



BEUTH HOCHSCHULE FÜR TECHNIK BERLIN
University of Applied Sciences

Masterarbeit

Medieninformatik

Fachbereich VI – Informatik und Medien

Untersuchung der Konfliktmanagementstrategien verschiedener offlinefähiger Systeme

Berlin, den 7. April 2018

Autorin:

Jacoba BRANDNER

Matrikelnummer:

833753

Betreuer:

Herr Prof. Dr. Hartmut SCHIRMACHER

Gutachterin:

Frau Prof. Dr. Petra SAUER

Abstract

In dieser Arbeit wird ...

Abstract

This work includes ...

INHALT

1 Einführung	5
1.1 Motivation	5
1.2 Zielstellung	5
2 Bestehende offlinefähige Systeme / Konzepte	6
2.1 Kollaborative Software	6
2.1.1 Google Docs	6
2.1.2 Google Wave	6
2.1.3 Dropbox / 'Clouds'	6
2.1.4 Kollaborative Editoren	6
2.2 Offline-First Frameworks/Bibliotheken	7
2.2.1 redux? react-native? webpack?	7
2.2.2 Realm	7
2.2.3 Datomic? (Closure)	8
2.2.4 Redux Offline	8
2.2.5 react-native-offline	10
2.2.6 offline-plugin für webpack	10
2.2.7 hoodie	11
2.2.8 CouchDB	11
2.2.9 PouchDB	11
3 Grundlagen	12
3.1 Konflikte	12
3.1.1 Consistency Availability Partition tolerance (CAP) Theorem?	13
3.2 Offline First	13
3.2.1 Anforderungen an Offline-First / PWA?	14
3.3 Progressive Web Apps?	14
3.3.1 ServiceWorker?	14
3.3.2 localForage AsyncStorage?	14
3.3.3 IndexedDB?	14
3.4 Replikation in verteilten Systemen	14
3.4.1 Last-Write-Wins (LWW) Blockieren?	14
3.4.2 Operational Transformation	15

3.4.3	Conflict-free replicated data type	17
3.5	Das CouchDB Replikationsmodell	18
3.5.1	Konsistenz	19
3.5.2	Replikation?	19
3.5.3	Konfliktmanagement	19
4	Vorgehen	20
5	Fazit	21
	Abkürzungen	22
	Glossar	23
	Abbildungsverzeichnis	24
	Literaturverzeichnis	24
	Anhang	27

1 EINFÜHRUNG

We live in a disconnected & battery powered world, but our technology and best practices are a leftover from the always connected & steadily powered past. [off]

Heutzutage besitzen mehr als fünf Milliarden Menschen ein Mobiltelefon und drei Milliarden haben Zugang zum Internet [Ban16].

langsame Verbindungen, Unterbrechungen. Auch bei 3G und 4G ist die Latenz schrecklich (Lie-Fi?) -> Offline-First Apps können eine bessere User experience bieten.

1.1 MOTIVATION

Ich möchte eine offlinefähige (mobile?) Anwendung entwickeln und stelle mir folgende Fragen.

Welche Software/ Framework benutze ich dazu?

Auf was muss ich bei der Auswahl achten?

Was erwarte ich von einer offline fähigen App?

(funktioniert und kein Datenverlust) -> Synchronisation und Konfliktmanagement

1.2 ZIELSTELLUNG

Wichtig: Offline nutzbar ohne Datenverlust.

Untersuchung des Verhaltens bei Konflikten (verursacht durch paralleles Arbeiten ohne Internetverbindung).

Wie leicht/schwer ist es zu implementieren?

2 BESTEHENDE OFFLINEFÄHIGE SYSTEME / KONZEPTE

2.1 KOLLABORATIVE SOFTWARE

rauslassen? Kann ich benutzen um kollaborativ zu arbeiten -> **was passiert wenn ich offline bin?**

2.1.1 GOOGLE DOCS

(benutzt OT)

2.1.2 GOOGLE WAVE

(benutzt OT)

2.1.3 DROPBOX / 'CLOUDS'

2.1.4 KOLLABORATIVE EDITOREN

Wiki: https://en.wikipedia.org/wiki/Collaborative_real-time_editor

<https://atom.io/packages/covalent>

<https://atom.io/packages/firepad>

Markdown: <https://hackmd.io/>

LaTeX: <https://www.sharelatex.com/>

Online editor: <http://etherpad.org/> (OT)

Mockingbird (tool for creating wireframes): <https://gomockingbird.com/home> (OT)

marvelapp? <https://marvelapp.com/collaboration/>

-> Zahl steigend, (nachdem Google die Drive Realtime API veröffentlicht hat, die auf Operational Transformation (OT) basiert und es *third-party Apps* ermöglicht, dieselbe Zusammenarbeit wie Google Docs zu verwenden)

plus wachsende Anzahl von offen zur Verfügung gestellter Bibliotheken und Frameworks die es ermöglichen, offlinefähige Anwendungen zu programmieren. Siehe Kapitel 2.2

2.2 OFFLINE-FIRST FRAMEWORKS/BIBLIOTHEKEN

Ich möchte aber auch eigenständig Software entwickeln die man vielleicht nicht nur zum Arbeiten nehmen kann, sondern auch um Quatsch zu machen wie Katzengifs zu teilen.

2.2.1 REDUX? REACT-NATIVE? WEBPACK?

Progressive Web App (PWA)

2.2.2 REALM

Backend für mobile Anwendungen (Java, Swift, C#, JS). Realm Datenbank oder Realm Platform(= DB+ Object Server). Schreiben groß 'Offline First Experience' überall hin (webseite, whitepaper...)

- lokale DB, plattformübergreifend
- Object Server fungiert als Middleware-Komponente in der mobilen Applikation (App)-Architektur und managed die Datensynchronisation, eventhandling und Integration in Legacy-Systeme. Kann Daten effizient und simultan synchronisieren und **löst in Echtzeit, automatisch Konflikte**
- **Key-features:** Datensynchronisation in Echtzeit, Skalierbarkeit, Cross-Platform Datenmodell, Eventhandling, regelmäßige Backups, Datenintegrations API, Datensicherheit
- **key mobile use cases:** Reactive app architectures, Offline-first experiences, mobilizing legacy apis, realtime collaboration

[rea17b]

- Realtime Data Sync (sendet automatisch Änderungen in Echtzeit)
- Daten-sync-Protokoll komprimiert die marginalen Änderungen (statt das ganze Objekt) in Binärformat und übergibt sie zwischen Gerät und Server.
- synchronisiert die spezifischen Operationen zusammen mit den Daten

- Diese zusätzlichen Informationen erfassen genau das, was man beabsichtigt hat, **sodass das System Konflikte automatisch auflösen kann**. Dies führt zu einer vorhersagbaren Synchronisation ohne manuellen Eingriff, der die Leistung beeinträchtigt.
- (Objektorientierte) Datenbank auf dem Gerät
- Echtzeit Synchronisation
- Konfliktlösung benutzt OT (und vorgegebene Regeln): Man kann custom Konfliktlösungs-Regeln erstellen
- Unterstützung von Transaktionen? Ist das nicht normal? – Konfliktlösung passiert auf **Transaktionsebene**
- Wenn Änderungen aufgrund einer unterbrochenen Netzwerkverbindung offline gehen oder das Gerät leer ist, gehen keine Daten verloren.

[rea17a]

2.2.3 DATOMIC? (CLOSURE)

2.2.4 REDUX OFFLINE

“Persistenter Redux store für *reasonaboutable*™ Offline-First Anwendungen“.

Ist ein eigenständiger Statuscontainer und kann mit jeder Webanwendung angewandt werden, die sich *deklarativ auf Basis einer einzigen Datenquelle rendern lässt*, wie beispielsweise React¹, Vue², oder Angular³ [reda].

ist eine experimentelle Bibliothek die auf *battle-tested* patterns der Offline-First Architektur aufbaut

REDUX OFFLINE verspricht nicht, die Webanwendung offlinefähig zu machen. Um *assets*(Bilder, Skripte etc) zwischenzuspeichern, muss zusätzlich noch ein ServiceWorker implementiert sein.

Benutzt redux-persist und redux-optimist.

Bei jeder Änderung wird der Redux store auf dem Datenträger gespeichert, und bei jedem Start automatisch neu geladen. (Standardmäßig IndexedDB, localForage, AsyncStorage)

Eine mit REDUX OFFLINE erstellte Anwendung funktioniert ohne weitere Anpassung offline im Lesemodus. Also wenn die benutzende Person vom (Redux-)Status lesen möchte. Um auch im Schreibmodus offline zu funktionieren, werden alle Netzwerkgebundenen Aktionen in einem Store-internem Queue gespeichert. Dann erstellt REDUX OFFLINE

¹JavaScript Bibliothek: <https://reactjs.org/>

²JavaScript Framework: <https://vuejs.org/>

³JavaScript Framework: <https://www.angular.io>

einen Unterbaum `offline`, wo unter anderem der interne Status und ein Array namens `outbox` verwaltet wird. Um diese Aktivitäten bei Internetverbindung ausführen zu können, müssen alle notwendigen Daten Plus Metadaten gespeichert werden. Die Metadaten sind für die Information zuständig, was davor oder danach passieren soll. Es gibt drei Metadaten die REDUX OFFLINE interpretieren kann:

`meta.offline.effect` - Die Daten die gesendet werden sollen?

`meta.offline.commit` - Aktion die ausgeführt wird sobald Daten erfolgreich gesendet wurden

`meta.offline.rollback` - Aktion die bei permanent fehlgeschlagener Internetverbindung [redb]

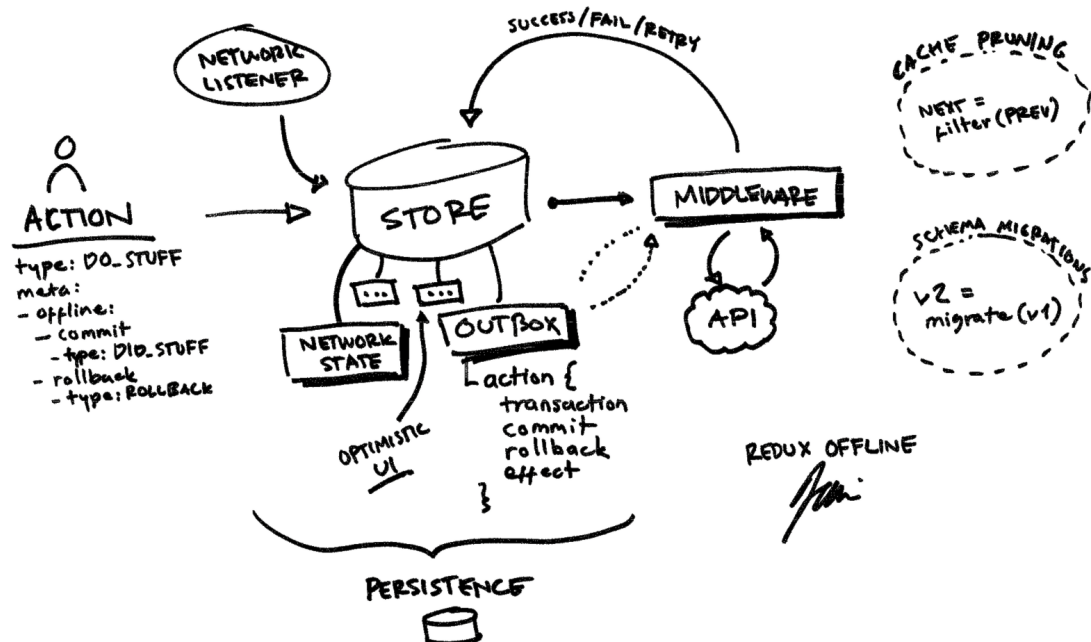


Abbildung 2.1: Redux Offline Architektur Quelle: [Evä17]

Die grundlegende Idee hinter Redux Offline ist, dass der REDUX STORE die Datenbank ersetzt/ist. Jede Aktion die benötigt wird um offline zu arbeiten, wird im STORE persiiert und durch die `meta.offline`-Daten weiß die Anwendung was online zu tun ist. Wie die Grafik 2.1 (links) zeigt, wird jede (offline-unterstützende) Aktion mit dem `offline.meta` Feld dekoriert. Darin wird beschrieben, wie der *Netzwerkeffekt* ausgeführt werden soll (`effect`) und welche Aktion ausgelöst werden soll, wenn sie erfolgreich (`commit`) ist oder fehlschlägt (`rollback`). Diese Offline-Aktionen werden im STORE-internen Queue gespeichert und werden, einmal online, an den Server gesendet.

User Interface (UI)

Es umfasst netzwerkfähige Application Programming Interface (API)-Aufrufe, das Persistieren des Zustands(Status), das Stapeln von Nachrichten und die Behandlung von Fehlern, Neuversuchen, optimistische Bedienoberfläche-Aktualisierungen, Migrationen, Cache-Bereinigung und mehr. Weil das kompliziert sein kann, zielt REDUX OFFLINE darauf ab, eine vernünftige Reihe von Standardverhaltensweisen zu bieten, die verwendet werden, und nach Bedarf einzeln überschrieben werden können.

Grundsätzlich ist das Problem, das REDUX OFFLINE löst, kein technisches, sondern ein Architekturproblem. Architekturen können im Code implementiert werden, aber um verstanden, angewendet und nachvollziehbar zu werden, müssen sie gut kommuniziert werden. -> Dokumentation

Konflikte?

REDUX-PERSIST

localStorage. github [redc] medium [San16]

REDUX-OPTIMIST

2.2.5 REACT-NATIVE-OFFLINE

React Native = JavaScript React Framework um native, mobile Apps zu bauen. blabla

Behandelt online/offline Verbindung. Kann die Internetverbindung auch regelmäßig prüfen

Speichert nur den Status online/offline im store. -> erlaubt so unterschiedliches Rendern von Komponenten.

```
const YourComponent = ( isConnected ) => (  
<Text>isConnected ? 'I am connected to the internet!' : 'Offline :(</Text>  
)
```

Zusammen mit Redux hat es einen 'Mehrwert': Hat dann einen 'Offline-Queue' um Aktionen zu wiederholen (im Intervall) `meta.retry?` Array of actions which, once dispatched, will trigger a dismissal from the queue oder nicht `meta.dismiss?` []. [Acu]

2.2.6 OFFLINE-PLUGIN FÜR WEBPACK

webpack ist ein JavaScript 'Bundler', packt JavaScript-Dateien und oder assets für die Verwendung in Browsern. blabla

Bietet offline experience für webpack Projekte. Benutzt SERVICEWORKER und APPCACHE unter der Haube -> cached nur (gebündelten) von webpack generierten assets. Für die anderen Dateien (z.B. index.html die nicht gebündelt wird oder Modul von CDN) braucht mal ein html-plugin oder benutzt die externals :

```
const OfflinePlugin = require('offline-plugin')
const offline = new OfflinePlugin
```

```
const offline = new OfflinePlugin(
  externals: ['index.html'],
)
```

Es gibt diverse config-options für SERVICEWORKER und AppCache... [Sto]

2.2.7 HOODIE

Benutzt CouchDB und PouchDB... [hoo]

2.2.8 COUCHDB

Apache Couch™ ist ein Datenbank Management System (DBMS) (kann mehrere Datenbanks (DBs) enthalten) Vektoruhr ⁴

content addressable versions: Idee: Nimm den Objekthalt (content) und jag ihn durch eine Hashfunktion

2.2.9 POUCHDB

PouchDB ist eine Open-Source-JavaScript-Datenbank, die so konzipiert wurde, dass sie im Browser läuft. PouchDB ermöglicht es Anwendungen zu erstellen, die sowohl offline als auch online funktionieren. Daten können lokal gespeichert werden, sodass alle Funktionen der Anwendung auch im Offline-Modus zur Verfügung stehen. Daten werden unabhängig von der nächsten Anmeldung (des nächsten Onlinezugangs) zwischen **Clients**, CouchDB oder kompatiblen Servern synchronisiert. PouchDB läuft auch in Node.js⁵ und kann als direkte Schnittstelle zu CouchDB-kompatiblen Servern verwendet werden [pou].
8iiiiiiiiiiiiiiiiiiii9kiiiiiiiiiiiiio

⁴https://en.wikipedia.org/wiki/Vector_clock

⁵JavaScript Laufzeitumgebung, steht unter <https://nodejs.org/en/download/> zum Download bereit

3 GRUNDLAGEN

Was bedeutet offlinefähig?

Software, mit der ein freigegebenes Dokument zusammen mit anderen über das Internet bearbeitet werden kann, kann überaus wertvoll sein wenn man beispielsweise in einem Team arbeitet. Heutzutage gibt es viele webbasierte Software für simultanes kollaboratives Editieren (*Textdokumente, Tabellenkalkulationen, Präsentationen, Quellcode*). Im Kapitel 2 werden Produkte und Frameworks vorgestellt, mit deren Hilfe man solche Software erstellen kann.

Lassen Sie uns einen Moment nehmen, um genauer zu definieren, was wir unter dem Begriff kollaboratives Echtzeit-Editieren verstehen. Wir möchten, dass mehrere Personen, die an verschiedenen Computern arbeiten, jederzeit Änderungen an einem auf einem Server gehosteten Dokument vornehmen können. Diese Änderungen werden sofort mit den anderen KollegInnen synchronisiert. Kein Client sollte vor einer Änderung mit dem Server oder einem anderen Client kommunizieren müssen. Insbesondere ist es nicht erforderlich, eine Sperre vom Server zu erhalten um ein Dokument zu bearbeiten und gleichzeitige Editierungen können auftreten. Nachdem alle Änderungen synchronisiert wurden, sollte jeder Client das exakt gleiche Dokument sehen.

3.1 KONFLIKTE

Verteilte Systeme: Das ist ein mächtiger Begriff für viele Ideen und Konzepten, aber es läuft in der Regel darauf hinaus: Da sind zwei oder mehr Computer, die durch ein Netzwerk verbunden sind und es wird versucht, dass einige der Daten auf beiden Computern gleich aussehen. ==> Ein System das zuverlässig über ein Netzwerk funktioniert.

Zwei Geräte, ein Server, über Netzwerk verbunden.

Spezielle Eigenschaft von Netzwerken: Verbindung kann jederzeit abbrechen: Acht Irrtümer der verteilten Datenverarbeitung:

1. Das Netzwerk ist zuverlässig

Der Strom kann ausfallen oder Glasfaserkabel können kaputt sein — Das Netzwerk ist nicht zuverlässig.

2. Die Latenz ist gleich null

Glasfaserkabel werden durch Mikrowellen (oder andere Technologien) ersetzt um Millisekunden an Zeit zu sparen. Das würde nicht passieren, wäre die Latenz bei null. Es dauert nun mal eine gewisse Zeit(ms) wenn ein Signal eine (geografisch)weite Strecke zurücklegen muss – Die Latenz ist nicht gleich null.

3. Die Bandbreite ist unendlich

Daten können nicht schneller fließen als die Komponenten die sie verarbeiten (Middleware, Datenbank ...) – Die Bandbreite ist nicht unendlich.

4. Das Netzwerk ist sicher

Der HEARTBEAT-BUG⁶, der im Jahr 2014 behoben wurde und die Sicherheitslücke im ICE-Wireless Local Area Network (WLAN) im Jahr 2016⁷ sind nur zwei Beispiele die zeigen, dass das Netzwerk nicht sicher ist.

5. Die Netzwerkstruktur wird sich nicht ändern

Eine Datenbank kann beispielsweise über mehrere Server verteilt sein, die (teilweise) voneinander abhängig sind. Ein Server mit Abhängigkeiten kann ausfallen, es kann eine Aktualisierung für einen anderen Server geben – die Struktur ändert sich.

5. Die Netzwerkstruktur wird sich nicht ändern

Eine Datenbank kann beispielsweise über mehrere Server verteilt sein, die (teilweise) voneinander abhängig sind. Ein Server mit Abhängigkeiten kann ausfallen, es kann eine Aktualisierung für einen anderen Server geben – die Struktur ändert sich.

6. Es gibt eineN AdministratorIn

Es kann beliebig viele AdministratorInnen geben.

7. Die Datentransportkosten sind gleich null

Netflix bezahlte anfang 2014 diversen InternetanbieterInnen dafür, dass Netflix KundInnen bevorzugten Internetzugang haben.

8. Das Netzwerk ist homogen

Es gibt verschiedene Arten von Netzwerk: 3G, 4G, LTE, WiFi. Wird beeinflusst durch Hardware (Smartphone, Tablet, PC, Laptop, Router ...) [fal]

3.1.1 CAP THEOREM?

3.2 OFFLINE FIRST

1. Separate Apps from Data

2. Deliver App Code (and make it cachable) appcache & ServiceWorkers

⁶<http://heartbleed.com/> – Zugriff: 07.04.2018

⁷<https://netzpolitik.org/2016/datenschutz-im-zug-deutsche-bahn-will-sicherheitsluecke-in-neuem-ice-wlan-schliessen/> – Zugriff: 07.04.2018

3. Save Data Offline localStrocare / localForage, IndexedDB, other Wrapper
4. Detect Connectivity navigator.onLine (Lie-fi)
5. Sync Data - Build upon existing solutions – CouchDB/PouchDB | remoteStorage

3.2.1 ANFORDERUNGEN AN OFFLINE-FIRST / PWA?

1. Lese- und Schreibfähigkeit
2. Sync im Hintergrund
3. Cache sollte in der Lage sein App-Aktualisierungen zu “überstehen”

3.3 PROGRESSIVE WEB APPS?

3.3.1 SERVICEWORKER?

3.3.2 LOCALFORAGE || ASYNCSTORAGE?

3.3.3 INDEXEDDB?

3.4 REPLIKATION IN VERTEILTEN SYSTEMEN

Es stellt sich heraus, dass die Implementierung dieser Art von Echtzeit-Zusammenarbeit alles andere als trivial ist. Im Folgenden werden die drei Strategien OT, Conflict-free replicated data type (CRDT) und Last-Write-Wins (LWW) vorgestellt plus CouchDBs Peplikationsmodell.

3.4.1 LAST-WRITE-WINS (LWW) || BLOCKIEREN?

NoSQL bietet dies, indem es einen Zeitstempel von irgendeiner Art beibehält, der ihnen hilft, zu entscheiden, welcher Schreibvorgang zuletzt kam. Datenbanken wie DynamoDB⁸ oder Cassandra⁹ verwenden LWW, um Schreibvorgänge zu verarbeiten. Bei diesen Leuten erfordern stark konsistente Schreibvorgänge das Schreiben in ein Quorum von Shards, wodurch Sie mehr Geld kosten.

⁸https://aws.amazon.com/de/dynamodb/faqs/What_is_a_readwrite_capacity_unit

⁹<https://docs.datastax.com/en/cassandra/3.0/cassandra/dml/dmlConfigConsistency.html>

3.4.2 OPERATIONAL TRANSFORMATION

OT ist eine weit verbreitete Technologie zur Unterstützung von Funktionalitäten in Kollaborativer Software. Sie stammt aus einer im Jahre 1989 veröffentlichten Forschungsarbeit und wurde ursprünglich nur für die gemeinsame Bearbeitung von Klartext-Dokumenten entwickelt [EG89]. Später ermöglichte weitere Forschung OT durch Unterstützung von *Sperrungen, Konfliktlösungen, Benachrichtigungen, Bearbeitung von Baumstrukturierten Dokumenten, ...* zu verbessern und erweitern. Im Jahr 2009: Google Wave, Google Docs

Es wird das Problem untersucht, dass OT in einer idealen Umgebung löst und dadurch zu einem funktionierendem Algorithmus gelangt.

Ziel: Mehrere BenutzerInnen können gleichzeitig an einem Dokument arbeiten, sehen Änderungen der anderen in Echtzeit (live), ohne dass einer Verzögerung durch die Latenz verursacht wird. Gleichzeitig auftretende Mehrfachänderungen sollen nicht zu unterschiedlichen Dokumentenzuständen führen.

FUNKTIONSWEISE

Kollaborative Systeme, die OT verwenden, benutzen normalerweise den replizierten Dokumentenspeicher. Das heißt jeder **Client** verfügt über eine eigene Kopie des Dokuments. Jede Änderung an einem freigegebenen Dokument wird als Operation dargestellt. Operationen sind Repräsentationen von Änderungen an einem Dokument. (Beispielsweise: Füge 'Hello world!' an Position 0 in das Textdokument ein). Eine Operation zeichnet im Wesentlichen den Unterschied zwischen einer und der nachfolgenden Version eines Dokuments auf. Die Anwendung einer Operation auf das aktuelle Dokument führt zu einem neuen Dokumentstatus. Die Operationen erfolgen auf lokalen Kopie und die Änderungen werden an alle anderen **Clients** weitergegeben. Wenn ein **Client** die Änderungen von einem anderen **Client** empfängt, werden die Änderungen normalerweise **vor** ihrer Ausführung transformiert. Die Transformation stellt sicher, dass anwendungsabhängige Konsistenzkriterien (Invarianten) von allen Standorten gepflegt werden.

Es gibt die Operationen **Einfügen**

Das Einfügen besteht aus dem eingefügten Text und dessen Position im Dokument (`insert('h', 0)`). Für die Position kann ein Koordinatensystem ermittelt werden (Zeilennummer: Position in Zeile oder einfacher: Dokument wie eine Folge von Zeichen behandeln, also einfach einen nullbasierten Index vergeben.)

und **Löschen**

Löschen(5,6) = löscht 5 Zeichen, beginnend bei Position 6. Mehr benötigt man nicht, denn update = delete & insert

Um gleichzeitige Operationen zu behandeln, gibt es eine Funktion (normalerweise Transform

genannt), die zwei Operationen übernimmt, die auf denselben Dokumentstatus angewendet wurden (aber auf verschiedenen Clients). Daraus wird eine neue Operation berechnet, die nach der zweiten Operation angewendet werden kann. Diese behält die erste beabsichtigte Änderung der Operation.

Des Weiteren unterstützt OT Operationen wie `update`, `point`, `lock`.

Beispiel : Benutzer A fügt an Position 12 das Zeichen 'A' ein Benutzer B fügt am Anfang des Dokuments ein 'B' ein. Die konkurrierenden Operationen sind daher Einfügen (12, 'A') und Einfügen (0, 'B'). Wenn wir die Operation von B einfach an Client A senden und dort anwenden würden, gäbe es ein Problem. Aber wenn die Operation von A an B gesendet, und angewandt wird nachdem Operation B angewandt wurde ist, würde das Zeichen 'A' eine Position zu weit links von der korrekten Position eingefügt werden. Dokumentstatus A und Dokumentstatus B sind nicht identisch.

Daher muss A's `insert(12, 'A')` gegen die Operation von B transformiert werden. So wird berücksichtigt, dass B ein Zeichen vor der Position 12 eingefügt hat (die die Operation `insert(13, 'A')` erzeugt.)

Diese neue Operation kann auf Dokument B nach B's Operation angewandt werden. Die Grundidee von OT besteht darin, die Parameter einer Editieroperation gemäß den Auswirkungen **zuvor ausgeführter** konkurrierender Operationen anzupassen, so dass die transformierte Operation die korrekte Wirkung erzielen und die Dokumentenkonsistenz aufrechterhalten kann.

STRUKTUR?

Transformations- 1. Kontrolle, 2. Eigenschaften, Bedingungen, 3. Funktionen
Vorteile der Trennung?

KRITIK

False-Tie puzzle? *A Generic Operation Transformation Scheme for Consistency Maintenance in Real-time Cooperative Editing Systems* und *Achieving convergence, causality-preservation, and intention-preservation in real-time cooperative editing systems*

Während der klassische OT-Ansatz, Operationen durch ihre Versätze im Text zu definieren, einfach und natürlich zu sein scheint, werfen real verteilte Systeme ernsthafte Probleme auf. Nämlich, dass sich die Operationen mit endlicher Geschwindigkeit fortpflanzen, die Zustände der TeilnehmerInnen sind oft verschieden, so dass die resultierenden Kombinationen von Zuständen und Operationen extrem schwer vorherzusehen und zu verstehen sind. Wie Li und Li es ausdrückten: "Aufgrund der Notwendigkeit, eine komplizierte Fallabdeckung in Betracht zu ziehen, sind formale Beweise sehr kompliziert und

fehleranfällig, selbst für OT-Algorithmen, die nur zwei charakteristische Primitive behandeln (Einfügen und Löschen)“ [LL10].

Damit OT funktioniert, muss jede einzelne Änderung an den Daten erfasst werden: “Einen Schnappschuss des Zustands zu erhalten, ist normalerweise trivial, aber das Erfassen von Bearbeitungen ist eine ganz andere Sache. [...] Der Reichtum moderner Benutzerschnittstellen kann dies problematisch machen, besonders in einer browserbasierten Umgebung“ [Fra09].

KONSISTENZMODELLE??? CC MODELL, CCI, CSM, CA

3.4.3 CONFLICT-FREE REPLICATED DATA TYPE

Die Idee von CRDTs ist, dass jeder "Typ"(wie ein Einkaufswagen) mit Intelligenz handelt, um Konflikte automatisch zu lösen.

CRDTs sind Objekte, die ohne teure Synchronisation aktualisiert werden können. Sie konvergieren schließlich, wenn alle gleichzeitigen Aktualisierungen kommutativ¹⁰ sind und wenn alle Aktualisierungen schließlich von jeder Replik ausgeführt werden [SPBZ11a]. Um diese Garantien zu geben, müssen diese Objekte bestimmte Kriterien erfüllen, welche im Folgenden beschrieben werden [SPBZ11b].

ZUSTANDBASIERTER ANSATZ

Wenn ein Replikat ein Update von einem Client empfängt, aktualisiert es zuerst seinen lokalen Status und dann, einige Zeit später, seinen **vollständigen Status**. So sendet jedes Replikat gelegentlich seinen vollständigen Status an ein anderes Replikat im System. Um ein Replikat, das den Status eines anderen Replikats empfängt, wendet eine **Zusammenführungsfunktion** (merge) an, um den empfangenen Status mit dem lokalen Status zusammenzuführen. Entsprechend sendet dieses Replikat gelegentlich auch seinen Status an ein anderes Replikat, sodass jedes Update schließlich alle Replikate im System erreicht.

OPERATIONSBASIERTER ANSATZ (OT?)

Bei diesem Ansatz sendet ein Replikat seinen vollständigen Status (kann groß sein) nicht an ein anderes Replikat. Stattdessen sendet es nur den **Aktualisierungsvorgang** an **alle** anderen Replikate im System und erwartet von ihnen, dass sie das Update auf sich anwenden.

¹⁰Unverändert bei Vertauschen der Operanden

Da es sich um einen Sendevorgang handelt, wenn zwei Updates u_1 und u_2 , bei einem Replikat i angewendet werden und diese Updates an zwei Replikate r_1 und r_2 gesendet werden, können diese Updates in unterschiedlicher Reihenfolge bei diesen replikaten ankommen. r_1 kann sie in der Reihenfolge u_1, u_2 empfangen, während bei r_2 die Updates in umgekehrter Reihenfolge (u_2, u_1) ankommen können. Sind die Aktualisierungen **kommutativ**, können die Repliken zusammengeführt werden, egal in welcher Reihenfolge die Updates bei ihnen ankommen - der resultierende Zustand ist derselbe. In diesem Modell wird ein Objekt, für das alle gleichzeitigen Aktualisierungen kommutativ sind, CmRDT (commutative replicated data type - kommutativ replizierter Datentyp) genannt.

Beispiel:...

CRDTs befassen sich mit einem interessanten und grundlegendem Problem in verteilten Systemen, haben jedoch eine wichtige Einschränkung: "Da ein CRDT konstruktionsbedingt keinen Konsens verwendet, hat der Ansatz starke Einschränkungen; Dennoch sind einige interessante und nicht-triviale CRDTs bekannt- [SPBZ11b]. Die Einschränkung ist, dass die CRDT-Adresse nur einen Teil des Problemraums betrifft, da nicht alle möglichen Aktualisierungsoperationen kommutativ sind und daher nicht alle Probleme in CRDTs umgewandelt werden können. Auf der anderen Seite können CRDTs für einige Arten von Anwendungen durchaus nützlich sein, da sie eine nette Abstraktion zur Implementierung replizierter verteilter Systeme bieten und gleichzeitig theoretische Konsistenzgarantien bieten.

3.5 DAS COUCHDB REPLIKATIONSMODELL

Aufgabe der Replikation von CouchDB ist die Synchronisation 2+n Datenbanken. Lösungen: Zuverlässige **Synchronisation** von Datenbanken auf verschiedenen Geräten. **Verteilung** der Daten über ein Cluster von DB-Instanzen die jeweils einen Teil des requests beantworten (Lastverteilung) und **Spiegelung** der Daten über geografisch weit verteilte Standorte.

Durch die inkrementelle (schrittweise) Arbeitsweise kann CouchDB genau dort weitermachen wo es unterbrochen wurde wenn während der Replikation ein Fehler auftritt, beispielsweise durch eine ausfallende Netzwerkverbindung *Es werden auch nur die Daten übertragen, die notwendig sind, um die Datenbanken zu synchronisieren.*

Das Besondere an CouchDB ist, dass es darauf ausgerichtet ist, Fehler/Konflikte vernünftig zu behandeln statt anzunehmen es träten keine auf (vgl. [ALS10] S. 7f). Wie oben beschrieben, gibt es in Verteilten Systemen einige Fehler die auftreten können.

Das CouchDB Replikationsmodell erlaubt eine nahtlose, peer-to-peer (direkte) Datensynchro-

nisation zwischen beliebig vielen Geräten. Das CouchDB Replikationsprotokoll ist in CouchDB selbst implementiert, das die Serverkomponente abdeckt. Dann gibt es das PouchDB-Projekt, das dasselbe Protokoll in JavaScript implementiert, das auf Browser- und Node.js-Anwendungen abzielt. das deckt Ihre Kunden und dev-Server ab. Schließlich gibt es Couchbase Mobile und Cloudant Sync, die auf iOS und Android laufen und das CouchDB Synchronisationsprotokoll in Objective-C bzw. Java implementieren.

3.5.1 KONSISTENZ

LOKALE KONSISTENZ

VERTEILTE KONSISTENZ

3.5.2 REPLIKATION?

3.5.3 KONFLIKTMANAGEMENT

4 VORGEHEN

- was verspricht redux-offline
- Wie funktioniert redux-offline? Welche Strategie zur Konfliktlösung wird verwendet?
- was könnte daran problematisch sein?
- Wie hoch ist der Implementierungsaufwand?
- Dasselbe für Couch & Pouch

Implementierungsaufwand: Anforderungen an Simulator? Konzeption? usw.

5 FAZIT

noch aus Jans Blog:

OT ist so konzipiert, dass beliebig viele Personen gleichzeitig am selben Text arbeiten können und es eine gewisse Netzwerkinstabilität bewältigen kann. Im Allgemeinen müssen die Personen jedoch jederzeit mit dem Internet verbunden sein. Bearbeiten sie den Text ohne Internetverbindung, können Ihre Änderungen später, aber nicht unbegrenzt später, integriert werden. Darüber hinaus ist es für Text und nicht für generische Objekte konzipiert. Für echte Offline-Funktionen generischer Datenobjekte sind Operational Transforms daher weniger nützlich.

*CRDTs sind spezialisierte Datenstrukturen, die für die Verwendung in verteilten Systemen entwickelt wurden. Sie verfügen über viele Eigenschaften, ..., haben aber kein Konfliktkonzept. Nun, sie sind spezialisierte Datenstrukturen wie *sets* und *counters* und keine generischen Objektrepräsentationen wie JavaScript Object Notation (JSON), also muss man sich in diese spezialisierten Datenstrukturen einarbeiten, und vielleicht gibt Schwierigkeiten, die verwendeten Anwendungsobjekte darauf abzubilden.*

ABKÜRZUNGEN

API	Application Programming Interface
App	Applikation
CAP	Consistency Availability Partition tolerance
CRDT	Conflict-free replicated data type
DB	Datenbank
DBMS	Datenbank Management System
JSON	JavaScript Object Notation
LWW	Last-Write-Wins
OT	Operational Transformation
PWA	Progressive Web App
UI	User Interface
WLAN	Wireless Local Area Network

GLOSSAR

Bandbreite

gibt an, wie viele Daten pro festgelegter Zeitspanne über ein Netzwerk übertragen werden können.

Hashfunktion

TODO: ist eine Abbildung, die eine große Eingabemenge (die Schlüssel) auf eine kleinere Zielmenge (die Hashwerte) abbildet

Kollaborativ

Als kollaborative Software oder kollaboratives System wird eine Software zur Unterstützung der computergestützten Zusammenarbeit in einer Gruppe über zeitliche und/oder räumliche Distanz hinweg bezeichnet

Latenz

Die Wartezeit, die im Netzwerk verbraucht wird bevor eine Kommunikation beginnen kann, wird als Latenz oder als Netzwerklatenz bezeichnet.

Middleware

Schicht zwischen Anwendung und Betriebssystem.

optimistische Bedienoberfläche

auch: optimistic UI, wartet nicht mit der Aktualisierung der Oberfläche auf das Ende einer Operation. Die zeigt also den gewünschten Zustand der App an, bevor die Anwendung fertig ist indem sie z.B. Fakedaten zeigt.

Progressive Web App

Oder "fortschrittliche Web App" eine mobil nutzbare Webseite, erstellt mit den Webstandards, die als Symbiose aus einer nativen, mobilen Anwendung und einer responsiven Webseite beschrieben werden kann. Die Idee dahinter ist, dass Apps zukünftig nicht mehr über einen App Store, sondern über den Browser installiert werden kann.

Queue

auch Warteschlange, eine Datenstruktur die zur Zwischenspeicherung von Objekten dient. Hierbei wird das zuerst eingegebene Objekt auch zuerst verarbeitet (wie bei einer Warteschlange).

ABBILDUNGSVERZEICHNIS

2.1	Redux Offline	9
-----	-------------------------	---

LITERATURVERZEICHNIS

- [Acu] ACUÑA, Raúl G.: *react-native-offline*. <https://github.com/rauliyohmc/react-native-offline>,. – Zugriff: 12.04.2018 2.2.5
- [ALS10] ANDERSON, J. C. ; LENHARDT, Jan ; SLATER, Noah: *CouchDB: The Definitive Guide*. Sebastopol, CA : O'Reilly Media, 2010. – ISBN 978-0-596-15589-6. – oreilly.com 3.5
- [Ban16] BANK, World: World Development Report 2016: Digital Dividends / International Bank for Reconstruction and Development / The World Bank. Washington DC, 2016. – Research Report. – doi: doi:10.1596/978-1-4648-0671-1 1
- [EG89] ELLIS, C. A. ; GIBBS, S. J.: Concurrency Control in Groupware Systems. In: *SIGMOD Rec.* 18 (1989), jun, Nr. 2, 399–407. <http://dx.doi.org/10.1145/66926.66963>. – DOI 10.1145/66926.66963. – ISSN 0163-5808 3.4.2
- [Evä17] EVÄKALLADO, Jani: Introducing Redux Offline: Offline-First Architecture for Progressive Web Applications and React Native. In: *HACKERnoon* (2017), 3. – Zugriff: 12.04.2018 2.1
- [fal] *Eight Fallacies of Distributed Computing*. <https://blog.fogcreek.com/eight-fallacies-of-distributed-computing-tech-talk/>,. – Zugriff: 12.04.2018 3.1
- [Fra09] FRASER, Neil: Differential Synchronization. Version: Jan 2009. <https://neil.fraser.name/writing/sync/eng047-fraser.pdf>. 2009. – Research Report. – 8 S. 3.4.2
- [hoo] *Hoodie – The Offline First Backend*. <http://hood.ie>,. – Zugriff: 12.04.2018 2.2.7
- [LL10] LI, Du ; LI, Rui: An Admissibility-Based Operational Transformation Framework for Collaborative Editing Systems. In: *Comput. Supported Coop. Work* 19 (2010), feb, Nr. 1, 1–43. <http://dx.doi.org/10.1007/s10606-009-9103-1>. – DOI 10.1007/s10606-009-9103-1. – ISSN 0925-9724 3.4.2
- [off] *Offline First*. <http://offlinefirst.org/>,. – Zugriff: 12.04.2018 1

- [pou] *pouchdb – Tha Database that Syncs!* <https://pouchdb.com/learn>, . – Zugriff: 12.04.2018 2.2.9
- [rea17a] *The Offline-First Approach to Mobile App Development - Beyond Caching to a Full Data Sync Platform.* <https://www2.realm.io/whitepaper/offline-first-approach-registration>, october 2017. – Zugriff: 12.04.2018 2.2.2
- [rea17b] *BUILD BETTER APPS, FASTER WITH REALM - An Overview of the Realm Platform.* <https://www2.realm.io/whitepaper/realm-overview-registration>, october 2017. – Zugriff: 12.04.2018 2.2.2
- [reda] *Redux Offline Release BREAKING: Migrate to Store Enhancer API.* <https://github.com/redux-offline/redux-offline/releases/tag/v2.0.0>, . – Zugriff: 12.04.2018 2.2.4
- [redb] *Redux Offline.* <https://github.com/redux-offline/redux-offline>, . – Zugriff: 12.04.2018 2.2.4
- [redc] *react-native-offline.* <https://github.com/rt2zz/redux-persist>, . – Zugriff: 12.04.2018 2.2.4
- [San16] SANFORD, Clark: Persistence is Key: Using Redux-Persist to Store Your State in LocalStorage. In: *Medium* (2016), 12. – Zugriff: 12.04.2018 2.2.4
- [SPBZ11a] SHAPIRO, Marc ; PREGUIÇA, Nuno ; BAQUERO, Carlos ; ZAWIRSKI, Marek: A comprehensive study of Convergent and Commutative Replicated Data Types / Inria – Centre Paris-Rocquencourt ; INRIA. Version: Jan 2011. <https://hal.inria.fr/inria-00555588>. 2011 (RR-7506). – Research Report. – 50 S. 3.4.3
- [SPBZ11b] SHAPIRO, Marc ; PREGUIÇA, Nuno ; BAQUERO, Carlos ; ZAWIRSKI, Marek: Conflict-free Replicated Data Types. Version: Jul 2011. <https://hal.inria.fr/inria-00609399>. 2011 (RR-7687). – Research Report. – 18 S. 3.4.3, 3.4.3
- [Sto] STOLYAR, Arthur: *offline-plugin.* <https://github.com/NekR/offline-plugin>, . – Zugriff: 12.04.2018 2.2.6

ANHANG

EIDESSTATTLICHE ERKLÄRUNG

CD-INHALT

Auf der beigefügten CD befinden sich

- Die schriftliche Ausarbeitung dieser Masterrarbeit im PDF-Format
- Das erstellte Projekt