CRDTs as replication strategy in large-scale edge distributed system: An overview

Miloš Simić, Milan Stojkov, Goran Sladić and Branko Milosavljević
Faculty of Technical Sciences, Novi Sad, Serbia
milos.simic@uns.ac.rs

***Abstract*—Replication is one of the most important parts of any distributed system. It is mainly used to provide fault-tolerant and robust systems. Data is copied on a few different machines, and often in several regions to ensure that loss of a single machine, cluster or even region does not affect the overall quality of the service given to users. Over several decades both academia and the industry developed few techniques on how to manage these tasks. These techniques usually involve some sort of expensive synchronization or consensus to ensure replication is done properly. Conflict free replicated data types or CRDTs, do not use consensus, but a set of strong mathematical rules and they are guaranteed to converge eventually if all concurrent updates are commutative. Expensive synchronization or consensus might be a problem with resource-limited devices. In this paper, we present a review of CRDTs, it's benefits and shortcomings in the domain of the large scale edge computing systems with constant data flow with processing on the very edge of the network. We provide answers to questions is this technique applicable as a data replication strategy in systems with limited hardware capabilities, and could we retain or improve availability and reduce the latency of the system, but at the same time avoid expensive synchronization and consensus-based approach, which could be problematic due to the nature of edge computing.**

Keywords: distributed systems, big data, service-oriented architectures, cloud computing, edge computing, consensus, replication, synchronization, CRDTs

I. Introduction

Nowadays, we are facing a massive movement in processing away from the standard, centralized computing model that is provided through the cloud computing paradigm, especially in the domain of Internet of Things (IoT) where we have a lot of sensors that are constantly sensing some events.

This movement returns the distribution of computing power and logic to the edge of the network. Edge network is the idea of connecting sensors to programmable automation controllers (PAC) which can then handle processing, storage, and communication. The basic concept of edge computing is to leverage new generation technologies, processes, services, and applications that are built to take advantage of this new infrastructure. The key difference with this model is that it operates and it is deployed on computing hardware closer to the edge of the network. Thus, it is bringing the cloud computing model closer to the ground.

With the architecture of large scale distributed edge devices [1], we also need to consider how to manage data replication on those devices to ensure resilience, consistency, and fault tolerance. Replication is a standard and must-have in any large-scale distributed systems and the cloud as well [2][3]. We can investigate, are the same strategies and techniques applicable in this new architecture? Can their properties benefit the edge computing model? Or we might need to consider new research breakthroughs in replication areas and embed them in the edge computing model?

In the large distributed systems such as the world wide web, peer-to-peer networks, or even cloud computing platforms, replication and consistency are essential features. They are mainly used to provide fault tolerance and data consistency. Data replication is a process that usually relies on expensive synchronization or some consensus protocol that runs in between replicas. When the majority of those replicas store a copy of the data, only then we can proceed with the response.

Two replication strategies have been used in distributed systems: 1) active replication where each client request is processed by all the servers and 2) passive replication where only one server at the time processes client requests. After processing a request, the primary server updates the state on the other servers and sends back the response to the client.

In traditional datacenters when talking about replication, we are usually thinking about strong consistency, and two-phase commit [2] is a commonly used algorithm. The strong consistency approach provides us a guarantee by serializing updates to the replicas in global total order. The downside of the strong consistency is that it is a performance and scalability bottleneck, and it also conflicts with availability and partition-tolerance due to the CAP theorem [3][4]. Strong consistency requires more network hops to ensure agreement. The application using such a system will not be able to read the data before the last operation. When network delays are large or partitioning is an issue as in delay-tolerant networks, disconnected operation, cloud computing, or peer-to-peer systems, the eventual consistency model promises higher availability and better

performance. Concurrent updates may conflict; conflict arbitration may require a consensus and a roll-back [5]. The main advantage of the eventual consistency model is that the system will be available to perform write operations even when the network connectivity between replicas is down [6][7]. This model gives us better availability and performance [8][9], avoids the round-trip time of strong consistency, and supports far more write operations per second than the other models. A problem that eventual consistency must address are conflicts in the data. Conflict-free replicated data types (CRDTs) model resolve conflicts by using some simple mathematical properties.

Research questions that we are trying to address are: 1) can we provide replication in the edge computing without extensive synchronization or consensus between replicas, 2) what are the benefits, and shortcomings of CRDTs in large scale edge computing, 3) can we make replication event-driven and asynchronous and 4) can we propagate replication over existing infrastructure like broadcast, gossip, spanning trees, etc. and avoid sending more data over the network.

This paper is organized as follows. Section II presents related work. Section III presents CRDTs designs and principles. Section IV describes the possible usage of CRDTs as a replication strategy in large scale edge computing distributed systems. Section V summarizes conclusions and briefly proposes ideas for future work.

II. Related work

NearCloud [10], run customer business logic on CDN edge and let these functions read and write customer data in real-time. They invested a lot of energy in the CRDTs data layer to achieve low-latency, dynamic web processing. Customers use NearCloud to migrate delay-sensitive applications from the cloud to the edge and turn them into global real-time applications [11].

TomTom users, having personal navigation devices, smartphones, MyDrive website accounts, expect their navigation information to be synchronized properly even in the occasional absence of network connection. Conflict-free Replicated Data Types (CRDTs) provide robust data structures to achieve proper synchronization in an unreliable network of devices. They enable the conflict resolution being done locally at the data type level while guaranteeing the eventual consistency between replicas [12].

Fastly cloud computing services provider is an edge cloud platform that uses CRDTs as its storage layer [13] and provides content delivery network, internet security services, load balancing, video and streaming services.

In academia, CRDTs started to take more and more space. In [14] authors explore the different alternatives that have been used to propagate updates to geo-replicated CRDTs on a large scale and in [15] authors present a state of the art geo-replicated store based on CRDTs. In [16] authors explore CRDTs in a different peer-to-peer environment such as collaborative editing and present an algorithm for a JSON data structure that automatically resolves concurrent modifications and all replicas converge towards the same state. In the same research authors show that every JSON data type can be converted to CRDTs, and combining them we can create more complicated types.

III. Conflict-free replicated data types (CRDTs) designs and principles

1. Principle aspects

Conflict-free replicated data type (CRDT) is a data structure that can be replicated over multiple machines in the network, and replicas can be updated independently and concurrently without coordination between them. It is always mathematically possible to resolve conflicts if they occur [17][18].

As a concept, CRDTs were defined in 2011 by Marc Shapiro and colleagues. It is initially developed for collaborative text editing and mobile computing purposes [19]. In recent years, CRDTs have gained more and more popularity because of their desirable properties. Today, they have been used in online chat systems, online gambling, and the SoundCloud audio distribution platform. Even NoSQL distributed databases such as Redis, Riak, and Cosmos DB have embedded CRDT data types.

Conflict-freedom ensures safety and liveness despite any number of failures. It leverages simple mathematical properties that ensure the absence of conflict, i.e., monotonicity in a semi-lattice and/or commutativity [5]. This means that CRDTs can be updated without expensive synchronization or consensus, and they are guaranteed to converge eventually if all concurrent updates are commutative. To ensure that concurrent updates are commutative, data has to satisfy certain mathematical conditions. The data structure must be:

1) *Commutative*: $a * b = b * a$
2) *Associative*: $(a * b) * c = a * (b * c)$
3) *Idempotent*: $(a * a) = a$

Where $*$ is a binary operation, for example, *max, union, or.* Commutative and associative properties of CRDTs ensure that operation or changes can be made out of order. While the idempotent function ensures that if we merge state with itself, we will get the same state. With these properties, CRTDs give us *strong eventual consistency* with the ability to automatically resolve conflicts. Therefore, we do not require a consensus protocol. On the other hand, *strong consistency* requires a real-time consensus protocol to solve conflicts and allow n/2-1 nodes to be down. This idea is at the root of the CAP theorem problem. If we loosen the *strong consistency* requirement, then s*trong eventual consistency* satisfies all properties in the CAP theorem.

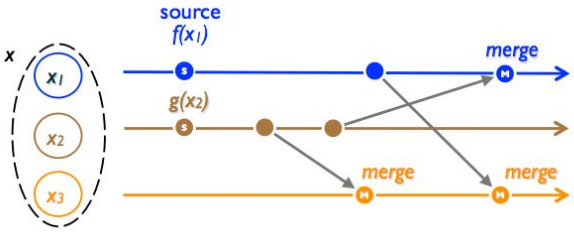
CRDTs have their disadvantages as well. The conflict resolution for the CRDT-based approach is predefined and cannot be overridden. The second problem is that not all of the possible update operations are commutative, and so CRDTs cannot be used for all problems. The third possible disadvantage is that CRDTs data types usually require more space for inner structure.

2. *Design aspects*

In an unstable and often partitioned distributed system, users want a set of base data types that can resolve conflicts for them. On the other hand, the system does not need to interact with a user or query an arbiter node. Both of these properties are promised by CRDTs design.

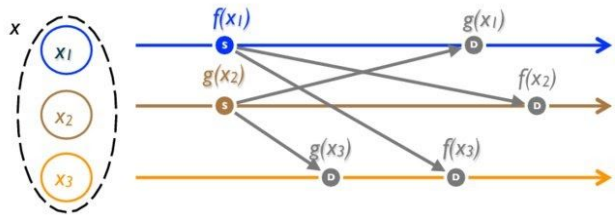
In their papers Shapiro et al. [5][20] consider two models of replication in an *eventually consistent* distributed system, based on the model of replication: 1) state-based or CvRDT (convergent replicated data type) and 2) operation-based or CmRDT (commutative replicated data type).

In state-based (or passive) replication, an update occurs entirely at the source, then propagates by transmitting the modified payload between replicas [20] Picture 1. When a replica receives an update from a client, it will first update its local state, and then after some time, it will send its full state to another replica in the system. So from time to time, every replica will send its full state to some other replica in the system. A replica that receives the state of another replica, applies a *merge function* to merge its local state with the state it just received. Similarly this replica also occasionally sends its state to another replica, so every update eventually reaches all replicas in the system. Merge operation has to be idempotent, associative, and commutative. A replicated object satisfying this property.



Picture 1. State based CRDTs [20].

Alternatively, to the state-based style, a replicated object may be specified in the operation based (or op-based) style [20] Picture 2. Operation-based replication approach is a bit different. A replica doesn't send its full state to another replica because the state could be large. Instead, it just broadcasts the update operation to all the other replicas in the system and expects them to replay that update. This process is similar to state machine replication [21]. Replicas can converge if these updates are *commutative*. The order of these updates is not important. Updates are applied to a replica and the resulting state will be the same.



Picture 2. Operation based CRDTs [20].

IV. CRDTs AS A REPLICATION STRATEGY IN EDGE COMPUTING

The system we propose in this paper is purely on the conceptual level, but it is influenced by few technical achievements: 1) the general availability of CRDTs [22] in Riak distributed database, 2) availability with Active-Active Multi-Region Deployments in the Redis database. Also a few other companies like tomtom, bet365, and PayPal, use CRDTs in production for scalability and performance.

Edge computing is moving centralized computing architectures to a more decentralized approach. With new applications like the internet of things, where we can collect and preprocess data at the very end of the network, we should find a new way to address these new applications. In [23] it is theorized that multiple ARM devices can be connected and form a micro datacenter that is logically separated into one or more clusters of nodes. This datacenter will mimic a real datacenter and could be combined into regions. In these micro datacenters users can run their own edge applications. These applications need to be specifically designed for edge computing, and they will filter and preprocess data before sending it to the cloud. Edge applications come in few forms like 1) streams, that should continually do some processing on data as it is arriving(long-running jobs), 2) standard batch jobs that do some batch processing over some collection of data and 3) reacting only if some value passes some user-defined threshold in the form of events. These applications should be run in some restricted environment like containers [24], to prevent system starvation.

Edge computing model could be used to help cloud computing by filtering and preprocessing data at the edge of the network, and only send valuable data to the future analysis. This accelerates time to market for the users and also save money on storing unnecessary data. To preprocess and filter data especially in batch applications we need to store data, and to ensure fault tolerance by replicating it on other nodes in the cluster.

Organizing edge computing to regions and clusters yields some form of replication and synchronization between nodes. In the distributed systems, replication is mainly used to provide fault tolerant and robust systems. The data replication usually relies on expensive synchronization or consensus between replicas. In traditional datacenters when using replication we are usually talking about strong consistency, as described in Section II. Consensus algorithms rely on a strong consistency approach. The strong consistency approach gives us a guarantee by serializing updates to the replicas in global total order [26]. Strong consistency has its downsides and problems, and it is usually a performance and scalability bottleneck. Also, it conflicts with availability and partition-tolerance due to the CAP theorem [3][4]. However, a solution that uses CRDTs for state convergence could allow the temporary violation of a global invariant, converge the state after the partition,

and then execute any needed compensations [27]. Strong consistency requires more network hops to ensure agreement. The application using strong consistency will not be able to read the data before the last operation.

If we want to spare our system of such expensive operations like a consensus, but still get proper synchronization, we could consider using strong eventual consistency models like CRDTs. The main advantage of the eventual consistency model is that the system will be available to perform write operations even when the network connectivity between replicas is down [6][7]. By avoiding the round-trip time of strong consistency, this model supports far more write operations per second than the other models. This property of eventual consistency can be very useful, if we know that in large scale edge computing systems will have constant data flow, and will do processing on the very edge of the network [23].

On the other hand, a problem that eventual consistency must address is conflicts. Conflict free replicated data types (CRDTs) can resolve conflicts if data could have some mathematical properties like 1) commutative property, 2) associative property and 3) idempotence. This means that CRDTs can be updated without expensive synchronization or consensus, and they are guaranteed to converge eventually if all concurrent updates are commutative. Problems that CRDTs bring is a predefined structure that cannot be overridden. So it can't be used for all problems, especially if operations are not commutative.

Proposed model [23] separate areas in clusters, regions, and (micro)datacenters, inevitably we will have some form of membership protocol running. By using gossip like protocol like SWIM [25] for example, we can piggyback data on every direct or indirect ping. This property could provide great benefits. If we piggyback the machine state as CRDT data and disseminate it over the cluster of nodes, every update eventually would reach every replica, either directly or indirectly. With this approach, we could save a lot of network hops and perform far more write operations even with resource-limited devices that are geo-distributed over some physical area. The second benefit would be asynchronous updates of the replicas, only when we receive a direct or indirect ping from the membership protocol.

V. Conclusion

The massive shift from centralized computing architectures, urges us that we should find a new way to address problems with applications like the internet of things where we can collect and preprocess data at the very end of the network.

Due to the nature of edge computing systems that run on resource-limited devices, replication in the large scale distributed architectures cannot rely that easily on expensive synchronization or consensus between replicas. Consensus algorithms rely on a strong consistency approach. This approach gives us a guarantee by serializing updates to the replicas in global total order but, it is usually a performance and scalability bottleneck. If we loosen the strong consistency requirements, then strong eventual consistency satisfies all properties in the CAP theorem.

If we want to spare our system of such expensive operations like a real-time consensus, but still keep proper synchronization, we could consider strong eventual consistency models like CRDTs. Because of all desirable properties, strong eventual consistency and CRDTs give a lot of possibilities in edge computing, especially in large scale systems. But we must keep in mind, despite all these desirable properties, CRDTs have some disadvantages and limitations that we must be aware of when designing systems and applications that will run on those systems to fully use its potential.

Since the system proposed in this paper is purely on a conceptual level, future work should include the design and development of such a system where we leverage the ability of membership protocol to piggyback the machine state as CRDT data and disseminate it over the cluster of nodes.

References

1. Simić, M., Stojkov, M., Sladić, G., Milosavljević, B. Edge computing system for large-scale distributed sensing systems. In: Konjović, Z., Zdravković, M., Trajanović, M. (Eds.) ICIST 2018 Proceedings Vol.1, pp.36-39, 2018
2. Lampson, B., and Lomet, D., "A New Presumed Commit Optimization for Two Phase Commit," Technical report, Digital Equipment Corporation, February 1993.
3. Eric A. Brewer. Towards robust distributed systems. (Invited Talk) Principles of Distributed Computing, Portland, Oregon, July 2000.
4. Seth Gilbert and Nancy Lynch, "Brewer's conjecture and the feasibility of consistent, available, partition-tolerant web services", ACM SIGACT News, Volume 33 Issue 2 (2002), pg. 51–59. doi:10.1145/564585.564601.
5. Marc Shapiro, Nuno Preguiça, Carlos Baquero, Marek Zawirski. Conflict-free Replicated Data Types. [Research Report] RR-7687, 2011, pp.18. ffinria-00609399v1f
6. Verifying strong eventual consistency in distributed systems, Victor B. F. Gomes, Martin Kleppmann, Dominic P. Mulligan, Alastair R. Beresford, Proceedings of the ACM on Programming LanguagesOctober 2017 Article No.: 109 https://doi.org/10.1145/3133933
7. CAP Twelve Years Later: How the "Rules" Have Changed Eric Brewer's 2012 article on CRDTs (conflict free replicated data types).
8. Yasushi Saito and Marc Shapiro. Optimistic replication. ACM Computing Surveys, 37(1):42–81, March 2005
9. Werner Vogels. Eventually consistent. ACM Queue, 6(6):14–19, October 2008.
10. Birth Of The NearCloud: Serverless + CRDTs @ Edge Is The New Next Thing Kuhiro 10X Faster Than Amazon Lambda, http://highscalability.com/blog/2017/11/6/birth-of-the-nearcloud-serverless-crdts-edge-is-the-new-next.html, accessed May 2020.
11. Serverless plus CRDT is the future of Edge, InfoQ sessions. https://www.infoq.cn/article/serverless-crdts-edge-is-the-new-next , accessed May 2020.
12. Practical Data Synchronization Using CRDTs, InfoQ conference, infoq.com/presentations/data-synchronization-crdt/, accessed May 2020.
13. Peter Bourgon on CRDTs and State at the Edge, InfoQ talks, https://www.infoq.com/podcasts/crdt-state-edge-compute/?utm_so

urce=twitter&utm_medium=link&utm_campaign=calendar, accessed May 2020.

14. Bartolomeu, Carlos and Cláudio Coutinho Bartolomeu. “Large-Scale Geo-Replicated Conflict-free Replicated Data Types.” (2015).
15. Dynamic adaptation of geo-replicated CRDTs, SAC '16: Proceedings of the 31st Annual ACM Symposium on Applied ComputingApril 2016 Pages 514–521.
16. Martin Kleppmann and Alastair R Beresford: “A Conflict-Free Replicated JSON Datatype,” arXiv:1608.03960, August 2016.
17. Shapiro, Marc; Preguiça, Nuno; Baquero, Carlos; Zawirski, Marek (2011), Conflict-Free Replicated Data Types (PDF), Lecture Notes in Computer Science, 6976, Grenoble, France: Springer Berlin Heidelberg, pp. 386–400, doi:10.1007/978-3-642-24550-3_29, ISBN 978-3-642-24549-7.
18. Letia, Mihai; Preguiça, Nuno; Shapiro, Marc (2009). "CRDTs: Consistency without Concurrency Control". Computing Research Repository (CoRR). abs/0907.0929. arXiv:0907.0929. Bibcode:2009arXiv0907.0929L.
19. Nuno Preguiça, Joan Manuel Marquès, Marc Shapiro, and Mihai Leția. A commutative replicated data type for cooperative editing. In Int. Conf. on Distributed Comp. Sys. (ICDCS), pages 395–403, Montréal, Canada, June 2009.
20. Marc Shapiro, Nuno Preguiça, Carlos Baquero, Marek Zawirski. A comprehensive study of Convergent and Commutative Replicated Data Types. [Research Report] RR-7506, Inria – Centre Paris Rocquencourt; INRIA. 2011, pp.50. ffinria-00555588f
21. Schneider, Fred (1990). "Implementing Fault-Tolerant Services Using the State Machine Approach: A Tutorial" (PS). ACM Computing Surveys. 22 (4): 299–319.
22. M. Shapiro, N. Preguiça, C. Baquero, and M. Zawirski. Conflict-free replicated data types. In Proc. of the 13th International Conference on Stabilization, Safety, and Security of Distributed Systems, SSS'11, pages 386--400, 2011. Springer-Verlag.
23. Simić, M., Stojkov, M., Sladić, G., Milosavljević, B. Edge computing system for large-scale distributed sensing systems. In: Konjović, Z., Zdravković, M., Trajanović, M. (Eds.) ICIST 2018 Proceedings Vol.1, pp.36-39, 2018.
24. Simić, M., Stojkov, M., Sladić, G., Milosavljević, B., Zarić, M. On container usability in large-scale edge distributed systems. In: Konjović, Z., Zdravković, M., Trajanović, M. (Eds.) ICIST 2019 Proceedings Vol.1, pp.97-101, 2019.
25. SWIM: Scalable Weakly-consistent Infection-style Process Group Membership Protocol, An DasIndranil GuptaAshish Motivala, Dependable Systems and Networks, 2002. DSN 2002. Proceedings. International, 2002.
26. Leslie Lamport. Time, clocks, and the ordering of events in a distributed system. Communications of the ACM, 21(7):558–565, July 1978.
27. E. Brewer, "CAP twelve years later: How the "rules" have changed," in Computer, vol. 45, no. 2, pp. 23-29, Feb. 2012, doi: 10.1109/MC.2012.37.