Using Conflict-Free Replicated Data Types for Serverless Mobile Social Applications

ABSTRACT

A simple reason for backend systems in mobile applications is the centralized management of state. Mobile clients synchronize local states with the backend in order to maintain an up-to-date view of the application state. As not all mobile social applications require strong consistency guarantees, we survey an alternative approach using special data structures for the mobile applications. These data structures only provide eventual consistency, but allow for conflict-free replication between peers. Our analysis collects the requirements of social mobile applications for being suitable for this approach. Based on exemplary mobile social applications, we also point out the benefits of serverless architecture or architectures with a thin backend layer.

Categories and Subject Descriptors

H.3.5 [**Information Storage and Retrieval**]: [Online Information Services]; C.2.4 [**Computer-Communication Networks**]: [Distributed Systems]

Keywords

Commutative Replicated Data Types, Mobile Systems; Social Applications; Replication

1. INTRODUCTION

The initial success of social applications in the early years of the 21st century has been primarily fuelled by then novel web technologies. The web had already become dynamic due to server-side programming, but emerging technologies such as AJAX suddenly enabled more interactive applications and finally a user experience similar to traditional desktop applications. As a result, new types of web applications appeared that were mainly focused on content generated by its users. The usage of these social web applications is inherently connected to the concept of *being online*: Every interaction between users directly takes place on the website. As the social web applications stores and manages all the data generated by their users, also asynchronous interaction is possible, mediated by the application. With the advent of prevalent mobile computing, social applications have been incorporating new concepts such as location-based services and adapting to content generated and consumed by mobile users. Two new paradigms emerged that focused on the increasing dominance of mobile users: *Mobile first* postulated the idea that new applications should be designed first of all for mobile usage, and web-based access often degraded to an afterthought. In order to temporarily cope with weak network coverage and availability problems, *offline first* became another design principle. Instead of taking network connectivity for granted, applications are now designed to handle a disconnected state as default.

All of these social applications still share a common perception: Whether the user interacts in a Desktop webbrowser, in a mobile browser or with a native mobile app—the actual application eventually resides on a dedicated backend architecture in most cases. This has a number of benefits, including service-aware content dissemination, consistent management of application state and the advantages of a centralized service for their providers. On the other hand, a centralized backend also several drawbacks. It represents a single point of failure, hinders resilience and robustness, and constitutes a threat of directed editorial control. Furthermore, a centralized application architecture cannot directly leverage the locality of its users.

While the idea of distributed, peer-to-peer-based mobile applications is far from new, our main contribution is the analysis of a certain set of data structures that inherently manage central issues thereof: conflict resolution and consistency during replication. After a short illustration of these data structure in Section 2, we want to focus specifically on their specific potential and applicability for mobile social applications. In Section 3 we point out the characteristics necessary for such applications in order to be compatible with the constraints of the data structures. In the subsequent sections we discuss CRDT-enabled applications without any servers (Section 4) or with very thin backends (Section 5). Finally, we touch on our future work and conclude our contribution in Section 6.

2. BACKGROUND

Conflict-Free Replicated Data Types (CRDT) are distributable data types that avoid conflicts during replication processes by adhering to a set of mathematical properties, namely commutativity, associativity and idempotence.

2.1 Conflict-Free Replicated Data Types

The literature distinguishes between two basic approaches of CRDTs: state-based and operation-based. This results in two different types of CRDTs [20] that we briefly explain.

2.1.1 CvRDTs

The first type form State-based Convergent Replicated Data Types (CvRDT). A semilattice is used as payload for the data type, therefore all operations of the type must be associative, commutative and idempotent. Additionally, instances of the data type are restricted to grow monotonically. For each CvRDT a merge function is defined that can merge two or more instances of the data type. Updates are usually first performed locally (*source*) and sent to a random other replica later, where it will be merged with the state of that replica using the merge function (see figure 1). If this step is repeated indefinitely often, every replica will evenutally converge towards the same state [21, 22].

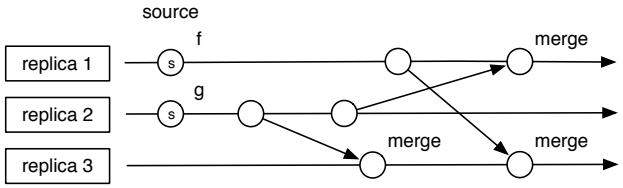

Figure 1: State-based replication (based on [21]).

2.1.2 CmRDTs

The second type are Operation-based Commutative Replicated Data Types (CmRDT). They require all their operations to be either commutative or to have their causal order captured inside the data type. Updates are first performed locally (*source*) and then sent to *all* other replicas (*downstream*, see figure 2). Therefore, a causally ordered, reliable broadcast (or multicast) must be implemented. Concurrent updates are only allowed for commutative operations [22].

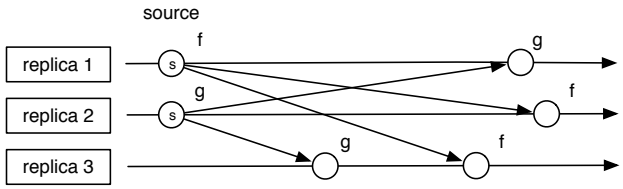

Figure 2: Operation-based replication (based on [21]).

2.1.3 Comparison

CvRDTs are usually easier to implement, because all information is carried by the state and they do not require additional mechanisms for replication. Furthermore, they have less requirements regarding the underlying communication channel. Using pairwise communication, an unknown, potentially indefinite number of replicas can be managed. Additionally, multiple updates can be combined to a single replication step. On the downside, because the whole state is being sent at each update, replication may be very inefficient regarding bandwidth. Building complex data types that satisfy the requirements of CvRDTs can further be costly. CvRDTs were initially designed for objects that only have assignments as operations (register-like objects). Currently they are mainly used in file systems (e.g. NFS, AFS and Coda) and key-value stores (e.g. Dynamo [6] and Riak [12]), as stated by Shapiro et al. [21, 22].

For CmRDTs, the operations must be transmitted alongside their causal order for replication. They require a causally ordered, reliable broadcast channel, which is not easily guaranteed. CmRDTs allow complex operations and therefore have higher expressive strength. They only transmit data about operations and their order, being more efficient regarding bandwidth and allowing larger states. Main use cases are databases (e.g. Swarm [9]) and cooperative systems (e.g. Bayou [16] and IceCube [18]) [21, 22].

Note, that both types are equivalent in the sense that they can be emulated by each other, as demonstrated by Shapiro et al. in their report [21].

2.2 Implementations of CRDTs

The properties of commutativity, associativity and idempotence make the design of CRDTs a non-trivial undertaking. Hence, the number of data structures that have been formalized and implemented is still very limited.

There exist a number of theoretical CRDT specifications for different data types. For some of them, there are also implementations, such as components of open source projects. Table 1 provides an overview of basic CRDTs and known implementations. Apart from basic primitives, there are also more specific CRDTs for certain use cases, that have been published. A list of more specific CRDTs is shown in Table 2.

3. CRDTS FOR MOBILE SOCIAL APPLICATIONS

Next, we want to apply the idea of CRDTs to the context of mobile social applications. This requires an analysis of the constraints of CRDTs and corresponding requirements of mobile social applications

3.1 Using CvRDTs for Mobile Applications

CmRDTs require a reliable broadcast, which is almost impossible to implement for distributed mobile applications with loose coupling. CvRDTs are less demanding towards the transmission channel, as the communication between replicas happens pairwise. This enables the support of peer-to-peer and ad-hoc networks. Therefore, we choose to use CvRDTs over CmRDTs in mobile applications. The following criteria should be considered when deciding whether a mobile (social) application is suitable for the usage of CvRDTs:

- Mutability of data
- Suitable CRDT representations of the used data types
- Size and growth of data
- Eventual Consistency as sufficient consistency model
- Time that nodes can be disconnected from the network
- Number of nodes that share the same data

Data Type	CRDT Specifications	Known Implementations
Integer Vectors	Increment-only Integer Vector [22]	
Counters	G-Counter, PN-Counter [21]	Riak 2.0 [1], Eventuate [19]
Registers	LWW-Register [11], MV-Register [21]	Riak 2.0 [1], Eventuate [19]
Sets	LWW-Element-Set, PN-Set, OR-Set [21], U-Set [21, 24]	Riak 2.0 [1], Eventuate [19]
Maps	Dictionary [24], Map [21]	Riak 2.0 [1]
Graphs	2P2P-Graph, Add-Remove Partial Order [21]	

Table 1: A list of basic CRDTs and known implementations.

Use Case	Underlying CRDTs Structure	Known Implementations
Dynamic Vector Clocks	Increment-only integer vector [22]	
Collaborative writing	Add-Remove Partial Order [21]	Logoot [23], Treedoc [17], WOOT [15], LSEQ [14]
Logs	U-Set [22, 24]	
Time-series Event Storage	LWW-Element-Set [21]	Roshi [4]
Shopping Carts	OR-Set [21]	
Location Data	Combination of 2P-Set, LWW-element-Set & OR-Set [10]	NavCloud (TomTom) [10]
Betting Data	OR-Set [13]	bet365 [13, 3]
Gameplay statistics	Sets of Counters [7]	League of Legends [2, 7, 3]

Table 2: Subset of use-case specific CRDTs and known implementations thereof.

3.1.1 Mutability of Data

The replicated data should be mutable, i.e. the application does not replicate *read-only* or *write-once* data. In those cases, the conflict-freedom of CRDTs does not provide any benefit, while using CRDTs over conventional append-only replication introduces an overhead.

3.1.2 Suitable CRDT Representations

The data the application is replicating must be mapped to a suitable CRDT (or more specifically: CvRDT). To check if this is the case, the data types can be compared to existing CvRDT representations (e.g. Counter or Set, see Section 2). If no suitable CvRDTs are defined, one can try to define their own CvRDT as described in Section 2. When defining a CvRDT, it is important to fulfill the following mathematical requirements:

- *Commutativity*: All operations of the data type are commutative, i.e. their order does not change the result.
- *Associativity*: All operations of the data type can be grouped arbitrarily without changing the result.
- *Idempotence*: All operations are repeatable without changing the result, therefore every single operation instance has to be uniquely identified (e.g. using a vector clock).

To meet these requirements, one will often have to restructure the inner representation of the data type and hide it behind its *actual value*.

3.1.3 Size and Growth of Data

The payload that is transmitted over a mobile network should be as small as possible. Using CvRDTs, the payload represents always the whole state of the data type available on that peer. Therefore, it is not recommended to use CvRDTs when large or indefinitely growing data is replicated in an application. However, application-specific mechanisms can be used to locally purge or restrict the amount of state maintained by a peer, when the loss of older information can be tolerated by the application.

3.1.4 Eventual Consistency

The consistency model of *eventual consistency* must be sufficient for the application, since CRDTs are based on this model. This means that the semantics of the application should be able to tolerate outdated data, i.e. incomplete data should not be regarded as *wrong* data. Rather, performing additional replications should either increase the quality or the up-to-dateness in terms of the application domain and context.

3.1.5 Lower and Upper Bounds of Connectivity

Associated to eventual consistency, the time that nodes can be disconnected from the network without violating the semantics of the application should be as flexible as possible. This means that both immediate replication as well as long network partitionings should be tolerated, depending on the actual application requirements.

3.1.6 Number of Mobile Nodes

The number of nodes (i.e. mobile devices) in the application that share the same data should not be too high, when a low upper bound for connectivity is desired. Otherwise the time necessary for updates to disseminate to all devices may be very long and a skew in local consistency become more apparent. However, when eventual consistency is fully supported, high inconsistencies between single nodes are not problematic which enables the support of a large number of nodes. Note that a number of nodes can represent their own replication domain, while others still use the same application, but with independent application state.

4. BACKENDLESS MOBILE APPLICATIONS

Backendless mobile applications are applications that operate without any backend. We limit our considerations to applications that share a (at least partially) globally shared

state and therefore require communication in order to replicate data between peers.

4.1 Replication and Consistency in Backendless Mobile Applications

For backendless mobile applications eventual consistency is a mandatory requirement. It is possible that within subgroups of devices data is consistent while the groups have high inconsistencies between each other if the number of devices per group is rather low and communications happens frequently, while communication between devices of the different groups happens infrequently. This may be due to geographic separation and the limitation to local replication. Note that a single device from one group communicating with one of the devices from another group is sufficient to cause (eventual) consistency between the groups. A single device will never be able to assess the *global state* or the quota of updates it has received in relation to the total number of updates in the application. Therefore no statements can be made about the *actual* consistency.

4.2 Ad-hoc Communication in Backendless Mobile Applications

Since generally no information about other devices are available, ad-hoc communication is used for replication. Ad-hoc protocols of mobile devices, such as Wi-Fi Peer-to-Peer [8], Bluetooth Low Energy (BLE)[1] or NFC[2], can be used to transmit data. A mechanism to discover available peers using the same application has to be implemented. For instance, the Wi-Fi Peer-to-Peer framework offers such a discovery inherently [8].

4.3 User Management in Backendless Mobile Applications

Devices have to be distinguished in order to identify changes of different peers (and therefore support vector clocks). A naive approach would be to use device identifiers like the *NSUUID Identifier For Vendor* for iOS[3] devices or the *Android-ID*[4] for Android devices. Problems arise because these identifiers are not guaranteed to be unique since they can be changed by the device user and the same user using multiple devices cannot be identified as well.

Assigning unique identifiers in mobile networks is a common problem. Usual solutions use a hash of the device's IP address as public key [5]. However, this solution can only guarantee the key to be unique within a single session. Furthermore, without a trusted and centralized instance to manage user accounts this solution still does not provide a way to authenticate the user (e.g. when mapping the key to a username). A more advanced solution is the generation of public-private key pairs on each device and the exchange of public keys on first contact. This may require authentication using near field technologies like NFC or the usage of QR codes[5]. If the user is allowed to use multiple devices, the private key has to be duplicated in a similar way. Every device then has its own list of trusted users it can communicate with. The keys could also be used for other security mechanisms. However, unique usernames are impossible to enforce, as the system design does not provide a global view on the common shared state.

4.4 Example: An Anonymous Sharing App

As example application for a backendless mobile application, we choose a geosocial, anonymous sharing application (e.g. Yik Yak[6] or Jodel[7]). This application allows users to anonymously post short, location-based messages that nearby user can comment on and vote on. This type of application is convenient since data is mutable (votes, set of posts/comments) and only basic data types are used (a counter for votes and a set each for posts and comments), which there already exist CRDT specifications for (see table 1). Furthermore, posts, comments and votes are relatively small in terms of data size and meta-data. Eventual Consistency is fully supported as the semantics of the application includes only local updates per definition. A single device can also be disconnected from the network for an arbitrary amount of time without violating this semantics. By purging older entries and only maintaining the latest N posts, the growth of application state can be restricted. Such an mechanism fosters inconsistencies of old data, but this is acceptable due to the application's ephemeral semantics.

5. MOBILE APPLICATIONS WITH THIN BACKENDS

As an alternative to the strict notion of backendless applications with CRDTs, we suggest another architecture that applies CRDTs for data replication between peers, but still employs a centralized backend for certain tasks.

This in-between approach seems unprofitable at first, as it introduces drawbacks of both concepts. However, a strict separation of concerns makes this approach very efficient: While the dissemination of application state between peers is governed by CRDTs using direct peer-to-peer replication, the backend provides orthogonal features that are hard to distribute. This includes application features such as account creation, user management, discovery/bootstrapping, or the management of meta-data on data consistency (note that the actual application state is still not part of the backend). Furthermore, a thin backend can take on the task of trusted third party that provides a key infrastructure for all peers.

5.1 Replication and Consistency in Mobile Applications with Thin Backends

When the mobile application leverages a thin backend, data about the number of devices sharing (and therefore replicating) the same data and the respective number of updates they have received and/or made can be stored globally. This way, devices could check their quota of updates in relation to the total number and initiate steps to counteract high inconsistencies if it is under a certain threshold. It is also possible to have the backend initiate this check and coordinate replication accordingly. Additionally, devices could sign in at the service on startup of the application to make their public IPs available through the network. This enables

[1] http://www.bluetooth.com/Pages/low-energy-tech-info.aspx
[2] http://nfc-forum.org/what-is-nfc/
[3] https://developer.apple.com/library/ios/documentation/UIKit/Reference/UIDevice_Class/
[4] http://developer.android.com/reference/android/provider/Settings.Secure.html#ANDROID_ID
[5] http://www.qrcode.com/en/
[6] https://www.yikyak.com/
[7] https://jodel-app.com/

global replication, since devices can obtain the IP of every single device in the network. It is also possible to identify groups of devices and only choose one representative instead of storing all IPs. Using these mechanisms, a higher level of (still eventual) consistency can hence be achieved.

5.2 Peer-to-Peer Communication in Mobile Applications with Thin Backends

For communication, the same mechanisms as for Backendless Mobile Applications can be used (see section 4.2). Additionally, communication via the Internet can be used when exchanging IPs as described in the previous section.

5.3 User Management in Mobile Applications with Thin Backends

The backend can be used to store user accounts and handle authentication. This way, the identity of a communication partner can be verified by the backend before replication. Additionally, the same account can be used on multiple devices and usernames can be guaranteed to be unique.

5.4 Example: A Campus Voting App

As example application for a mobile application using a thin backend we choose a voting application, where users can vote on different topics but it is important to verify that the user is authorized to vote on a topic and to ensure that every authorized user only has a restricted number of votes. The actual votes should be anonymous, i.e. not be visible to other users. This type of application is convenient because data (topics and their respective votes) are mutable. The data types can easily be represented as CRDT (set for the different options and a counter for the votes each). Replicated data size is relatively small as only votes have to be replicated once the topics are set up. Also, data is limited in size by the number of topics and users, as every user can only vote once. Eventual Consistency is sufficient as consistency model since every vote makes the result more accurate, missing votes are considered as temporary result (as if the user has not yet voted). Furthermore, devices can be disconnected from the network for an arbitrary amount of time without violating the semantics of the application. The number of nodes that share the same data is limited to the number of users that are authorized to vote on a topic. To ensure users only vote once, for each topic the backend creates tokens with a private key and transmits one token to each device of a user that is authorized to vote for that topic. User rights are stored at the backend. Tokens can then be verified on each device (at each replication step) by using the respective public key, initially provided by the backend.

6. FUTURE WORK & CONCLUSION

So far, we have implemented several prototypical Android-based applications. They apply CRDTs for maintaining application state and rely on the Wi-Fi peer-to-peer API of Androird 4.0. In the next step, we plan to synthesize our experiences and extract a generic framework for CRDT-based application state handling. By modularizing and packaging the replication logic, we hope to make the concept of CRDTs more accessible for Android developers in general. The framework is currently designed to provide a set of basic types and a modular component for peer discovery and data communication (e.g. Wi-Fi peer-to-peer).

Apart from the framework, we also want to address more general problems of CRDTs when used in mobile systems—security and privacy. As part of a distributed backend or database application, CRDTs are mainly used in closed system with a limited scope. For mobile applications, the context is very different, as replication may happen in a non-trusted environment or with peers with with potentially malicious intents.

Due to the peer-to-peer nature of such systems, privacy requires special attention. A replication might unintentionally leak data or the timing and order of replications may yield information specific to the usage context. CRDTs generally provide a way of data distribution or modification that is exploitable by malicious peers. Attack vectors include, but are not limited to, denial of service attacks, spam flooding, or the injection of counterfeit operations. Therefore, trust management represents another important issue for the usage of CRDTs in an open environment without any backend system. We believe it is important to survey existing cryptographic mechanisms and privacy-preserving technologies and assess their applicability either on top of the existing data structures, or as an integrated part thereof.

The properties of CRDTs yield a number of distinct restrictions for mobile applications, such as relaxed consistency, assessable data growth, and limited operations. As long as application logics can cope with these restrictions, we believe that the usage of CRDTs is an interesting approach for mobile applications that refrain from a centralized backend and embrace an offline-first design. Due to the fact that a couple of social mobile applications do not require strict consistency guarantees, we argue that CRDTs are especially well suited for them. CRDTs then allow a mobile application architecture that requires either a thin backend or that operate even without any backend. Thanks to the properties of CRDTs, the mobile application developers do not have to consider state conflicts or resolutions and can immediately focus on the actual application logic of their system.

7. REFERENCES

[1] Basho Technologies, Inc. Riak Documentation - Riak 2.0. `http://docs.basho.com/riak/latest/intro-v20/`. Accessed: 2015-06-16.

[2] Basho Technologies, Inc. Riak for Gaming. `http://basho.com/posts/technical/riak-for-gaming/`. Accessed: 2016-03-21.

[3] Basho Technologies, Inc. Solution Brief: Riak KV Solution for Gaming. 2015.

[4] P. Bourgon. Roshi. `https://github.com/soundcloud/roshi`. Accessed: 2014-06-01.

[5] K. R. B. Butler, S. Ryu, P. Traynor, and P. D. McDaniel. Leveraging identity-based cryptography for node id assignment in structured p2p systems. *IEEE Transactions on Parallel and Distributed Systems*, 20(12):1803–1815, Dec 2009.

[6] G. DeCandia, D. Hastorun, M. Jampani, G. Kakulapati, A. Lakshman, A. Pilchin, S. Sivasubramanian, P. Vosshall, and W. Vogels. Dynamo: Amazon's highly available key-value store. In *ACM SIGOPS Operating Systems Review*, volume 41, pages 205–220. ACM, 2007.

[7] G. Eardley. Tracking Millions of Ganks in Near Real Time. Strange Loop, 2013.

[8] Google Inc. Wi-fi peer-to-peer. `https://developer.android.com/guide/topics/connectivity/wifip2p.html`. Accessed: 2015-06-08.

[9] V. Grishchenko. Swarm. `https://github.com/gritzko/swarm`. Accessed: 2016-03-11.

[10] D. Ivanov. Practical demystification of crdts. Amsterdam.Scala meetup, August 27, 2015.

[11] P. R. Johnson and R. H. Thomas. The maintenance of duplicate databases. internet request for comments rfc 677. *Information Sciences Institute*, 1976.

[12] R. Klophaus. Riak core: Building distributed applications without shared state. In *ACM SIGPLAN Commercial Users of Functional Programming*, page 14. ACM, 2010.

[13] D. Macklin. Why bet365 chose riak. 2015.

[14] B. Nédelec, P. Molli, A. Mostefaoui, and E. Desmontils. Lseq: An adaptive structure for sequences in distributed collaborative editing. In *Proceedings of the 2013 ACM symposium on Document engineering*, pages 37–46. ACM, 2013.

[15] G. Oster, P. Urso, P. Molli, and A. Imine. Data consistency for p2p collaborative editing. In *Proceedings of the 2006 20th anniversary conference on Computer supported cooperative work*, pages 259–268. ACM, 2006.

[16] K. Petersen, M. J. Spreitzer, D. B. Terry, M. M. Theimer, and A. J. Demers. *Flexible Update Propagation for Weakly Consistent Replication*, volume 31. ACM, 1997.

[17] N. Preguica, J. M. Marques, M. Shapiro, and M. Letia. A commutative replicated data type for cooperative editing. In *Distributed Computing Systems, 2009. ICDCS'09. 29th IEEE International Conference on*, pages 395–403. IEEE, 2009.

[18] N. Preguiça, M. Shapiro, and C. Matheson. Semantics-based reconciliation for collaborative and mobile environments. In *On The Move to Meaningful Internet Systems 2003: CoopIS, DOA, and ODBASE*, pages 38–55. Springer, 2003.

[19] Red Bull Media House. Eventuate. `http://rbmhtechnology.github.io/eventuate/`. Accessed: 2016-03-20.

[20] Y. Saito and M. Shapiro. Optimistic replication. *ACM Computing Surveys (CSUR)*, 37(1):42–81, 2005.

[21] M. Shapiro, N. Preguiça, C. Baquero, and M. Zawirski. A comprehensive study of convergent and commutative replicated data types. Technical Report 1, Institut national de recherche en informatique et en automatique, Jan 2011.

[22] M. Shapiro, N. Preguiça, C. Baquero, and M. Zawirski. Conflict-free replicated data types. Technical Report 2, Institut national de recherche en informatique et en automatique, Aug 2011.

[23] S. Weiss, P. Urso, and P. Molli. Logoot-undo: Distributed collaborative editing system on p2p networks. *Parallel and Distributed Systems, IEEE Transactions on*, 21(8):1162–1174, 2010.

[24] G. T. Wuu and A. J. Bernstein. Efficient solutions to the replicated log and dictionary problems. In *Proceedings of the third annual ACM symposium on Principles of distributed computing*, pages 233–242. ACM, 1984.