Schiekofer, Johannes Behl, and Tobias Distler. 2017. Agora: A Dependable High-Performance Coordination Service for Multi-Cores. In *Proceedings of the 47th International Conference on Dependable Systems and Networks (DSN '17)*. 333–344.

[30] Fred B. Schneider. 1990. Implementing Fault-tolerant Services Using the State Machine Approach: A Tutorial. *Comput. Surveys* 22, 4 (1990), 299–319.

[31] João Sousa and Alysson Bessani. 2015. Separating the WHEAT from the Chaff: An Empirical Design for Geo-Replicated State Machines. In *Proceedings of the 34th International Symposium on Reliable Distributed Systems (SRDS '15)*. 146–155.

[32] Douglas B. Terry, Vijayan Prabhakaran, Ramakrishna Kotla, Mahesh Balakrishnan, Marcos K. Aguilera, and Hussam Abu-Libdeh. 2013. Consistency-Based Service Level Agreements for Cloud Storage. In *Proceedings of the 24th Symposium on Operating Systems Principles (SOSP '13)*. 309–324.

[33] Muhammed Uluyol, Anthony Huang, Ayush Goel, Mosharaf Chowdhury, and Harsha V. Madhyastha. 2020. Near-Optimal Latency Versus Cost Tradeoffs in Geo-Distributed Storage. In *Proceedings of the 17th USENIX Symposium on Networked Systems Design and Implementation (NSDI '20)*. 157–180.

[34] Hanyu Zhao, Quanlu Zhang, Zhi Yang, Ming Wu, and Yafei Dai. 2018. SDPaxos: Building Efficient Semi-Decentralized Geo-Replicated State Machines. In *Proceedings of the ACM Symposium on Cloud Computing (SoCC '18)*. 68–81.