Secure and Resilient Data Replication for the Client-centric Decentralized Web

Kristof Jannes
imec-DistriNet, KU Leuven
Leuven, Belgium
kristof.jannes@kuleuven.be

Abstract

Part of the web is shifting to a client-centric, decentralized model where web clients become the leading execution environment for application logic and data storage. However, the basic paradigm of the web and the browser is still server-centric. The key data is stored, served, processed and analyzed on central servers owned by the service provider. This means that a good internet connection is required, otherwise the application will be slow, or even unreachable when there is no internet. Furthermore, user data can be stolen, modified, deleted or sold by the service provider itself, or by an external attacker targeting the service provider.

We propose three middleware solutions for a client-centric web paradigm where clients also become authoritative copies of the data. Data can be replicated peer-to-peer between many clients and servers. This kind of distributed and decentralized web application architecture should be supported by both the browser and client-side web middleware. The middlewares differ in their consistency and trust model.

CCS Concepts: • **Security and privacy** → **Distributed systems security**; • **Networks** → **Peer-to-peer protocols**; • **Information systems** → *Web applications.*

ACM Reference Format:
Kristof Jannes. 2022. Secure and Resilient Data Replication for the Client-centric Decentralized Web. In *Proceedings of 23nd International Middleware Conference Doctoral Symposium (Middleware '22 Doctoral Symposium).* ACM, New York, NY, USA, 5 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 Motivation and related work

The computing landscape has changed dramatically over the last 50 years. It started with a centralized model where all computations are done on a central mainframe controlled by thin clients. Later, these clients became more powerful and computation shifted to these client devices. With the rise of cloud computing and Software-as-a-Service offerings, the computations shifted back to central servers. Cloud computing is beneficial for users in terms of availability, durability, security and collaboration. However, lots of data, and therefore power, is given to few large tech companies and governments, and end-users are losing control over their data. Furthermore, a good internet connection is required, otherwise the application will be slow, or even unreachable.

With the advent of mobile networks, offline-first software [13], and distributed ledgers, the landscape is once again changing to a more decentralized and client-centric model [7]. In this new paradigm, clients also become authoritative copies of the data. We will focus on this paradigm change in web applications, as many applications these days run in the browser. Even seemingly native applications, are often just a wrapper around a web browser. The introduction of fast mobile networks (5G) can help to improve reliability and performance, but does not allow to drop the need for a robust, offline-first approach. Network availability is still sparse in airplanes, tunnels and remote areas. On a global scale, latency is limited by the speed of light.

We propose three different protocols, implemented in three browser-based middlewares, to support shared data editing in such new architecture by using replication across two different dimensions: *trusted* and *untrusted* environments, as well as *eventual consistency* and *strong consistency.* We focus on collaborative web applications, where multiple users, with multiple clients, are working concurrently on a shared data set.

Use cases. The scenario with eventual consistency in a trusted setting is the default for collaborative applications such as shared document editing. Multiple users are editing the same document, potentially at the same time. Some users might be offline, e.g. in an airplane, yet they can still continue to edit the document and their changes will be replicated later when they come back online. We have two use cases in an untrusted setting. One use case is an integrated loyalty program with several merchants at a farmer's market or small shopping street [8]. Rather then relying on a trusted third party, it should be possible to setup a peer-to-peer network between them to make sure no customer sends their loyalty points twice, and that no merchant tries to cheat the

Middleware '22 Doctoral Symposium, November 7–11, 2022, Quebec, Canada

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM...$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

system. This requires strong consistency. The last use case is similar to the first one of shared document editing, but for environments where trust is lacking. This can be distrust against other users in the peer-to-peer network, but also distrust against the service provider hosting the server.

Related work. Existing collaborative frameworks in a trusted environment, such as Google Docs, use a central server to coordinate concurrent operations using Operational Transformation (OT) [6]. The server will transform concurrent operations for each client so they can be applied out of order. Using a central server to handle conflicts does not allow extended offline usage, and scalability and reliability are dependent on that server. Conflict-free Replicated Data Types (CRDTs) [15] are data structures that guarantee eventual consistency without explicit coordination. There are three different categories of CRDTs. Operation-based CRDTs [12] are similar to OT in the sense that replication is done by replicating the operations. However, concurrent operations must be commutative. This removes the dependency on a central server, and makes offline use possible. However, operation-based CRDTs need to keep track of the state of every replica. For example, by using a vector clock with one entry per client that has ever made an edit. This means that the total size of the metadata grows without bound over time. This is especially the case for web applications, where one user typically has multiple clients, many of them which are used infrequently. State-based CRDTs solve this problem by only relying on the state and some metadata independent from the number of operations or clients. However, to replicate such a CRDT, the whole state has to be sent every time. For this reason, such state-based CRDTs are only used between servers today. Delta-state based CRDTs [1, 16] try to solve this problem by using extra metadata to calculate a small delta state between two replicas, only the delta state must be sent instead of the full state. However, to calculate these deltas, replicas have to keep track of all editing replicas, for example using vector clocks. This inherits the same problems as operation-based CRDTs, where the size of the data will grow with the number of replicas over time.

Decentralized interactions in an untrusted environment, where strong consistency is required, can be enabled by using a Byzantine Fault Tolerant (BFT) consensus protocol. A classical BFT consensus protocol, such as PBFT [5], BFT-SMART [3] or HotStuff [18], assumes a low-latency network link between all replicas, and one replica will be elected as leader to drive the protocol. This does not match the client-centric idea. In reality, the network link is a mobile or WiFi connection, and not every replica has enough resources to be elected as leader. More recent BFT protocols, such as Tendermint [4], relax the networking requirement by using a multi-hop gossip protocol between the replicas. However, the leader still remains a single point-of-failure, leading to large latencies each time a new leader has to be elected.

If an application in an untrusted environment can deal with eventual consistency, this would be a more pragmatic option. However, traditional CRDTs only guarantee strong eventual consistency in a trusted setting. One malicious replica is able to let other replicas diverge forever. Rational misbehaving replicas might be kept honest by always being able to detect they have misbehaved [17], however this does not guarantee convergence if they still choose to behave Byzantine. Kleppmann [11] showed that it is possible to make operation-based CRDT tolerate malicious replicas and guarantee strong eventual consistency. Besides keeping the replicas consistent, data should be encrypted so that not all replicas are able to read all data. This would allow to use untrusted servers as central replication point. Such a server is almost always online, and is typically more reliable then a client device. Barbosa et al. extended standard CRDTs with cryptographic protocols [2]. One last requirement is that each update should be attributable to the user that made the update. One framework supporting this is Snapdoc [14]. However, there is no framework yet that supports both strong eventual consistency in a Byzantine environment, which also provides confidentiality, integrity and attributability. Furthermore, all these approaches only work well in a mostly static environment where the set of replicas is known beforehand and is not often changed. They can be reconfigured by stopping the protocol, adding or removing a new replica, and start it again. In case such a membership change would be done online, concurrent updates to the actual data might be lost, or the membership reconfiguration has to be restarted each time a concurrent update occurs.

2 Contributions

In this section, we briefly present three proposed approaches for client-centric replication in different trust-environments and consistency models. The first approach is eventual consistency in a trusted setting, for example shared document editing where users can go offline. The second approach is strong consistency in an untrusted setting, for example an integrated loyalty points network between merchants. The third approach is eventual consistency in an untrusted setting, for example shared document editing where the central server or other clients are not trusted. We will not discuss the full protocol in detail, but focus on their achieved properties.

2.1 Eventual consistency in a trusted setting

This first scenario is both a client-server, as well as peer-to-peer network (Figure 1). Clients can be offline, but should be able to continue working and should be synchronized quickly when coming back online.

We propose a novel replication protocol for state-based CRDTs that allow replicas to use a Merkle-tree that follows the tree-shaped structure of the actual data to replicate changes in a fine-grained way. The Merkle-tree makes it

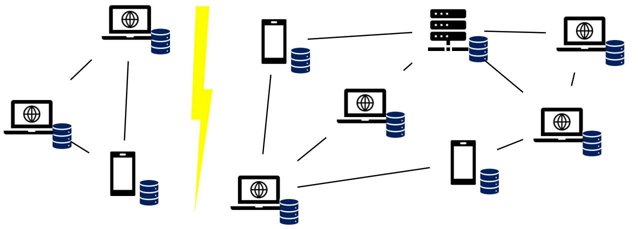

Figure 1. Peer-to-peer network of trusted devices. Each device has a local copy of the data, and can continue making changes even when network partitions occur.

possible to quickly find out the updates data items, and only replicate these. This solves the biggest problem of state-based CRDTs, namely that normally the full data set has to be sent to replicate and merge data. This way, state-based CRDTs become suitable for a client-centric deployment on mobile networks with low-bandwidth and large latency compared to server-to-server networks where state-based CRDTs are used today. The state-based approach makes the replication much more reliable and robust against temporary network failures compared to operation-based approaches. There are also no client specific identifiers required for correct replication. This means that the total size of the metadata is only a function of the actual data size itself, it does not grow with the number of client, nor with the number of operations.

We implemented this protocol in OWebSync [9, 10], a web-based middleware for data replication in interactive groupware with fast resychronization of offline clients, and continuous, interactive synchronization of other clients. Application developers can leverage OWebSync to model their application data in a JSON-tree, which will internally be mapped on a Merkle-tree and several state-based CRDTs.

2.2 Strong consistency in an untrusted setting

Some applications will require strong consistency. In a distributed setting with potentially malicious clients, and multiple readers and writers, this translates into the Byzantine Fault Tolerant (BFT) consensus problem. In such a peer-to-peer network replicas can be actively malicious (Figure 2). It also won't be possible to make updates offline, as the requirement for strong consistency obligates coordination between the replicas before an update is accepted.

We propose a novel BFT protocol that is designed to be leaderless, lightweight and robust. It does not rely on a leader, removing the need for a costly leader-election procedure when it is malicious or loses its network connection. The latter scenario is common in our target environment. The full state, including the consensus votes, is replicated by using a state-based gossip protocol similar to the protocol from the previous section. A major feature of gossip-based communication is its reliability. Thanks to the state-based nature, replicas that experienced short intermittent failure, can be quickly up-to-date again and continue voting.

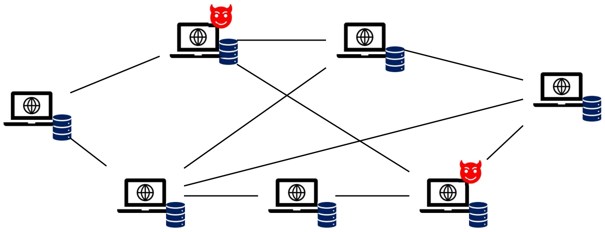

Figure 2. Peer-to-peer network of untrusted clients, reaching strong consistency via Byzantine Fault Tolerant consensus.

We implemented this protocol in a web-based middleware for decentralized BFT consensus in client-centric, community-driven web applications. It exposes a replicated key-value store to developers for which the browser replicas coordinate agreement using our novel BFT protocol. It has a similar lightweight setup and architecture as a trusted peer-to-peer data synchronization framework such as OWebSync: only a peer-to-peer discovery service and the actual browser replicas are required to setup a decentralized network.

Compared to other BFT frameworks using a classical BFT consensus protocol with a leader, it is much more robust when the network connection of the leader replica fails. There is no difference in commit latency for our protocol, while the classical protocols need 10s of seconds to repair, or even fail to recover for larger networks. The disadvantage of our protocol is the increased latency in optimal scenario's, without failures, from 100s of milleseconds for a classical BFT protocol to 1-4 seconds for our protocol. Compared to other BFT frameworks that also use a gossip-based approach to achieve consensus, our protocol confirms transactions faster, uses less bandwidth, and is more robust against failures.

2.3 Eventual consistency in an untrusted setting

For applications that do not require strong consistency, choosing for eventual consistency is the most pragmatic option. As client devices are often offline, reaching a global consensus to have strong consistency would be nearly impossible. This third scenario is a hybrid approach of a peer-to-peer network of mostly client devices and some centralized servers to improve availability and durability (Figure 3). Having some kind of centralized server can be beneficial to aid the client-centric vision. The server is most of the time online, and all clients can use this server to replicate their data to each other. Even when they are never online simultaneously.

We propose a novel, secure, state-based CRDT protocol that extends classical state-based CRDTs to guarantee strong eventual consistency, even with Byzantine replicas, and with confidentiality, integrity and availability properties. All user-data is encrypted per field in every (sub)-document, to preserve confidentiality and integrity. This allows for a dynamic membership with fine-grained key management. This protocol is the first to allow both concurrent data updates, as

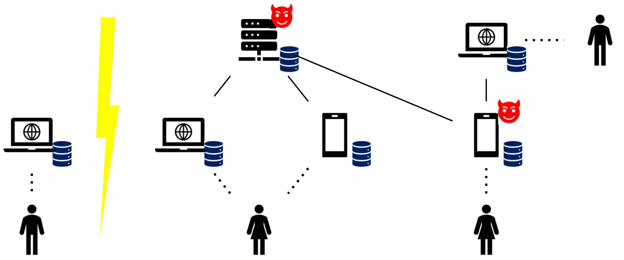

Figure 3. Hybrid architecture of a peer-to-peer network with a centralized server. Not every replica may have access to the actual plain data, and some devices can even be malicious.

well as concurrent updates to the access control policy. This means that a user can share a document, or revoke access, without losing concurrent updates to that document.

This protocol is implemented in a web-based middleware for data synchronization in interactive groupware with untrusted replicas. Similar to OWebSync, this framework allows for concurrent editing by multiple replicas, even when they are offline or partitioned. However, all data is encrypted by default and every change is attributable to the editor. Despite this fine-grained encryption, the middleware can be used for interactive, collaborative document editing. The drawback is the much higher storage space required, as the size of the data, including metadata, is increased up to a factor 19. However, this increased size is constant. The size does not grow over time, nor with the number of replicas.

Compared to the state-of-the-art, this middleware is more dynamic and fine-grained. Strong eventual consistency will be maintained, even when users are added or removed, when users are malicious, or when the access control policy changes. This means that no rollbacks are necessary and no concurrent updates will be lost, even when they are concurrent to policy changes. No explicit coordination is required between the collaborators when rotating an encryption key.

3 Conclusion

We proposed three different protocols to support data replication in distributed and decentralized web applications in different trust settings and consistency models. These protocols are implemented and evaluated in three different web-based middlewares. The middlewares can support the client-centric, offline-first architecture, giving more control to end-users over their data. Existing frameworks often lacked robustness and resilience to (temporary) network failures, offer no security guarantees, do not allow fine-grained online changes to the access control policy, or do not scale well with the client-centric nature of the decentralized web.

Acknowledgments

I would like to thank my supervisors Prof. Bert Lagaisse and Prof. Wouter Joosen for their guidance and support.

References

[1] Paulo Sérgio Almeida, Ali Shoker, and Carlos Baquero. 2018. Delta state replicated data types. *J. Parallel and Distrib. Comput.* 111 (2018), 162–173.

[2] Manuel Barbosa, Bernardo Ferreira, João Marques, Bernardo Portela, and Nuno Preguiça. 2021. Secure Conflict-Free Replicated Data Types. In *International Conference on Distributed Computing and Networking 2021 (ICDCN '21)*. ACM, USA, 6–15.

[3] Alysson Bessani, Joao Sousa, and Eduardo E. P. Alchieri. 2014. State Machine Replication for the Masses with BFT-SMART. In *Proceedings of the 2014 44th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN '14)*. IEEE, USA, 355–362.

[4] Ethan Buchman, Jae Kwon, and Zarko Milosevic. 2018. The latest gossip on BFT consensus. arXiv:1807.04938

[5] Miguel Castro, Barbara Liskov, et al. 1999. Practical Byzantine fault tolerance. In *Proceedings of the Third Symposium on Operating Systems Design and Implementation (OSDI '99)*. USENIX, USA, 173–186.

[6] C. A. Ellis and S. J. Gibbs. 1989. Concurrency Control in Groupware Systems. *SIGMOD Rec.* 18, 2 (1989), 399–407.

[7] Kristof Jannes, Bert Lagaisse, and Wouter Joosen. 2019. The Web Browser as Distributed Application Server: Towards Decentralized Web Applications in the Edge. In *Proceedings of the 2nd International Workshop on Edge Systems, Analytics and Networking (EdgeSys '19)*. ACM, USA, 7–11.

[8] Kristof Jannes, Bert Lagaisse, and Wouter Joosen. 2019. You Don't Need a Ledger: Lightweight Decentralized Consensus Between Mobile Web Clients. In *Proceedings of the 3rd Workshop on Scalable and Resilient Infrastructures for Distributed Ledgers (SERIAL '19)*. ACM, USA, 3–8.

[9] Kristof Jannes, Bert Lagaisse, and Wouter Joosen. 2021. OWebSync: Seamless Synchronization of Distributed Web Clients. *IEEE Transactions on Parallel and Distributed Systems* 32, 9 (2021), 2338–2351.

[10] Kristof Jannes, Bert Lagaisse, and Wouter Joosen. 2022. Seamless Synchronization for Collaborative Web Services. In *Service-Oriented Computing – ICSOC 2021 Workshops*. Springer, Cham, 311–314.

[11] Martin Kleppmann. 2022. Making CRDTs Byzantine Fault Tolerant. In *Proceedings of the 9th Workshop on Principles and Practice of Consistency for Distributed Data (PaPoC '22)*. ACM, USA, 8–15.

[12] Martin Kleppmann and Alastair R Beresford. 2017. A Conflict-free Replicated JSON Datatype. *IEEE Transactions on Parallel and Distributed Systems* 28, 10 (2017), 2733–2746.

[13] Martin Kleppmann, Adam Wiggins, Peter van Hardenberg, and Mark McGranaghan. 2019. Local-First Software: You Own Your Data, in Spite of the Cloud. In *Proceedings of the 2019 ACM SIGPLAN International Symposium on New Ideas, New Paradigms, and Reflections on Programming and Software (Onward! 2019)*. ACM, USA, 154–178.

[14] Stephan A. Kollmann, Martin Kleppmann, and Alastair R. Beresford. 2019. Snapdoc: Authenticated snapshots with history privacy in peer-to-peer collaborative editing. *Proceedings on Privacy Enhancing Technologies* 2019, 3 (2019), 210–232.

[15] Marc Shapiro, Nuno Perguiça, Carlos Baquero, and Marek Zawirski. 2011. Conflict-Free Replicated Data Types. In *SSS 2011 - 13th International Symposium Stabilization, Safety, and Security of Distributed Systems (LNCS)*. Springer, Berlin, 386–400.

[16] Albert van der Linde, Pedro Fouto, João Leitão, Nuno Preguiça, Santiago Castiñeira, and Annette Bieniusa. 2017. Legion: Enriching Internet Services with Peer-to-Peer Interactions. In *Proceedings of the 26th International Conference on World Wide Web (WWW '17)*. International World Wide Web Conferences Steering Committee, CHE, 283–292.

[17] Albert van der Linde, João Leitão, and Nuno Preguiça. 2020. Practical Client-Side Replication: Weak Consistency Semantics for Insecure Settings. *Proc. VLDB Endow.* 13, 12 (2020), 2590–2605.

[18] Maofan Yin, Dahlia Malkhi, Michael K. Reiter, Guy Golan Gueta, and Ittai Abraham. 2019. HotStuff: BFT Consensus with Linearity and Responsiveness. In *Proceedings of the 2019 ACM Symposium on Principles of Distributed Computing (PODC '19)*. ACM, USA, 347–356.