UiT The Arctic University of Norway

Faculty of Science and Technology
Department of Computer Science

Conflict-Free Data Types for Local-first Software

Stian Økland

Capstone thesis in Computer Science December 2020

Abstract

This research project is about the learning and determining if concepts Local-first, Automerge, Hypermerge with Hypercore and Hyperswarm are suitable for a distributed system, and contains theory and experiments. Local-first is an "old fashion", and later become a modern idea of storing files locally. In the current state of technology, using the cloud to store and do functional operation on user-data is popular. In some cases where an internet connection is not necessary for the system to function, but the need to share data or collaborate with other user is a must, a Local-first approach is well suited. Automerge is a library built for collaboration and use in distributed systems. Hypermerge consists of Hyperswarm and Hypercore and is using functionalities from Automerge. Hypercore is an append-only log to keep track of changes and modifications. Hyperswarm is a combination of Kademlia-based distributed hash table (DHT) and multicast domain name system (MDNS) and is used to detect and connect to other peers in a local network. Documents to be collaborated on in Automerge are CRDTs. CRDT, conflict-free replicated data types, are used in applications with collaboration between users. This data abstractions guaranties convergences for replication. CRDT provides both availability and partition-tolerance but does not guarantee consistency of the CAP theorem.

Contents

List of Figures

1

Introduction

This research project aims to research Local-first, Automerge and CRDT. Experiments on update time and concurrent remote instances are completed to do benchmarks. The goal is to learn if concepts, in a combination of each other, is suitable for a collaboration oriented distributed system.

Cloud systems can be used for almost anything, but in some cases internet connection can unstable or even non-existent. Being dependent on a connection to the cloud is not always optional. Local-first [1] is a concept where an "old fashion", later a modern, approach is used. This approach is about storing files and documents locally and all reads and writes to these files are done locally. This meaning the user is in no need of an internet connection to modify them. In a distributed system or application where the users are required to collaborate with each other, libraries like Automerge and Hypermerge is used. Together they provide functionality like document convergence (CRDT), detecting and connecting tho other peers in a local network, and a log to keep track of modifications and changes done locally to documents shared between multiple peers.

Automerge [14] is a data structure library, specifically, built for use in real-time collaborative applications and are versatile in the context of communication and framework. Developers can, for example, use the likes of Bluetooth for connection and React [11] or Redux [12] as a framework to develop the system or application in. A document, where users store data, used in Automerge is a Conflict-free Replicated Data Type (CRDT)[14]. CRDT are concurrently

modification supported data structures and guarantees convergence for modifications done in parallel. This makes users using or collaborate on the same document are seeing the same changes. In all distributed system, the CAP theorem is a factor. Consistence - multiple users have the same copy of a document, availability - data availability, and partition-tolerance - the system works even when partitioned on the network. CRDT provides both availability and partition-tolerance, but consistency is weakened since CRDTs consistency are strong eventual consistency [3].

Hypermerge [18] is a library combined of Automerge, Hypercore [23] and Hyperswarm [20], and CRDT. Hypercore is an append-only log, providing an overview of modifications and changes done locally to documents. Hyperswarm is a combination of Kademlia[21]-based distributed hash table (DHT)[24] and multicast domain name system (MDNS)[22]. DHT to provide and store information about the peers in the local network and MDNS to find peers and fetch IP addresses to use in the connection process.

Combining Automerge and Local-first works well, as demonstrated in pixelpusher [9] and Capstone [10]. Experiments & Results (chapter 3) show a downside of using Hyperswarm on multiple concurrent instances on a single device.

1.0.1 Outline

Structure of the remainder of this research project:

Chapter 2 - Theoretical background Presents theoretical background about Local-first, CAP-theorem, CRDT, and Automerge in general. Followed by Hypercore, Hyperswarm, and Hypermerge. React Native is also presented.

Chapter 3 - Experiments & Results Two experiments are presented and evaluated, one concerning update time with multiple remote instances of Hypermerge, and the other concerning update time dependent on number of messages to update.

Chapter 4 - Conclusion Concludes the research project.

2

Theoretical background

2.1 Local First

Local-first is about ownership. Ownership of documents, notes, drawings, code, and so on. In the modern time cloud are more and more used and cloud apps, in general, does not necessarily provide a feeling of ownership since the data is stored somewhere else and not on your local computer. The cloud lets the user access its data whenever it wants and provide consistency when sharing with other users [1].

The concept of Local-first is old and comes from "old-fashioned" apps, also called programs. Later "modern" apps in case of collaboration concerning merging data. These programs run locally on the user's computer. This program writes and reads files on the local disk and all the data is the user needs are stored in these files. Because of this the user has full control over the data and has the freedom to do what it wants with it, like backups, manipulation the files or long-term archiving [1].

It is possible to combine the functionality of both Local-first and cloud-based software. By taking the aspect of ownership from Local-first and by using the cloud as a back-up. In the traditional cloud-based application the primary instance of a document is the one on the server and any other copy of the document, the user's devices, is secondary. Applications based on Local-first principles use the cloud as a way to store documents as back-up or collaboration platform and this instance is secondary, and the users local stored instance is

considered as the primary [1].

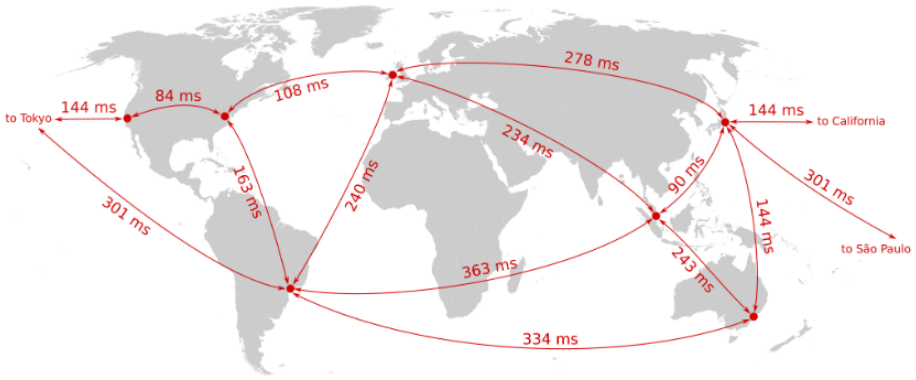

Figure 2.1: Server-to-server times between AWS datacenters around the world. Data from Basilis et al.[2] and figure form Kleppmann et al.[1]

2.1.1 Pros and cons

There is both pros and cons with a Local-first approach to applications, here are some examples.

Pros:

- For users dependent on applications to work without internet connections, or are in areas with bad connectivity.
- Write and read requests are done directly on the document the user is modifying. By having the primary copy locally, latency around the globe has little to nothing impact.

Cons:

- If the user is dependent on live collaboration with other users Local-first is a disadvantage, since the users have to be connected. Especially if multiple users are modifying the same data.
- In Local-first, latency is not only the milliseconds delay displayed in Figure 2.1. Since offline is not a wrong state, the latency can be minutes, days or even months.

2.2 CAP theorem

A sort description of consistency, availability and partition-tolerance in a distributed system.

Consistency

Where consistency is guaranteed there is a total order of operations. Although these operations are done in a distributed fashion, it appears like it is been done on a single device. In a system with shared memory, it is required that all operations are handled one by one, and not in parallel to avoid conflicts [5]. In a Local-first approached distributed system, consistency is weakened. As stated consistency is all about copies being updated and documents having the same state. For example, when the time is taken between every time connection to a network happens. If an application is based on finding peers on a local network, users have to meet to get the update in a Local-first versus an application based around the cloud.

Availability

To be available, a distributed system all nodes in the system has to send a response when receiving a request. In this case, the system has a method to detect if a node is unavailable [5].

Partition-tolerance

Partitioning is when messages are lost when sent from one device to another. Partition-tolerance is dependent on both atomicity, every response sent to be atomic, and availability, the node receiving a request will response. This implies, even if nodes in the network are failing, a valid response are generated [5].

2.2.1 Strong eventual consistency

Strong eventual consistency is a strict form of eventual consistency. Eventual consistency means that all replicas will reach the same state, eventually, when the user of the document stops submitting modifications. Although updates are eventually converged, some systems execute updates immediately, which will in some cases result in conflicts and the system has to use resources to roll back to solve the conflict. Strong eventual consistency solves this problem by

making sure all replicas that have the delivers the same updates have the same state [3].

2.3 Conflict-free Replicated Data Types

When using devices with cloud technology, such as smartphones, the user expects applications on the device to work at all time, both online and offline. The cloud makes data shareable globally and is easy to access whenever the user is. Each device working on a document has its replica of the original document and may do changes to it locally. Eventually, there will be more than one copy of a document different from the rest of them. This is when conflict-free data types are used [4]. In a Local-first distributed system consistency is considered as eventual consistency, in contrast to a cloud-dependent system that can support strong consistency.

CRDTs are data structures that support modification concurrently, and updates happening at the same time are guaranteed to be converged. With CRDT, the convergent happen when all replicas have applied the same updates [4].

CRDT follows three principles [6].

- A updated documents and its replicas should all automatically converge and have the same sate. This is also known as strong eventual consistency.
- All user updates should be done due to concurrent update.
- All updates done concurrent should be seen as the same as if the updates were done sequential.

An example of a CRDT is a Grow Only set (G-set) [7]. This implies that an element can only be added to the set. These elements describe changes done to a document. A case where changes are added to a document, a new element is added, but if the same changes are deleted, the element in the set is not removed. It adds a new element describing which changes got deleted from the document. When a merge happens between two G-sets a union operation is performed to get all changes.

2.4 Automerge

Automerge is a library for applications built for collaboration in JavaScript. This library is a pure data structure library, and have a very similar way of dealing with states of an application as normal JavaScript, but supports syncing and merging automatically. In a Local-first application, a copy is stated locally and each device with a copy can do changes offline and save it to a local disk. Merging are done when a network connection is available, and Automerge is handling this by figuring out which changes are done and needs to be synced between the connected devices [13]. Used in real-life applications like Trellis, a Trello clone [8], pixelpusher, pixel art to CSS application [9], and Capstone, a real-time idea-sharing platform [10].

Since Automerge is a data structure library, the user or developer has the freedom to choose their preferred way of communication. Client/server, Bluetooth, peer-to-peer and many more. Automerge is also compatible with different platforms, such as Chrome, Firefox and Node.js, and each Automerge object is an immutable snapshot of the application which makes the application accessible for changes and compatible with functional reactive style of programming as React [11] and Redux [12].

An Automerge instance is the current state of an application and is immutable, hence, updates are never done in place. Updates are completed by taking the current state and the new state and merge them. Changes can either be local or remote. Where local changes are triggered by the user changing some of the application data on the users' device. Remote changes are when another user on a different device are doing changes their copy and sent via the network.

Automerge, for the most part, can be seen as two parts: frontend and backend. With two parts, the application can be run on two threads. One handling the user application, while thread number two handles the backend. The backend thread can perform computational tasks without the frontend being affected. Frontend and backend use asynchronous message-passing as communication, which are JavaScript objects and arrays, which results in possible different languages for frontend and backend.

JavaScript is a programming language with First-class function, which are functions that are treated as a variable and can be passed as an argument to other functions. The language is lightweight and interpreted, and are most known as a scripting language for web pages [15]. Also used in for example Node.js[16], Adobe Acrobat[17], and many more non-browser environments.

A local user's modifications are done to the frontend and sent to the backend for processing. When processing is done, frontend receives confirmation. A remote

user's modifications are received on the backend for processing before frontend receives the modifications. This saves both time and resources, by not involving the frontend before the new state is processed and updated [14].

In Automerge each change is a JSON object. JSON being a JavaScript Object Notation format, are both easy to read and write for humans, and uses universal structures like name/value pairs and ordered list, which makes it compatible with almost all existing programming languages [19]. This JSON object contains five properties: actor - a UUID for the actor, seq - sequence number for the change, deps - changes dependencies, message - human-readable message that describes the change, and ops - array of operations. Each operation operates on an object, and in Automerge there are four types of objects. A map and a list are both represented as a JavaScript object, but text and table are represented as an Automerge instance (Automerge.Text/Table). Since tables behave like maps, and text behaves like list, Automerge maps and list is considered as object types [14].

2.5 Hypercore

Hypercore [23] is append-only log designed to use in a distributed system, and with a secure transport protocol, scalability and performance are well in reach. Hypercores list function is used to log changes done to a document. An append-only log can be seen as a list the user only can append to, this implies no changes can be done to already existing element in the list. By only appending each user can choose to only exchange bits of the log they may be interested in, and since the user only uses this bits the distribution process is efficient.

2.5.1 Hyperswarm

Hyperswarm [20] is part of Hypercore and are a combination of Kademlia-based distributed hash table (DHT)[24] and multicast domain name system (MDNS). Hypercore is used in a distributed system to detect other peers in a local network. Each Hypercore has a hash containing the Hypercores public key, in Hyperswarm used as a discovery key. This discovery key is used as a topic to find and share IPs and ports between other peers. A domain name system is a system who converts hostname into a computer-friendly IP address. Each device has an IP address and this essential when looking up other devices to connect to [22].

The way Kademlia protocol works, each node in the tree knows about at least

one node in the other subtrees. Each node in a Kademlia protocol has a 160-bit ID, and each node is treated as leaves in a binary tree. The trees structure described in Kademlia: A Peer-to-peer Information System Based on the XOR Metric [21].

> For any given node, we divide the binary tree into a series of successively lower subtrees that don't containing the node. The highest subtree consists of the half of the binary tree not containing the node. The next subtree consists of the half of the remaining tree not containing the node, and so forth.

As seen in figure 2.2, node 0011 finds node 1110 by querying the best node to know the targeted node. For each search, the node 0011 learns and querying closer and closer to the targeted node. The numbered lines in the figure define which order the querying is happening.

Kademlia is an UDP-based DHT. As stated in A Fast UDP [25],

> UDP simply adds transport-level addressing and an optional checksum to the Internet Protocol (IP) service of best-effort datagram delivery.

In comparison to other protocols, UDP is fast, which makes it preferable in cases of real-time functional applications. Two common known weakness of UDP, package loss and packages out of order does not get any attention to get fixed. UDP does have a mechanism to provide detection of corrupt data, checksum. Where the two last bits of the UDP header and comparing these two bits before sending and after receiving a segment. If the checksums are different the segment is corrupted [26].

2.6 Hypermerge

Hypermerge is a combination of Automerge, CRDT, and Hypercore, and are a library in Node.js for building p2p applications. These applications are collaborative and do not need server infrastructure to function as a distributed system and suites applications using functional and reactive languages.

Each user of an application using Hypermerge has its repository with data. Each repository has a list of Hypercores, an append-only log, and when a connection is made between two peers they exchange lists of Hypercores. These lists contain changes done and each user has to replicate all missing changes to continue to maintain the state of the application [18].

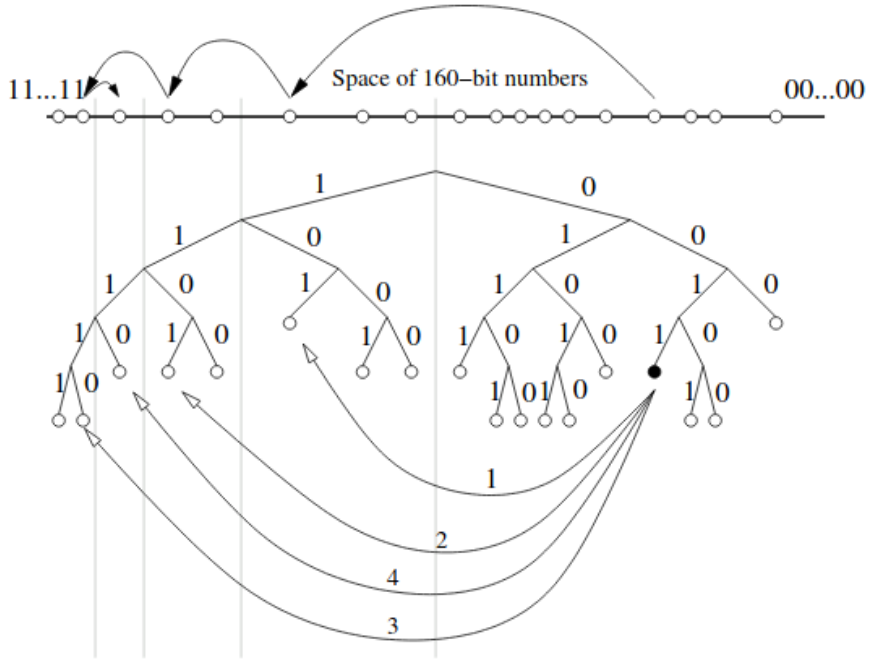

Figure 2.2: Node 0011 lookup schema for finding node 1110 [21]

.

2.7 React Native

React Native is a framework for developing cross-platform applications for mobile, both iOS and Android. React Native uses JavaScript as a language and CSS. By using both JavaScript and Cascading Style Sheets (CSS), a description of a JavaScript document like layouts, fonts and colours, mobile applications are not so unlike web applications [27].

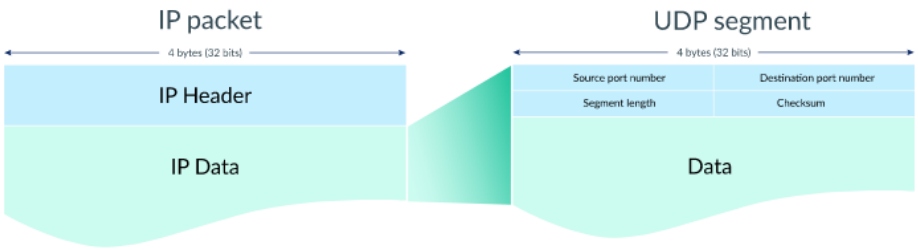

Figure 2.3: An UDP overview [26]

3

Experiments & Results

To study the performance of Hypermerge, tests were implemented to measure how the system handles differences in both the number of concurrent repositories and number of messages. Each instance of Hypermerge gets a repository in the startup. The tests are done locally on an HP EliteDesk, configured with an Intel duo quad-core i7-4770 CPU, 15.6 GiB System memory, running Ubuntu 18.04.5 LTS.

Experiment 1 results in figure 3.1 were done 10 times with cases of 2 to 5 repositories where all repositories listen to the same document. Repository number 1 does insert 1000 messages in form of numbers from 0 to 999. Since the remaining repository listens, time taken for the other repositories to get the changes are logged and displayed. Since this is done locally on one device, the maximum of concurrent repositories to guarantee results every run are 5.

Figure 3.2 is an overview of CPU activity during a successful test run of experiment 1. The blue spike at the start are resources used to do the first changes to a document. The flat region is the time frame the other repositories use to detect the changes. Figure 3.3 illustrate what happens when too many repositories fight over the same resources, 7 in this case. The startup of the test is the same as the last one, and two repositories get updated, but one resource get hung up and a deadlock may happen.

Experiment 2 results in figure 3.4 are the time taken for one repository to

update its document with varying number of messages. Results from 100, 1000 and 10 000 messages are inclining for each point. In a real-life scenario, an update with 10 000 messages rarely happens or never at all.

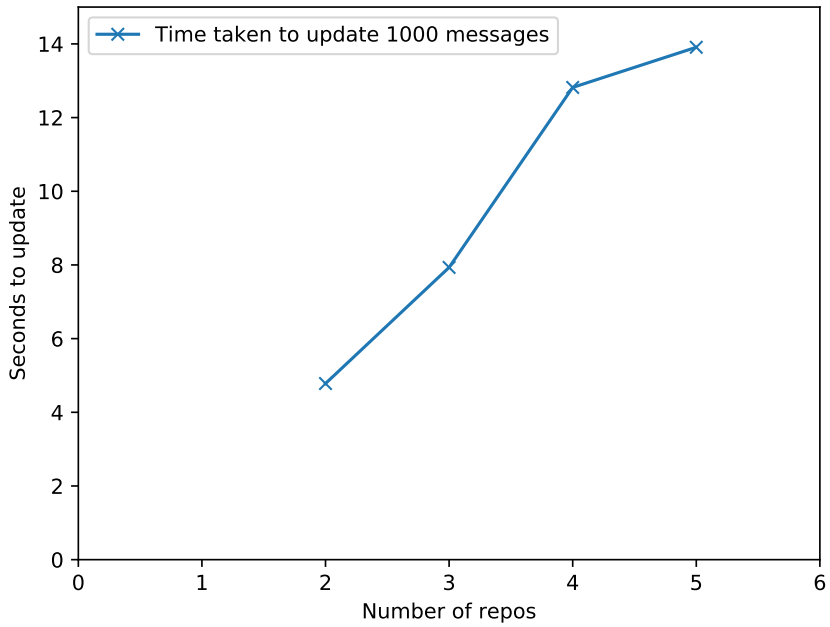

Figure 3.1: Update time of Experiment 1. 1 to 4 remote repositories, where the first repository is the one doing local changes.

Figure 3.2: CPU history of experiment 1, 5 repositories.

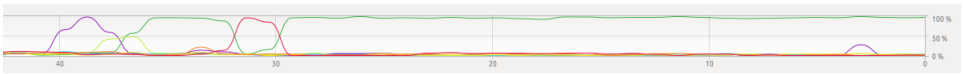

Figure 3.3: CPU history of Experiment 1, 7 repositories

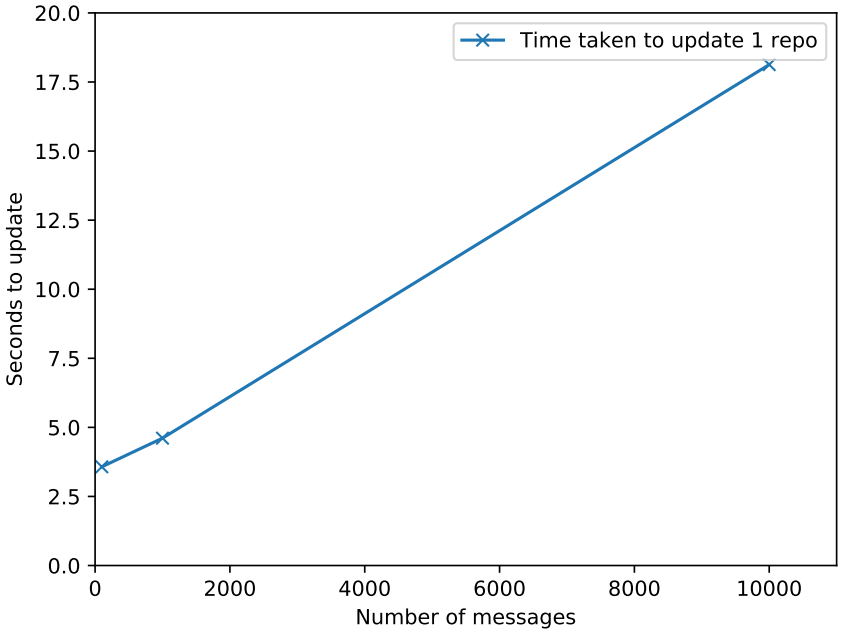

Figure 3.4: Update time in Experiment 2

4

Conclusion

This research project aimed to explore the possibilities the use of Automerge, integrated into Hypermerge, in a Local-first system. The goal was to determine if these libraries are suitable for a potential Master project, both the architecture and the functionality provided. By researching technologies and methods related to Automerge and Local-first, and simple testing of the practical aspect of Hypermerge. Analyses of test results indicate that the scaling ability and cases of multiple instances of Hypermerge may not be suitable for a Local-first application when multiple instances are deployed at a single site. On the other hand, research and prototypes done by Automerge and Hypermerge creators prove usage of these libraries suites real-time collaboration, without sharing sensible data, very well.

Bibliography

[1] Kleppmann, M., Wiggins, A., van Hardenberg, P., and McGranaghan, M. Local-first Software: You Own Your Data, in spite of the Cloud. In *Proceedings of the 2019 ACM SIGPLAN In-ternational Symposium on New Ideas, New Paradigms, and Reflec-tions on Programming and Software (Onward! '19), 2019.*

[2] Basilis, P., Davidson, A., Fekete, A., Ghodsi, A., Hellerstein, J., and Stoica, I. Highly Available Transactions: Virtues and Limitations (Extended Version). In *40th International Conference onVery Large Data Bases, September 1st - 5th 2014, Hangzhou, China.Proceedings of the VLDB Endowment,Vol. 7, No. 3, 2014*

[3] Shapiro, M., Preguiça, N., Baquero, C., and Zawirski, M. Conflict-Free Replicated Data Types. In *Défago X., Petit F., Villain V. (eds) Stabilization, Safety, and Security of Distributed Systems, (SSS 2011). 2011.*

[4] Yu, W., and Ignat, C.-L. Conflict-free replicated relations for multi-synchronous database management at edge. In *IEEE International Conference on Smart Data Services (SMDS). 2020*

[5] Gilbert, S., and Lynch, N. Brewer's conjecture and the feasibility of consistent, available, partition-tolerant web services. In *SIGACT News,vol. 33, no. 2, pp. 51–59, 2002. 2002*

[6] Kleppmann, M., and Beresford., A. A Conflict-Free Replicated JSON Datatype. In *IEEE Transactions on Parallel and Distributed Systems. 2016*

[7] Ahmed-Nacer, M., Martin, S., and Urso, P. File system on CRDT. In *RESEARCH REPORT N° 8027. 2012*

[8] Automerge Trellis. *https://github.com/automerge/trellis. 2020*

[9] Automerge pixelpusher.
https://github.com/automerge/pixelpusher. 2020

[10] Ink & Switch Capstone.
https://github.com/inkandswitch/capstone. 2020

[11] Facebook Open Source React.
https://reactjs.org/. 2020

[12] Abramov, D., et al. Redux.
https://redux.js.org/. 2020

[13] Kleppmann, M., et al. Automerge.
https://github.com/automerge/automerge. 2020

[14] Kleppmann, M., et al. Automerge internals.
https://github.com/automerge/automerge/blob/main/INTERNALS.md. 2020

[15] Mozilla, et al. JavaScript.
https://developer.mozilla.org/en-US/docs/Web/JavaScript. 2020

[16] OpenJS Foundation Node.js.
https://nodejs.org/en/. 2020

[17] Adobe Adobe.
https://www.adobe.com/devnet/acrobat/javascript.html. 2020

[18] Hypermerge ARCHITECTURE
https://github.com/automerge/hypermerge/blob/master/ARCHITECTURE.md. 2020

[19] JSON Introduction JSON.
https://www.json.org/json-en.html, 2020

[20] Hypercore Protocol Hyperswarm is a DHT for the home.
https://hypercore-protocol.org/#hyperswarm, 2020

[21] Maymounkov, P., and Mazieres, D. Kademlia: A Peer-to-peer Information System Based on the XOR Metric. In *Peer-to-Peer Systems. 2002*

[22] Cloudflare, Inc What is DNS?
https://www.cloudflare.com/learning/dns/what-is-dns/. 2020

[23] Hypercore Protocol Hypercore is a distributed append-only log *https://hypercore-protocol.org/#hyperswarm, 2020*

[24] Galuba, W., and Girdzijauskas, S. Distributed Hash Table. *https://link.springer.com/referenceworkentry/10.1007%2F978-0-387-39940-9_1232. 2009*

[25] Partridge, C., and Pink., S. A Faster UDP. In *IEE/ACM TRANSACTIONS ON NETWORKING, VOL. I. 1993*

[26] Khan Academy User Datagram Protocol (UDP). *https://www.khanacademy.org/computing/computers-and-internet/xcae6f4a7ff015e7d:the-internet/xcae6f4a7ff015e7d:transporting-packets/a/user-datagram-protocol-udp. 2020*

[27] Zhou, X., Hu, W., and Liu, G. React-Native Based Mobile App for Online Experimentation. In *IEE 39th Chinese Control Conference (CCC), 2020*