Abstract—Replication is an important building block to achieve high availability in the presence of failures. Until recently, wide-area replication with strong consistency guarantees was regarded as impractical due to performance constraints. We investigate how informed leader election combined with a network overlay can improve the performance of distributed consensus, which is at the heart of every replicated data store. Leader election and overlay construction are particularly relevant when replicating data at global scale where network links exhibit diverse performance characteristics. We propose to incorporate knowledge about the link quality and network overlay topology into the leader election algorithm. In particular, we show how optimizing only for a quorum, instead of all replicas, we can increase replication throughput or decrease the request latency. Our measurements show a throughput increase of 1.5x when optimizing for throughput of all replicas and a 3x improvement when the throughput is optimized only for a quorum.

Keywords—leader election; overlay networks; replication; wide-area networks; strong consistency;

I. INTRODUCTION

Wide-area replication with strong-consistency guarantees is of renewed interest to the data-store community. While it is possible to replicate data across wide-area networks, the performance trade-offs were deemed unfavorable so far. Increasing the performance of any strongly-consistent, replicated data store is important. One major difference between local and wide-area networks is the link characteristics. We argue that replication across a wide-area network must take the link quality between replicas into account to achieve good replication performance. Link quality in a local-area network is more homogeneous with respect to latency, bandwidth, and packet loss, compared to a wide-area network. Because the network is no longer irrelevant when electing a leader, arbitrarily choosing a leader from a set of identical servers leads to suboptimal performance.

Recently, interest in strongly consistent data stores is re-surring. Google’s Spanner [6] and Megastore [2] both support strong consistency with replication at a global scale. Each data item is replicated across multiple data centers, ensuring data availability even if an entire data center goes offline. While Megastore guarantees strong consistency semantics only within a partition, Spanner does not have this restriction anymore. Strong consistency is desirable because it eases the burden for developers. Application-level code does not have to reconcile inconsistencies bubbling up from the data-store layer. The main challenge is to achieve reasonable performance while supporting strong consistency.

The performance penalty is in large part due to the increased network latency. At the core of each strongly-consistent data store is a distributed agreement protocol, for example, Paxos [10]. Paxos guarantees that all participants agree on the order in which updates to the data store are applied. To achieve agreement, Paxos requires multiple network round-trips between the participants for each update to the data store. In a local area network, where network links have similar performance characteristics, the choice of Paxos leader has little impact on the performance of Paxos. This is no longer the case when running Paxos over a wide-area network.

To improve the performance of Paxos in a heterogeneous wide-area network we propose to two things: (1) we elect the Paxos leader based on network link characteristics. This prevents the election of an arbitrary leader at the end of a low performance network link. (2) we construct a network overlay optimized for a subset (quorum) of the replicas. The overlay allows the replication to better cope with load surges. By intelligently placing the leader and using an overlay optimized for the current quorum members, we show that the throughput increases and latency decreases compared to our uninformed baseline. The proposed performance improvements to Paxos will directly benefit every replicated data store depending on it.

To summarize the contributions of our work:

- We identify and explain why leader election and quorum topology in the context of wide-area replication are important factors for the system’s overall performance.
- We propose the use of (1) informed leader placement and (2) quorum overlay construction to improve the system’s performance.
- We present strategies for overlay construction and leader placement.
- We evaluate the performance improvement in term of
II. RATIONALE

Our work focuses on strongly-consistent data stores\textsuperscript{[6, 2]} and coordination services\textsuperscript{[4, 9]}. At a general level, these systems provide an interface to read and write data. For reasons such as fault tolerance, the data is replicated among an ensemble of servers, also called replicas. Replicating data over a wide-area network has two immediate benefits: first, having replicas close to the clients boosts read performance of systems that support single-replica reads. Second, replicas can be placed in different geographical regions to reduce the chance of dependent failures caused, for example, by natural disasters. Most recently, hurricane Sandy hit the US East Coast in October 2012 disabling multiple data centers simultaneously. However, deploying replicas across wide-area networks also poses additional challenges. Among them are increased latency, jitter, and reduced bandwidth. Whereas latency within a data center is typically on the order of 100s of microseconds, it increases to 10s of milliseconds across data centers. Jitter is usually also higher for wide-area links. The maximum bandwidth between two data centers is less than what is available within a data center. Even more important than absolute numbers is the heterogeneity of wide-area network links. Choosing a leader arbitrarily from a set of candidates within a data center does not negatively impact performance. This is no longer true for wide-area networks.

By recognizing and building on the network heterogeneity our goal is to improve the data store’s performance. We start by identifying two main building blocks used in real-world replication: (1) leader election and (2) broadcast. The leader election algorithm forms an integral part of every distributed agreement protocol, of which the most well known is probably Paxos\textsuperscript{[10]}. The leader serializes all updates and informs the replicas of the total order by broadcasting all updates to them. We propose to adapt the leader election and broadcast primitives for WANs to improve replication performance.

A. Motivating Examples

To illustrate how we intend to optimize the agreement protocol building blocks, we continue with a motivating example. We start with a broadcast primitive which makes use of a carefully constructed overlay. Figure 1 depicts a scenario with five replicas distributed across three regions. Region 2 contains the replication leader, $s_1$, and has a total outgoing bandwidth $B$. If replica $s_1$ replicates to $s_2$, $s_4$, and $s_5$, the maximum replication throughput is $B/3$ at best. By introducing an overlay and letting $s_5$ replicate the data to $s_4$ on behalf of $s_1$, the replication throughput increases to $B/2$. Only two copies are sent across to other regions. An overlay is also beneficial when the bandwidth between regions 1, 2, and 3 is different altogether. For example, if a high bandwidth link between region 1 and 3 exists, sending updates to region 3 via region 1 may be preferable.

The overlay can be optimized with respect to other metrics too. For example, an overlay constructed to reduce the average request latency may look different than an overlay optimized for throughput. We conclude that a naive broadcast, where the leader sends updates directly to all replicas, is, in general, suboptimal. It is thus desirable to parametrize the broadcast with a cost metric. Using the cost metric the overlay messaging service can adapt itself according to the environmental parameters. The overlay is also beneficial to unicast messages. One example for unicast messages in agreement protocols are the acknowledgment messages sent by the replicas to the leader.

Optimizations based on link characteristics are also possible for the leader election service. Becker et al.\textsuperscript{[3]} have shown how the leader election influences the response time for cases in which the request distribution is skewed. In our example, if all requests are issued at replica $s_1$, replica $s_1$ is a better leader than, for instance, replica $s_2$. In this work, we argue that leader election and overlay construction must be done in unison, as they will mutually influence each other.

B. Problem Statement

To summarize, in this paper we propose two complementary services to improve the performance of wide-area replication:

1) An overlay messaging service optimized for replication and configurable with user-defined optimization goals. The overlay messaging service only has weak delivery guarantees since agreement algorithms, e.g., Paxos, retry transmissions after a timeout.

2) A broadcast-aware leader election service which is also configurable with additional optimization goals, e.g., the request distribution.

\textsuperscript{[6]}\url{http://arstechnica.com/information-technology/2012/10/hurricane-sandy-takes-data-centers-offline-with-flooding-power-outages/}

\textsuperscript{[2]}See PingER project for end-to-end performance of links in wide-area networks – \url{http://www-iepn.slac.stanford.edu/pinger/}
Algorithm\[1\] shows pseudo-code for the replication leader election. The code is executed by each of the $n$ replicas. The election progresses in rounds, where the current round for each replica is denoted by $r$. A new round is started either because $2\delta$ time units elapsed since the last round (Line\[20\]) or the replica receives a message indicating a higher round number (Line\[22\]). A new round is started by invoking the $\text{startRound}()$ procedure. The coordinator $c$ for each round is determined by the simple equation $k \mod n$, i.e., the coordinator $c$ is the round number $k$ modulo the number of participating replicas $n$. Hence, the coordinator role passes between replicas in a round-robin fashion. Each replica that starts a new round warns the following coordinator with a $\text{start}$ message (Line\[8\]). Until the replica hears back a $\text{lead}$ message from the coordinator, the replica assumes the replication leader is unknown (Line\[10\]). In regular intervals the coordinator determines the replication leader by calling out to a $\text{getBestLeader}()$ function and informing all other replicas of the result (Line\[14\]).

The functionality provided by $\text{getBestLeader}()$ relies on the performance metrics gathered by the replicas. All slave replicas periodically send their local performance observations to the coordinator (not depicted in the algorithm). The coordinator aggregates all the local observations and combines them to a global view, which is subsequently disseminated by the coordinator to the slave replicas.

Global view construction involves the elimination of crashed and unavailable replicas. The (un)availability of a replica is determined by a failure detector (see Figure\[2\]).

B. Overlay Messaging Service

Electing a replication leader is only one part of our optimized replication system. Overlay-based message routing is the other. The overlay messaging service exposes two functions: $\text{ucast}()$ and $\text{bcast}()$. The former is used for sending a unicast message, the latter for sending a message to all replicas. In the case of replication, the functionality to send messages to all replicas is only required by the replication leader. The unicast message interface is available to and used by all replicas. It is, for example, used to send acknowledgment messages.

For both message types, the message’s path to the destination is dependent on the global view and the user-defined optimization function. Also note, that the path taken by a unicast message may very well be different from the path taken by a broadcast message to the same destination. For unicast messages the optimal route is a finite, acyclic path with the sender as the start vertex and the receiver as the end vertex. In the case of a broadcast message, however, the optimal route corresponds to a tree. This is the key idea behind our overlay approach: instead of the leader sending $n-1$ unicast messages, one to each replica, replicas may forward messages on behalf of the leader. In this way, we
make the heterogeneity of wide-area networks work to our advantage.

Based on the global view each replica can calculate an optimized path to send messages to other replicas. Internally, the path is calculated by generating all trees with the sending node as the tree’s root. The tree’s edges are annotated with the performance metrics, e.g., latency or bandwidth. The user-defined optimization function \( F_3 \) is applied to each tree to determine the minimal-cost tree. Calculating the minimal cost tree is fast because the number of possible trees is bound by the number of replicas. The number of replicas in a typical deployment is seven or less [4].

C. Optimization Algorithms

This section covers how the user-defined optimization functions interact with the overlay messaging and election service. The optimization functions \( F_0 \) and \( F_e \) take as input the global-view gathered by the coordinator. Calculating the optimal path for a unicast message involves enumerating all possible acyclic paths from the source to the destination. The user-defined function \( F_0 \) assigns a cost to each of the possible paths. The message is sent along the least-cost path.

For broadcasting messages, the choice is between trees not paths. A cost is computed for every possible tree with the sender as the root node. The least-cost tree wins.

An optimization function may, for example, look at the average path latency and select the path with the minimal average latency. Alternatively, the least congested path may be taken instead. It is also possible to combine different metrics and weigh them. In this study we restrict ourselves to some common cost functions involving latency, bandwidth and request rate. Due to space constraints, we only sketch their implementation below.

1) Bandwidth: The user-defined function \( F_0 \) calculates the best tree for a given replica as follows. For each replica \( s \), calculate all its possible trees excluding crashed replicas. For each tree, find the link with the least bandwidth and mark the tree with this value. Select the tree with the highest marked bandwidth as the best tree for replica \( s \). This bandwidth represents the maximal bandwidth that \( s \) has to reach all replicas. If replica \( s \) performs a broadcast, then this tree will be used. The user-defined function \( F_0 \) selects as replication leader the replica with the highest marked tree.

2) Quorum-Bandwidth: The replication algorithm is only concerned with propagating the information to a quorum of replicas. By only considering the members of a quorum when constructing the messaging overlay, it is possible to extract even more performance out of the underlying network. Instead of propagating the data according to the cheapest tree involving all replicas, only the trees involving a majority – i.e., a quorum – of replicas are enumerated. The cheapest tree containing a majority of all replicas is used for disseminating the replicated data. A cheapest tree is also computed for the non-quorum replicas. The data must be replicated to the remaining replicas after all, so we might as well do it efficiently. The bandwidth computed for each tree represents the maximal bandwidth to a quorum. \( F_0 \) selects the best tree for each replica based on the highest quorum-bandwidth, and \( F_0 \) selects the replica with the best tree as the next replication leader. Note that maximal quorum-bandwidth can only be achieved for a short period of time, for example, load spikes. Using this bandwidth for prolonged time periods not only results in diverging replicas, but also in an unstable system once the buffers fill up.

3) Least Latency: \( F_0 \) calculates for each replica \( s \) all its possible trees excluding crashed replicas. For each tree, we calculate the latency from the root to every other replica in the tree. The latency to reach a replica in a multi-hop path is the sum of the edge values along the path. The highest latency value is the tree’s cost. The cost represents the time a request takes to reach every replica. \( F_e \) selects as replication leader the replica with the least-latency tree. Optimizing only for a quorum yields the same tree as optimizing for all replicas. For a quorum the tree’s cost is the \( k \)th element in the sorted list of latencies. Although the \( k \)th element may be a different latency value, i.e., it is faster to reach the replica represented by the \( k \)th entry than the replica represented by the last entry, the relative order of the trees does not change.

Algorithm 1. Informed Leader Election algorithm for every replica \( p \).

1. procedure getLeader()
2.  | return leader; \hspace{1cm} – return replication leader
3. init
4.  | invoke startRound(0);
5. procedure startRound(\( k \))
6.  | \( c \leftarrow k \mod n; \) \hspace{1cm} – set new coordinator
7.  | if \( p \neq c \) then
8.  | \( r \leftarrow \text{send}(\text{start,} k) \text{ to } c; \)
9.  | \( r \leftarrow k; \) \hspace{1cm} – set round number
10. leader \( \leftarrow 1; \hspace{1cm} \) – replication leader unknown
11. reset timer;
12. on every \( \delta \) time units do
13.  | if \( p = c \) then
14.  | \( \text{leader} \leftarrow \text{getBestLeader}(); \)
15.  | \( \text{send}(\text{lead,} r, \text{leader}) \text{ to all;} \)
16. upon receive \( \langle \text{lead,} k, q \rangle \) with \( k = r \) do
17.  | reset timer;
18.  | \( \text{leader} \leftarrow q; \) \hspace{1cm} – set replication leader
19. upon timer > \( 2\delta \) do
20.  | invoke startRound(\( r + 1 \));
21. upon receive \( \langle \text{lead,} k, q \rangle \) or \( \langle \text{start,} k \rangle \) with \( k > r \) do
22.  | invoke startRound(\( k \));
4) Request Rate and Combinations: As a final optimization function, \( F_e \) selects the replica that receives most requests as next replication leader.

Note that the functions presented in this section can be combined to break ties. For example, if two replicas have an equally good bandwidth-optimized tree, selecting the replica which has the highest request rate might reduce the request latency perceived by the clients.

IV. Experimental Evaluation

In our experimental evaluation we address the questions (1) whether systems using informed overlay and election services perform better than systems using uninformed services; (2) whether latency-optimized services can indeed reduce the 99th percentile of response times; (3) whether bandwidth-quorum-optimized services can achieve a higher throughput than bandwidth-optimized; (4) whether bandwidth-optimized services are more robust to load variation than latency-optimized; (5) and finally, whether combining request rate with other optimizations can reduce the response time perceived by the clients.

A. Implementation Details

For our evaluation we implemented the leader election and overlay messaging services in C++. Our prototype uses libev (http://libev.schmorp.de) as a general event notification framework and for asynchronous network communication. Our messaging overlay service optimizes either for throughput or latency using the optimization algorithms described in Section III-C. In the case of throughput we opt for the highest throughput possible, whereas we minimize in the case of optimizing for latency. Our leader election service can optimize the leader election based on request rate at each replica, i.e., the replica receiving most request is elected as the replication leader. Moreover, it can combine the rate optimization with the latency optimizations.

At the application layer we use the popular Redis key-value store (http://redis.io) for our benchmarks. Redis belongs to the class of weakly consistent data stores, i.e., Redis favors availability over consistency. Because we are interested in strongly consistent data stores distributed across wide-area networks, we enhanced Redis to also support strong consistency. Our strongly consistent Redis replicates data in accordance with the multi-instance Paxos protocol [10]. To ease the engineering effort, the implementation is constrained to the fault-free case. This does not compromise the validity of our results, as our optimizations only target the common, fault-free execution anyway. Our Paxos implementation utilizes our leader election and overlay messaging service.

To demonstrate the benefit of our leader election and overlay messaging service we compare them against a baseline configuration. The baseline configuration uses an uninformed, rotating leader election and direct connections between the leader and replicas.

B. Workload Generator

To drive load against our modified Redis implementation we use the Yahoo Cloud Serving Benchmark (YCSB) [5]. YCSB has standardized benchmarks with different mixes of read, update, and insert operations. Out of the standard mixes, we use four workload mixes, namely A, B, D, and E. We excluded workload C from our measurements because it is a read-only workload. Only if the workload consists at least partially of update operations will our improvements be noticeable in the benchmarks. For details on the standard workload mixes and their random distributions the reader is referred to Cooper et al. [5]. Since each workload issues requests with different average sizes, we present our results in k.ops/s, i.e., 1000s of operations per second.

C. Setup and WAN Emulation

Our experiments were performed on a 5-computer cluster, each computer equipped with 8-core Intel Xeon CPUs and 8 GB of RAM. All nodes are connected via Gigabit Ethernet (1000BaseT full duplex). The nodes run Debian Linux 5.0 with kernel 2.6.32. Each computer runs a single replica and multiple YCSB clients.
We used a fully connected mesh between the computers as our network topology for the benchmarks. The bandwidth and latency setting for each link are shown in Figure 3. netem, the Linux network emulator, enabled us to emulate wide-area network properties. With netem we can control the bandwidth and latency individually for each link.

D. Optimizing for Bandwidth

In this experiment we demonstrate the effects of optimizing the messaging overlay for throughput. Each replica has five clients connected to it. The baseline configuration uses node 1 as the replication leader. Based on the topology and the link performance information the informed leader election also selects node 1 as the replication leader. The throughput is constrained by the links on node 1: the highest bandwidth link for node 1 transmits 40 Mbps in our example. Because all nodes would have the same maximal throughput if selected as replication leader, 40 Mbps, the algorithm chose the node with the smallest id. Figure 4 shows the overlay chosen when optimizing for bandwidth. The replication tree’s bandwidth is constrained at 40 Mbps by the link between node 1 and 2. The overlay from node 2 to the remaining nodes is just one of many possible configurations; it does not impact the overall replication throughput as the bottleneck is between node 1 and 2. Overlays which distribute data over the link between node 1 and 3 are discarded because they offer a smaller throughput than the overlays involving the link between node 1 and 2.

Figure 6 shows the aggregated throughput, measured in operations per seconds (ops/sec), over all clients. For each of the four workloads we observe an increase in throughput when going from an uninformed leader election to an informed leader election. Even more dramatic than the throughput increase with an informed leader election is the increase associated with a quorum-optimized leader election. For workload A, which has the highest update rate among all workload mixes, the throughput increases by almost 3x when optimizing for a quorum. The other workloads exhibit a more modest improvement of around 2x between overlay-optimized and quorum-optimized. The quorum-optimized tree is shown in Figure 5. Replicas \( s_2 \), \( s_3 \), and \( s_5 \) are connected with a bandwidth of 350 Mbps and form a majority in an ensemble of five replicas.

A replication system optimized for throughput gains flexibility during times of short load surges. Requiring a replication bandwidth higher than the minimum bandwidth link for prolonged time periods results in diverging replicas. In our example, \( s_1 \) and \( s_4 \) will fall behind the replicas with a higher connection bandwidth. Once replication load falls below the minimum bandwidth, in our example 20 Mbps, lagging replicas can catch up again. In our experiments, YCSB cannot sustain the maximum throughput during the complete execution since the buffers eventually fill up and produce back pressure. Therefore, the quorum-optimized overlay does not present aggregated throughput values 17 times larger than the bandwidth-optimized – as possibly expected since, \( 350 / 20 = 17.5 \).

E. Optimizing for Latency

We now investigate how the overlay changes for the example in Figure 5. The algorithm for latency-based overlay optimization was described in Section III-C. When the overlay is optimized for latency, node 2 becomes the replication leader. Figure 7 shows the 99% percentile latency observed during our benchmark runs. We observe that the 99% percentile latency decreases for each workload. For example, the latency for workload mix A decreases from

---

1. [http://www.linuxfoundation.org/collaborate/workgroups/networking/netem](http://www.linuxfoundation.org/collaborate/workgroups/networking/netem)
50 ms to 37 ms. Optimizing only for a quorum gives the same results as optimizing over all replicas. The values assigned to each tree are different, but the order imposed on the trees by the cost metric stays the same. There are no additional gains over a latency-optimized overlay. The latency for workload D is twice as other mixes due to the larger size of its requests.

F. Robustness to Load Variation

The last set of experiments shows how different overlay optimizations affect the system under varying load conditions. In Figure 8 and Figure 9 the overlay is either optimized for bandwidth or latency as noted in the legend. The x-axis tracks the target load, while the y-axis alternatively plots the achieved throughput or 99th percentile latency. The bandwidth-optimized overlay is able to sustain a higher update rate than the latency-optimized overlay. Up to about 500 operations per second both overlays perform identical (Figure 8). The latency-optimized overlay, however, does not scale beyond 500 ops/sec, whereas the bandwidth-optimized overlay achieves up to 1500 ops/sec. The latency optimization only shows its benefits at low load levels: up to 200 ops/sec is the 99th percentile below that of the bandwidth-optimized tree. For higher request rates, it is actually higher than for the bandwidth-optimized tree. If we switched the optimization target dynamically based on the perceived load this behavior could be averted. That is, switching from a latency-optimized tree to a bandwidth-optimized tree when load grows beyond a certain threshold.

G. Optimizing for Connection Count

A third optimization criteria, besides bandwidth and latency, can be the request rate at each replica. Uneven request rate distributions are, for example, a result of the shifting geographic location of active users, e.g., from Asia to Europe to the US and back to Asia. To demonstrate the effect of an uneven request rate distribution between replicas, we instantiated a setup with five replicas. Four replicas had each one YCSB instance connected while the fifth replica had three YCSB instances connected.

The consequence of blindly using the request rate is visible in Figure 10. Designating the replica with the most clients connected to it as the leader impacts the 99th percentile latency negatively. Switching to an informed leader election which combines the link latency and request rate to decide on a leader, the 99th percentile latency is less than that of the uninformed leader election.

V. RELATED WORK

Replication has been optimized by using communication topologies different than the traditional star. Examples of such optimizations are the works on Ring Paxos by Primi et al. [11] and on Chain Replication by Van Renesse and Schneider [13]. These works have proposed new replication algorithms focusing LAN and either rely on IP-multicast primitives or assume servers have homogeneous performance – in WANs however neither IP-multicast is available nor homogeneity can be assumed. In our work we have proposed no new replication algorithm, but instead we have presented two general services that can be used by a variety of replication algorithms in WANs.

E lecting a good leader for replication in WAN has been studied by Becker et al. [3]. In their work, they propose a bullying leader election configurable with user-defined scores. Instead of using their algorithm, we decided to modify a simpler algorithm: the rotating leader election as described by Aguilera et al. [1]. The same algorithm has already been modified to elect an optimal leader by Nuno et al. [12]. Nevertheless, their algorithm exclusively focus on latency minimization providing no configurability.

Overlays have been applied in WANs for content distribution. Ganguly [7], for example, introduces the approach of performing replication of content with the help of such an
overlay. Besides focusing a different problem, the approach in our work allows the overlay to be configured with different goals than bandwidth optimization. The authors are unaware of overlay approaches in the context of strong consistency over WANs. We believe that systems such as Google’s Spanner [6] and Megastore [2] could exploit the approach proposed in our work without redesigning their replication algorithms.

VI. CONCLUSION

Wide-area replication for data stores is important to ensure availability even if an entire data center goes offline. While replication across a wide-area network is possible the challenge is to achieve reasonable performance. A major difference between replication within a data center and across data centers is the network’s heterogeneity. We presented a system design which takes the network link diversity into account when electing a leader. Our replication leader also uses an optimized network overlay to broadcast updates. While the optimization criteria are user-configurable, we used common metrics such as bandwidth and link latency for the overlay construction. One key insight is, that the overlay may be optimized differently if only a quorum of nodes is optimized for. Optimizing only for a quorum makes the replication system more robust to load surges. Our evaluation showed how an informed leader election combined with a message overlay outperforms the uninformed baseline system.

ACKNOWLEDGMENTS

This research has been funded in part by the European Community’s Seventh Framework Programme [FP7/2007-2013] under the ParaDIME Project (www.paradime-project.eu), grant agreement no. 318693.

REFERENCES


