Zentrum für sichere Informationstechnologie – Austria
Secure Information Technology Center – Austria

A-SIT

Seidlgasse 22 / 9, 1030 Wien
Tel.: (+43 1) 503 19 63–0
Fax: (+43 1) 503 19 63–66

Inffeldgasse 16a, 8010 Graz
Tel.: (+43 316) 873-5514
Fax: (+43 316) 873-5520

http://www.a-sit.at
E-Mail: office@a-sit.at

DVR: 1035461
ZVR: 948166612
UID: ATU60778947

DEZENTRALE VERFAHREN FÜR GLEICHZEITIGE BEARBEITUNG GEMEINSAMER DATENSTRUKTUREN

Version 1.0 vom 23.07.2021
Bernd Prünster – bernd.pruenster@a-sit.at

Abstract/Zusammenfassung: Systeme wie Google Docs, oder webbasierte Versionen von Microsoft Office haben entscheidend dazu beigetragen, kollaborative Dokumentbearbeitung voranzutreiben. Verteilte Versionskontrollsysteme wie Git aber auch dezentrale Kryptowährungen wie Bitcoin zeigen (wenn auch auf unterschiedliche Weise), dass gleichzeitiges Arbeiten auf einer geteilten Datenbasis auch ohne zentrale Instanz möglich ist, wobei im Rahmen dieser Systeme eine automatische Konfliktbereinigung nur unter bestimmten Bedingungen, bzw. nicht zerstörungsfrei möglich ist.
Speziell ohne zentrale Kontrolle ist konfliktfreie Datenreplikation unter Berücksichtigung von in der Praxis relevanten Randbedienungen nach wie vor Gegenstand aktueller Forschung, wobei in jüngerer Vergangenheit große Fortschritte erzielt wurden. Dieses Dokument klassifiziert Verfahren aus diesem Themenkomplex und analysiert diese im Hinblick auf sichere, verteile Bearbeitung einer gemeinsamen Datenbasis.

Inhaltsverzeichnis

1. Einleitung

Die rasante Weiterentwicklung von Webtechnologien wie HTML5 und Websockets hat neue, verteilte und kollaborativ nutzbare, browserbasierte Anwendungen populär gemacht. Dienste wie *Google Docs*[1] oder Online-Versionen von *Microsoft Office*[2], aber auch Chat- und Videotelefonieplattformen veranschaulichen diesen Trend. Darüber hinaus haben verteile Versionskontrollsysteme wie *Git*[3] einen nachhaltigen Einfluss auf (verteilte) Software-Entwicklungsprozesse ausgeübt, und Projekte wie *IPFS*[4] sowie diverse Kryptowährungen dezentrale verteilte Systeme zusätzlich ins Rampenlicht gerückt.
All diesen Ansätzen ist das gleichzeitige verteile Arbeiten auf einer gemeinsamen Datenbasis gemein. Die konkreten Umsetzungen dieser zentralen Säule von verteilen Systemen im Allgemeinen unterscheiden sich jedoch je nach Anwendungsfall drastisch. Während Webanwendungen zum kollaborativen Bearbeiten von Dokumenten üblicherweise eine weitgehende stabile Verbindung aller Teilnehmer zu einem zentralen Server voraussetzen, sind Systeme wie Git, IPFS und moderne Kryptowährungen dezentral organisiert und kommen vollkommen ohne Abhängigkeiten zu zentralen Instanzen aus. Aus diesen Unterschieden ergeben sich weitreichende Auswirkungen auf die Organisation verteilter Datenbasen und die Nutzerinteraktionsmöglichkeiten. In letzter Zeit findet jedoch auch im Kontext (webbasierter) kollaborativer Dokumentbearbeitung ein Umdenken hin zu dezentral organisierten Konzepten statt. Maßgeblich dafür ist der Forschungs- und Entwicklungsfortschritt im Bereich *Conflict-Free Replicated Data Types* (CRDTs) [1], welcher eine dezentrale Umsetzung derartiger Anwendungen erst möglich macht.

Dieses Dokument gibt einen Überblick über unterschiedliche Konzepte, welche kollaboratives Arbeiten auf einer gemeinsamen Datenbasis ermöglichen. Ausgehend von frühen, serverbasierten Ansätzen wird auch die Wandlung dieses Themenfeldes exemplarisch an Hand eine Reihe ehemals und aktuell relevanter Entwicklungen veranschaulicht. Die Anforderungen an derartige Datenstrukturen sind auf Grund der spezifischen Anwendungsfälle, die diese bedienen, eher spezieller Natur und wurden daher traditionell isoliert von generischen Sicherheitsaspekten wie Accountability, Integrität oder Non-Repudiation behandelt. Speziell im Fall dezentraler Konzepte rücken Fragen nach Konsistenz, Datenintegrität und Konfliktbereinigung in den Vordergrund und haben teilweise auf Grund der verhältnismäßig nur kurz zurückliegenden maßgeblichen Forschungsfortschritte eine Auseinandersetzung mit klassischen Sicherheitsaspekten verdrängt. Ebendiese Fortschritte (und die seit kurzem gegebene Praxistauglichkeit einiger im Rahmen dieses Projekts analysierter Technologien) ermöglichen jedoch eine erste Auseinandersetzung mit diesem Thema.

Verteile Systeme bestehend aus einer Vielzahl von Teilnehmern benötigen aus naheliegenden Gründen Datenstrukturen und Prozesse, welche an dieses Umfeld angepasst sind. Selbst im einfachen Fall einer zentralen Kontrollinstanz, welche exklusiven Schreibzugriff auf einen gemeinsamen Datensatz ohne gleichzeitige Bearbeitungsmöglichkeit kontrolliert, müssen entsprechende Zugriffsprozesse vorgesehen werden. Die Komplexität solcher Lösungen steigt, umso interaktiver und kollaborativer die Bearbeitung geteilter Daten ausfällt. Speziell beim simultanen Bearbeiten von Dokumenten sind die Anforderungen hoch, da das Verhalten eines solchen Systems nicht nur technisch korrekt, sondern auch für Nutzerinnen und Nutzer intuitiv nachvollziehbar sein muss. Auf der anderen Seite lassen verteilte Versionskontrollsysteme wie Git eine direkte, gleichzeitige Bearbeitung derselben Details einer gemeinsame Codebasis nur eingeschränkt zu: Wenn eine automatisierte Konfliktbehebung (welche innerhalb derselben Codeabschnitte nur erfolgreich ist, wenn unterschiedliche Zeilen bearbeitet wurden) fehlschlägt, wird ein Konflikt gemeldet, welcher von der Nutzerin, bzw. vom Nutzer manuell behoben werden muss. Das zu Grunde liegende System ist somit zwar verteilt, eine direkte Kollaboration wird jedoch nur bedingt, bzw. mit relativ grober Granularität unterstützt.

[1] https://docs.google.com/
[2] https://www.microsoft.com/en-us/microsoft-365/
[3] https://git-scm.com/
[4] https://ipfs.io/

Die nachfolgenden Abschnitte zeigen die Charakteristika unterschiedlicher Typen verteilter Datenstrukturen auf und gehen im Detail auf deren Stärken und Schwächen ein. Insbesondere werden Abhängigkeit zu zentralen Instanzen sowie Praktikabilität diskutiert.

2. Verteilte Datenstrukturen

Verteilte Datenstrukturen zeichnen sich im Allgemeinen dadurch aus, dass keine zentrale Instanz benötigt wird, um eine Bearbeitung durch mehrere unabhängige Teilnehmerinnen, bzw. Teilnehmer zu koordinieren. Die bekanntesten Vertreter dieser Kategorie sind verteilte Hashtabellen (*Distributed Hash Tables*; DHTs) [2], sowie von Hash-Bäumen [3] abgeleitete Datenstrukturen, wie *Merkle DAGs*, *Merkle Trees* und die Blockchain. Allen Ansätzen ist die limitierte Grundfunktionalität, sowie nur beschränkt mögliche, bzw. radikale automatisierte Auflösung von Konflikten gemein.

2.1. Verteilte Hashtabellen

Verteilte Hashtabellen (Distributed Hash Tables, DHTs) basieren im Kern auf demselben Prinzip wie reguläre Hashtabellen: Um ein Datum zu speichern, wird dessen Hashwert berechnet, aus welchem sich die Position des Datums innerhalb der Tabelle ergibt. Allerdings weisen verteilte Hashtabellen zwei Besonderheiten auf:

1. Der Schlüssel, an Hand dessen Daten identifiziert werden, entspricht dem Hashwert der Daten. Diese Eigenschaft wird auch als *Content-Addressibility* bezeichnet, da der Identifikator eines Datums unmittelbar und ausschließlich von dessen Inhalt abhängig ist. Es ist somit im Gegensatz zu regulären Hashtabellen nicht möglich, Daten unter frei definierbaren Schlüsselwörtern abzulegen und abzurufen.

2. Die Speicherung selbst erfolgt verteilt, üblicherweise innerhalb eines Peer-to-Peer (P2P)-Netzwerks. Dabei wird die Entscheidung, wo, bzw. von welchem Netzwerkknoten ein Datum gespeichert werden soll, auf Basis einer Distanzfunktion gefällt: Im Wesentlichen ist derjenige Knoten für die Speicherung zuständig, dessen Identifikator laut verwendeter Distanzfunktion dem Hashwert des Datums am nächsten ist. Folglich lässt sich nicht frei bestimmen, welcher Netzwerkknoten für die Speicherung eines bestimmten Datums verantwortlich ist.

Zweitere Eigenschaft ist dabei unmittelbar von der ersten abhängig und macht effizientes Auffinden gespeicherter Daten erst möglich. Konfliktbehebung ist als Konsequenz dieser Strategie ebenfalls in der Praxis irrelevant: Einerseits werden identische Daten automatisch dedupliziert (da deren Hashwert ident ist, und daher ein erneutes Hinzufügen mehrfach vorhandener Daten schlicht keine Auswirkung hat). Andererseits führt eine Datenänderung unweigerlich zur Änderung des Hashwerts, was wiederum die unabhängige Speicherung zur Folge hat. Insgesamt handelt es sich daher bei verteilten Hashtabellen um selbstorganisierende Datenstrukturen, welche keiner zentralen Kontrolle bedürfen. Inkonsistenzen können lediglich dadurch auftreten, dass sich einzelne Teilnehmerinnen und Teilnehmer unehrlich verhalten und Anfragen nach Daten nicht wahrheitsgemäß beantworten. Die Integrität abgerufener Daten lässt sich jedoch trivial verifizieren, indem deren Hashwert berechnet, und mit den zur Lokalisierung verwendeten abgeglichen wird. Kommen und Gehen von Teilnehmerinnen und Teilnehmern wird durch Replikation kompensiert. Im einfachsten Fall wird ein Datum nicht an einem Netzwerkknoten, sondern an den k Knoten, deren Identifikatoren dem Hashwert des Datums am nächsten sind, gespeichert. Durch periodisches Republishing wird eine regelmäßige Neuverteilung und Replikation über die aktuell im Peer-to-Peer-Netzwerk befindlichen Knoten sichergestellt. Weitere Details zu den Eigenschaften und die Verwendung von DHTs im Kontext dezentraler Peer-to-Peer-Netze können einem vorangegangenen A-SIT-Projekt zum Thema entnommen werden [4].

Für kollaboratives Arbeiten auf einem gemeinsamen Datensatz eignen sich verteile Hashtabellen auf Grund ihrer beschränkten Funktionalität nicht ohne Weiteres. Vielmehr sind sie als Fundament für den Betrieb dezentraler Peer-to-Peer-Netzwerke essentiell, welche das dezentrale Vernetzen von Teilnehmerinnen und Teilnehmern unabhängig von der IP-Netzwerktopologie erst ermöglichen.

Populäre DHT-Implementierungen finden sich beispielsweise im Kontext von Filesharing per *BitTorrent* [5] und als Kernkomponente von IPFS.

2.2. Hashbaumartige Strukturen

Während verteile Hashtabellen die unabhängige, dezentrale Speicherung von in sich abgeschlossenen Datenfragmenten (wie einzelnen Dateien) ermöglichen, erlauben Hashbäume (nach ihrem Erfinder Ralph Merkle auch *Merkle-Trees* genannt) und hashbaumartige Strukturen die Verteilung, bzw. Verkettung zusammenhängender Daten. Zum einen lässt sich dadurch beispielsweise eine große Datei auf mehrere Teilnehmerinnen und Teilnehmer eines Peer-to-Peer-Netzwerks verteilen. Andererseits lassen sich dadurch auch (verteilte) append-only Datenstrukturen umsetzen; Datenstrukturen, welche zwar scheinbar beliebig bearbeitet werden können, technisch jedoch alle Änderungen protokollieren. In beiden Fällen kann nach den Datenabruf, selbst wenn dieser aus nicht vertrauenswürdigen Quellen erfolgt, die Integrität der abgerufenen Daten, ähnlich wie im Fall von verteilten Hashtabellen, garantiert werden. Im Falle einer Integritätsverletzung lässt sich sogar der Bereich, in den diese erfolgt ist, eng eingrenzen.

2.2.1. Hashbäume

Ausgangspunkt für alle derartigen Datenstrukturen sind so genannte Hashbäume. Ursprünglich diente die von Ralph Merkle 1979 erdachte Struktur dem Schlüsselmanagement im Rahmen von Lamport-Einmalsignaturen [3]. Im Kontext dezentraler Datenspeicher wurden Hashbäume bereits in der Blütezeit des Filesharing erfolgreich eingesetzt, um zu verifizieren, ob eine Datei, deren Segmente aus verschiedenen (üblicherweise nicht vertrauenswürdigen Quellen) bezogen wurden, auch integer ist. Grundlage bildet dabei die kryptografische Verkettung von Datensegmenten nach folgendem Schema:

- Eine Datei wird in gleich große Blöcke partitioniert, welche unabhängig voneinander gespeichert und abgerufen werden können.
- Im Zuge der Partitionierung wird von jedem Block ein kryptografischer Hashwert berechnet.
- Anschließend werden (im Falle eines binären Hashbaums) immer zwei Hashwerte konkateniert und erneut gehasht, wodurch sich die namensgebende Baumstruktur ergibt. Der Wurzelknoten des Baums fungiert als eindeutiger Identifikator der gehashten Daten, analog zur Content-Addressibility von verteilten Hashtabellen.

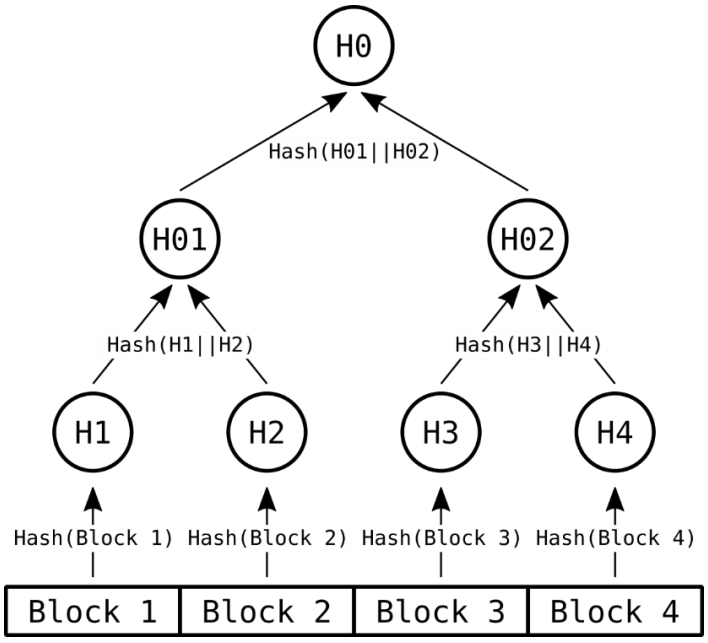

Abbildung 1: Schematische Darstellung eines binären Hashbaums

Eine Integritätsprüfung erfolgt ebenfalls über die Erzeugung einer solchen Baumstruktur; eine Modifikation der Daten hat eine Änderung des Identifikators zur Folge, wodurch sich schnell eine generelle Aussage bezüglich der Datenintegrität machen lässt. Wird eine Integritätsverletzung festgestellt, kann das Segment, bzw. können die Segmente, welche fehlerhaft sind, mittels binärer Suche über den Hashbaum effizient ermittelt werden. Je nachdem wie detailliert die Struktur (bzw. die Werte der Blätter) eines von einer partitionierten Datei erzeugten Hashbaums im Rahmen der Verifikation bekannt ist, umso exakter können eventuelle Integritätsverletzungen lokalisiert werden. Durch die Wahl einer kleineren Blockgröße können Fehler noch exakter eingegrenzt werden. Allerdings steigt der Overhead bei kleiner werdender Blockgröße. Eine schematische Darstellung eines Hashbaums ist in Abbildung 1 skizziert. Ein gemeinsames Bearbeiten oder direkte Kollaboration auf Basis von Hashbäumen ist nicht vorgesehen.

2.2.2. Blockchain

Eine Datenstruktur, welche einige Gemeinsamkeiten mit Hashbäumen aufweist und seit einigen Jahren besonders hohe Aufmerksamkeit erfährt, ist die von der dezentralen Kryptowährung *Bitcoin* [6] populär gemachte *Blockchain*. Dabei handelt es sich eben um keine Baumstruktur, sondern um eine lineare kryptografische Verkettung von Datenblöcken, wobei jeder Block den Hashwert seines Vorgängerblocks enthält. Ist (ähnlich wie im Fall eines Hashbaums), der Hashwert der Spitze (bzw. der aktuellste Block) bekannt, lassen sich nach Erhalt aller vorangegangenen Blöcke Ungereimtheiten feststellen und auf betroffene Blöcke eingrenzen. Im Kontext moderner Kryptowährungen wird die Blockchain als append-only Transaktionsregister eingesetzt, um zu verhindern, dass Transaktionsdaten nachträglich manipuliert werden können. Unter den Gesichtspunkten der Anforderungen, welche an verteilte Datenstrukturen gestellt werden, handelt es sich dabei um ein vergleichsweise primitives Konstrukt. Insbesondere die Manipulationssicherheit ergibt sich in der Praxis nicht aus inhärenten Eigenschaften der Datenstruktur selbst, sondern aus von ökonomischen Interessen getriebenem Schwarmverhalten der Nutzergemeinschaft von Kryptowährungen. Auch das Fortführen dieses Transaktionsregisters ist von ebendiesen ökonomischen Interessen und zugehörigem Schwarmverhalten getrieben – eine tatsächliche Kollaboration in der Bearbeitung, bzw. Fortführung eine Blockchain ist nicht gegeben, da zu jedem Zeitpunkt immer nur eine Instanz einen gültigen Block produzieren kann, welcher auch akzeptiert wird. Vereinfacht formuliert gilt schlichtweg das Recht des Stärkeren, wodurch eine Blockchain einerseits eine völlig ungeeignete Basis für kollaboratives Arbeiten ist, andererseits jedoch keine Abhängigkeiten auf eine zentrale Instanz aufweist.

2.2.3. Merkle DAG

Eine zur Blockhain konträre Adaption des Hashbaum-Konzepts sind sogenannte *Merkle-DAGs*, wobei DAG für *Directed Acyclic Graph* (ungerichteter, azyklischer Graph) steht. Dabei handelt es sich schlicht um eine Verallgemeinerung von Hashbäumen auf eine komplexere Graphstruktur, welche nicht der Einschränkung eines einzelnen Wurzelknoten unterliegt. Der wohl bekannteste und erfolgreichste Einsatz dieser Datenstruktur ist das verteile Versionskontrollsystem Git. Obwohl oft als Paradebeispiel für kollaboratives dezentrales, verteiltes Arbeiten angeführt, lässt insbesondere der Kollaborationsaspekt in Teilen zu wünschen übrig. Analog zum Blockchain-Fall, nutzt Git einen Merkle-DAG intern als append-only Datenstruktur. Im Kontext von Git werden Änderungen an Daten als Knoten in diesem Graph abgebildet, wobei auch Erstellen und Entfernen als Änderungen gelten. Darüber hinaus wird eine Codebasis selbst als Baumstruktur (als Merkle-Baum) abgebildet (was sich mit der Organisation eines typischen hierarchischen Dateisystems, bzw. Ordnerstrukturen deckt). Änderungen an der Codebasis können durch diese Doppelstruktur einerseits protokolliert, andererseits effizient abgebildet werden. Weiters ist ein schlichtes Verschieben von Daten effizient als solches detektierbar und zueinander in Konflikt stehende Änderungen können unabhängig davon, wie umfangreich eine als Änderung im Merkle-DAG abgebildete Aktion ausfällt, auf die betroffenen Dateien eingegrenzt werden. Wie in der Einleitung erwähnt, ist eine automatisierte Konfliktbehebung innerhalb von Dateien üblicherweise jedoch nur möglich, so lange lediglich unterschiedliche Codezeilen bearbeitet wurden. Eine Echtzeitkollaboration, wie beispielsweise das gemeinsame Bearbeiten eines Dokuments ist somit nicht möglich und verlangt nach speziell auf solche Szenarien ausgelegten Datenstrukturen. Der nachfolgende Abschnitt beleuchtet diese näher.

2.3. Auf Echtzeitkollaboration ausgelegte Systeme

Verteilte Versionskontrollsysteme wie Git haben kollaborative Softwareentwicklung in großen Teams erst praktikabel gemacht. Es gibt jedoch Anwendungsfälle, welche höhere Ansprüche an die (für Menschen nachvollziehbare) automatisierte Behebung von Konflikten und Echtzeitfähigkeit beim kollaborativen Arbeiten auf gemeinsamen Daten stellen. Systeme wie Google Docs oder die Online-Version von Microsoft Office sind, wie eingangs erwähnt, die klassischen Beispiele dafür. In diesem Zusammenhang ist die Bezeichnung Datenstruktur irreführend, bzw. unpräzise, da es ganzer Software-Stacks bedarf, um derartige Funktionalität bereitzustellen. Grundsätzlich ist in diesem Zusammenhang zwischen serverbasierten und dezentralen Lösungen zu unterscheiden, wobei letztere erst kürzlich praktikabel wurden.

2.3.1. Serverbasierte Konzepte

Serverzentrische interaktive Kollaborationslösungen basieren üblicherweise auf sogenannter *Operational Transformation* (OT) [7], welche erstmals 1989 beschrieben wurde. OT-basierte Systeme setzen im Wesentlichen persistente Verbindungen aller Teilnehmerinnen und Teilnehmer zu einem zentralen Server voraus, welcher eingehende Änderungen verarbeitet, auf der Datenbasis umsetzt, und schließlich an alle Teilnehmerinnen und Teilnehmer verteilt. Diese harte Abhängigkeit zu einer zentralen Instanz erlaubt es, verhältnismäßig einfache und vor allem effiziente, hochperformante Verfahren zur Sicherstellung eines kohärenten und über alle Teilnehmerinnen und Teilnehmer hinweg konsistenten Zustands anzuwenden. Insbesondere der Performance-Aspekt war über Jahrzehnte hinweg ein Grund, warum sich dezentrale Alternativen nicht durchsetzen konnten. Allerdings muss in diesem Zusammenhang erwähnt werden, dass *verhältnismäßig einfach* in der Praxis immer noch ein hochkomplexes System beschreibt: „*Due to the need to consider complicated case coverage, formal proofs are very complicated and error-prone, even for OT algorithms that only treat two characterwise primitives (insert and delete)*“[5] [8]. Darin liegt auch die Tatsache begründet, dass viele Weiterentwicklungen dieser Technologie konzeptionelle, bzw. formale Fehler und unzureichende Performance bei in der Praxis auftretenden Spezialfällen vorangegangener Umsetzungen aufdeckten. Über die Jahre hinweg hat sich dadurch ein regelrechter Wildwuchs unterschiedlicher Implementierungen gebildet: „*Unfortunately, implementing OT sucks. There's a million algorithms with different tradeoffs, mostly trapped in academic papers*“[6]. Nichts desto trotz sind derartige Systeme ungleich einfacher umzusetzen, zu durchschauen und formal auf Korrektheit zu prüfen als dezentrale Ansätze. Nach wie vor dominieren OT-basierte Kollaborationstools, weshalb deren Funktionsprinzipien nachfolgend beschrieben werden.

Da Operational Transformations ihren Ursprung in kollaborativer Echtzeittextverarbeitung haben, lassen sich deren grundlegenden Eigenschaften auch an Hand eines Beispiels aus diesem Kontext illustrieren. Abbildung 2 veranschaulicht dabei die Problematik, welche bereits entsteht, wenn lediglich zwei Parteien gleichzeitig ein Dokument bearbeiten: Während beide Parteien initial noch am selben Stand („abc“) sind, reicht bereits ein unkoordiniertes Hinzufügen und Löschen einzelner Zeichen aus, um Inkonsistenzen zu erzeugen. Berücksichtigt man in diesem Kontext noch Netzwerklatenzen, wird offensichtlich, dass eine unreflektierte Anwendung einzelner Operationen über alle Parteien hinweg schnell von Inkonsistenzen hin zu völligem Chaos führt. Operational Transformations verhindern dies, indem sie jeglichen direkten Austausch von Änderungen zwischen Instanzen strikt verbieten. Stattdessen wird eine zentrale Instanz eingeführt, welche Operationen derart adaptiert („transformiert“), dass allen Instanzen dieselbe Sicht auf die gemeinsame Datenbasis garantiert wird.

[5] „Durch die Notwendigkeit komplizierte Fälle abzudecken, sind formale Beweise kompliziert zu führen und fehleranfällig, selbst für OT-Algorithmen, welche lediglich zwei buchstabenweise Primitive behandeln (Einfügen und Löschen)“

[6] „Leider nervt das Implementieren von OT. Es gibt eine Million Algorithmen, jeder mit unterschiedlichen Einschränkungen, meist in akademischen Publikationen gefangen“, siehe https://sharejs.org/

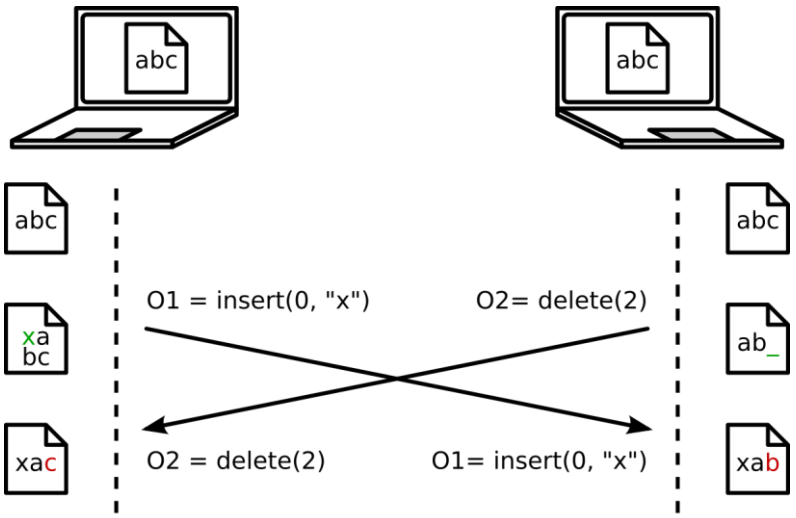

Abbildung 2: Problematik beim gleichzeitigen Bearbeiten eines Textdokuments

Abbildung 3 veranschaulicht diese Strategie und hebt auch die Transformationen hervor, welche nötig sind, um die sich im Gegensatz zum direkten Propagieren von Änderungen zwischen Instanzen einschleichenden Unregelmäßigkeiten zu verhindern. Einerseits lassen sich durch den Zwang, alle Änderungen zentral zu verarbeiten, zu transformieren und die Resultate koordiniert zu verteilen, Konflikte und Inkonsistenzen effektiv vorbeugen. Andererseits wird dadurch verunmöglicht, dass beispielsweise Änderungen zwischen zwei Nutzerinnen oder Nutzern, die sich im selben Netzwerk befinden, direkt ausgetauscht werden: „*But, really, I don't want to have to wait for messages to be sent to a server on a different continent in order to just exchange data between my phone and my laptop which are fifty centimetres apart from each other*“[7] [9].

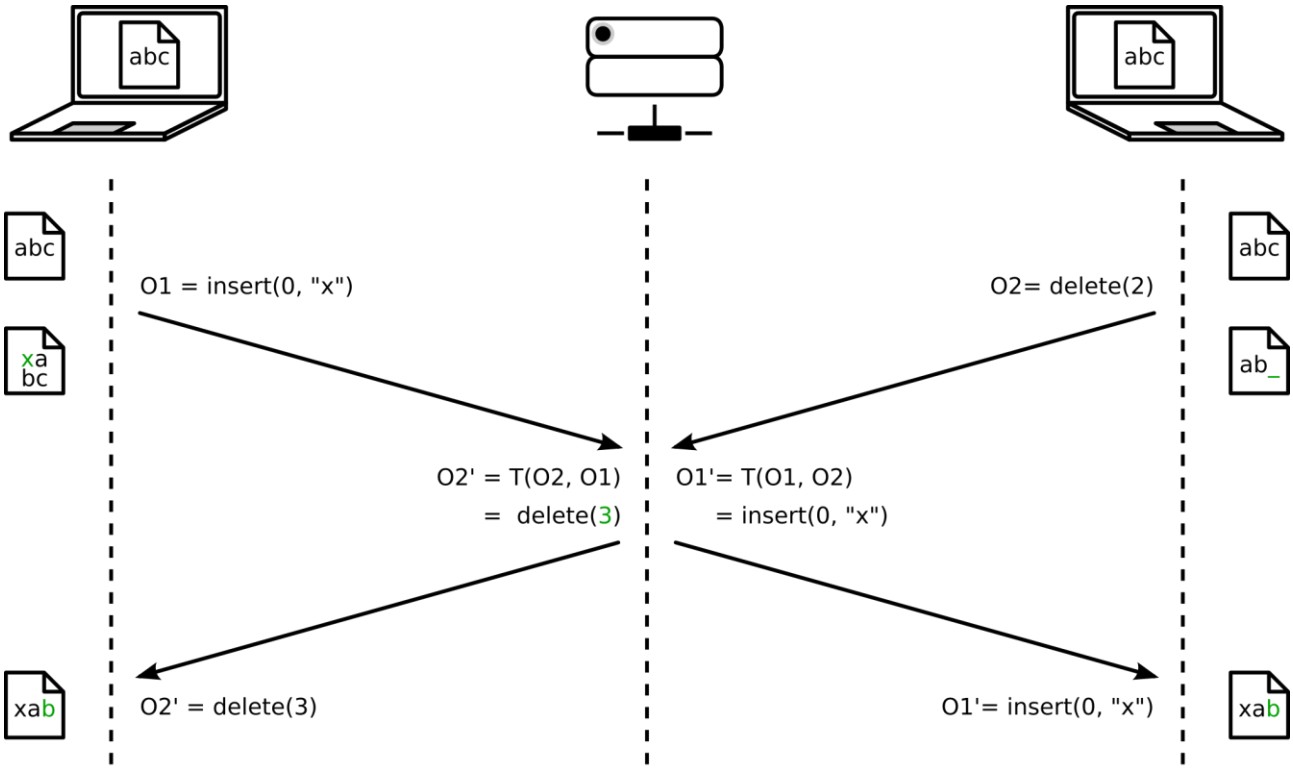

Abbildung 3: Funktionsprinzip von Operational Transformations

[7] „Doch in Wirklichkeit möchte ich nicht darauf warten müssen, dass Nachrichten an einen Server auf einem anderen Kontinent gesendet werden, nur um Daten zwischen meinem Telefon und meinem Laptop auszutauschen, welche fünfzig Zentimeter voneinander entfernt sind.“

In einem einfachen wie in Abbildung 2 dargestelltem Vergleich beider resultierender Zustände ist noch relativ klar nachvollziehbar, dass ein Off-By-One-Fehler durch einen verschobenen Index der Grund für die inkonsistenten Zustände ist. Solche Fälle ließen sich durch den Einsatz naiver Strategien auch ohne zentral koordinierende Instanz noch relativ einfach abdecken. An dezentralen Systemen mit vergleichbarer Funktionalität wurde hingegen jahrelang intensiv geforscht, um an einen Punkt zu gelangen, der einen Produktiveinsatz möglich macht. Der nachfolgende Abschnitt geht im Detail auf solche Verfahren ein.

2.3.2. Dezentrale Systeme

Eine unmittelbare Konsequenz aus dem Wegfall einer zentralen Instanz ist, dass jegliche Verarbeitung direkt auf Endgeräten von gemeinsam Arbeitenden stattfinden muss. Dass sich hierdurch ein erhöhter Rechenbedarf ergibt, erscheint offensichtlich, allerdings geht diese Verlagerung insbesondere mit einem potentiell enorm erhöhten Speicherbedarf einher. Frühe Systeme zur dezentralen Kollaboration, welche vornehmlich von akademischer Relevanz waren, haben bereits nach relativ wenigen, auf alle Teilnehmerinnen und Teilnehmer replizierten Änderungen mehrere Größenordnungen mehr Speicherplatz benötigt, als der aktuelle Datensatz an sich. Mittlerweile sind diese Schwächen zwar großteils behoben, die ursprüngliche Ursache für dieses Verhalten liegt jedoch im Konsistenzmodell der verwendeten Konzepte begründet.

(Strong) Eventual Consistency

Mangels einer zentralen koordinierenden Instanz können weniger strenge Garantien bezüglich Aktualität und Konsistenz der bei Teilnehmerinnen und Teilnehmern lokal replizierten Daten gemacht werden. Als Mindestanforderung gilt in diesem Kontext *Eventual Consistency*. Dieses Konsistenzmodell garantiert dem Namen entsprechend, dass alle lokale Datenreplikate irgendwann konvergieren, sofern lange genug keine Änderungen mehr durchgeführt werden. Unter dem Gesichtspunkt der Echtzeitkollaboration ist von verhältnismäßig niedrigen Netzwerklatenzen auszugehen, wodurch dieser Zeitrahmen typischerweise im Sekundenbereich liegt.
Allerdings gibt Eventual Consistency keine Einschränkungen bezüglich Rollback-Verhalten o.Ä. vor. In der Praxis bedeutet dies, dass man lokal keine Aussage treffen kann, ob der aktuell beobachtete Stand auch tatsächlich aktuell ist. In Folge dessen ist nicht garantiert, dass eine lokal ausgeführte Änderung tatsächlich gültig ist und auch konfliktfrei an alle anderen Instanzen propagiert werden kann. Darunter leidet nicht nur die Nutzerfreundlichkeit und die intuitive Nachvollziehbarkeit des Verhaltens eines solchen Systems, sondern auch die Performance. Insbesondere während der Kollaboration mit Teilnehmerinnen und Teilnehmern, welche über instabile Verbindungen verfügen, kann dies zu Problemen führen, da ein derartiges System in solchen Fällen potentiell größtenteils mit der Konfliktbehebung beschäftigt ist. Je nach Implementierung handelt es sich dabei um Operationen von exponentieller Speicherplatz- und Zeitkomplexität, deren Konstanten und Faktoren in der Praxis zu einem tatsächlichen Laufzeit- und Speicherbedarf führen, welcher schlichtweg unpraktikabel ist. Darüber hinaus ist auch die algorithmische Komplexität derartiger Verfahren sehr hoch, weshalb sich die Implementierung einer korrekten und vor allem in der Praxis nutzbaren Lösung schwierig gestaltet.

Ein schärfer abgestecktes Konsistenzmodell ist *Strong Eventual Consistency*, welches auf elementarer Ebene inhärent konfliktfrei arbeitet, da Konflikte per se nicht vorkommen können. Alle Datenreplikate konvergieren folglich monoton, was wiederum bedeutet, dass Rollbacks überflüssig werden. Was im ersten Moment unmöglich erscheint, ergibt sich schlicht aus der Beschränktheit der Datenstrukturen, auf welche dieses Konsistenzmodell anwendbar ist.

Conflict-Free Replicated Data Types

Unter dem Überbegriff *Conflict-Free Replicated Data Types* (CRDTs) werden Datenstrukturen im Kontext von Echtzeitsystemen zur Kollaboration zusammengefasst, welche die Voraussetzung der Strong Eventual Consistency erfüllen. Generell können CRDTs an Hand ihres Replikationsmodells in unterschiedliche Kategorien klassifiziert werden. Diese sind funktional äquivalent, stellen jedoch unterschiedliche Anforderungen an den Netzwerk-Layer und unterscheiden sich im verursachten

Overhead. Derartige technische Details sind jedoch im Wesentlichen nur relevant, wenn es darum geht, sich auf Basis der Betriebsumgebung für ein Modell zu entscheiden, nicht jedoch auf konzeptioneller Ebene. Die nachfolgend beschriebenen Datenstrukturen können in jedem Fall von unterschiedlichen CRDT-Varianten zur Verfügung gestellt werden. Die garantierte Konfliktfreiheit ermöglicht durch Komposition simpler CRDTs erst die Konzeption komplexerer Strukturen.

Zähler

Die elementarsten CRDTs sind Zähler, wobei der so genannte *G-Counter* (Grow-Only Counter) einer Art minimalste nützliche Funktionalität darstellt. Wie der Name vermuten lässt, handelt es sich dabei um einen Zähler, welcher ausschließlich inkrementierbar ist. Im Wesentlichen wird jedem Teilnehmer und jeder Teilnehmerin ein persönlicher Zähler zugewiesen, welcher beliebig inkrementiert werden kann. Der aktuelle globale Zählerstand entspricht schlicht der Summe über alle individuellen Zähler. Ein konvergenter Zustand lässt sich auf Grund der Monotonie der individuellen Zähler ebenfalls einfach erreichen, indem im Falle widersprüchlicher Updates schlicht der höhere Wert akzeptiert wird.
Zwei G-Counter lassen sich trivial zu einem so genannten *PN-Counter* (Positive-Negative Counter) kombinieren: Ein G-Zähler führt über Inkremente Buch, während ein weiterer G-Zähler Dekremente protokolliert. Der Gesamtzählerstand ergibt sich über eine simple Subtraktion des zweiten Zählerstands vom ersten.

Mengen

Analog zu verteilten Zählern lassen sich verteilte, konvergierende Mengen umsetzen. Die Basis bildet auch hier eine elementare, monoton wachsende Struktur. Dieses so genannte *G-Set* (Grow-Only Set) erlaubt lediglich das Einfügen von Elementen, wobei die formale Definition einer Menge selbst innerhalb eines (unkoordinierten) verteilten Systems Konfliktfreiheit garantiert, solange nur das Hinzufügen gestattet ist: Da jedes Element nur einmal in einer Menge vorhanden sein kann, bzw. mehrfaches Hinzufügen keine Auswirkung hat, gibt es schlicht kein Konfliktpotential. Um auch das Entfernen von Elementen zu unterstützen, können zwei G-Sets analog zu PN-Countern zu einem *2P-Set* (Two-Phase Set) kombiniert werden. Dabei protokolliert eine Menge das Hinzufügen, während eine Zweite alle entfernten Elemente aufnimmt. Daraus ergibt sich jedoch direkt eine Einschränkung im Vergleich zu regulären Mengen: Ein einmal als entfernt aufgenommenes Element kann nicht mehr wieder hinzugefügt werden.

Komplexere Funktionalität

Die inhärente Konfliktfreiheit elementarster CRDTs ermöglicht prinzipiell die Komposition zu komplexeren Strukturen. Beispielsweise lassen sich Mengen um zeitliche Informationen zu Einfüge- und Entfernoperationen erweitern, wodurch ein erneutes Hinzufügen von bereits entfernten Elementen ermöglicht wird. Ungeachtet der potentiellen algorithmischen Komplexität, ist ein offensichtlicher Nachteil, der sich aus der Komposition monoton wachsender Grundelemente ergibt, dass der Speicherbedarf ebenfalls monoton anwächst, selbst wenn beispielsweise eine konvergierte Menge leer erscheint. Da ein unbeschränktes Wachstum insbesondere im Produktivbetrieb wenig praktikabel ist, wurden Optimierungsstrategien entwickelt, welche intern zwar Zähler immer weiter erhöhen, jedoch gelöschte Elemente nicht mehrfach speichern müssen, sondern lediglich die Elemente einer Menge referenzieren. Auch wenn Elemente nie wirklich gelöscht werden, ergibt sich dadurch nicht notwendigerweise ein Nachteil. Im Gegenteil kann es unter bestimmten Voraussetzungen vorteilhaft sein, eine lückenlose Änderungshistorie zur Verfügung zu haben. Darüber hinaus ist es selbst in dezentralen Szenarien möglich, Zustände zu identifizieren, die jedem Teilnehmer und jeder Teilnehmerin bekannt sind, sodass die dahinter zurückliegende Änderungshistorie verworfen werden kann, falls dies gewünscht ist. In jedem Fall müssen komplexere CRDTs auf einzelne Szenarien spezialisiert werden, um auch nachvollziehbares Verhalten zu produzieren, was besonders auf die Konfliktbehebung zutrifft.

Produktiveinsatz

Eine Erweiterung auf geordnete Mengen und Maps ermöglicht schließlich komplexere Systeme wie kollaborative Echtzeittextverarbeitung ohne zentrale Instanzen. Zwar liegen solchen Systemen bereits auf rein technischer Ebene komplexe Konzepte zu Grunde, die wahre Herausforderung ist in diesem Kontext jedoch, ein für Nutzerinnen und Nutzer nachvollziehbares Verhalten zu implementieren. Mittlerweile gibt es jedoch mehr als nur eine Lösung, welche auch in der Praxis nutzerfreundliches Verhalten zeigt und auch produktiv eingesetzt wird. Zahllose Optimierungen haben dazu ihr Übriges beigetragen, sodass auch der Overhead (sowohl was den Rechenbedarf, als auch den benötigten Speicherplatz angeht) auf ein akzeptables Niveau reduziert werden konnte. In derart komplexen Szenarien können jedoch in der Praxis wieder Konflikte entstehen, wobei es für viele (nicht jedoch alle) Möglichkeiten automatisierte Lösungsstrategien (wie z.B. Last Write Wins) gibt. Beispiele für praxistaugliche CRDTs und CRDT-basiere Systeme sind *Automerge*[8] und *Redis*[9].

3. Sicherheitsaspekte

An sich ist Protokollkonformität im Rahmen dezentraler, CRDT-basierter Kollaborationssysteme Sache des bzw. der Einzelnen. Monotonieverletzendes Verhalten kann jedoch in weiten Teilen trivial erkannt und verworfen werden – Konvergenz ist in jedem Fall garantiert. Aus rein organisatorischen Gründen protokollieren sowohl CRDT-Primitive, als auch komplexe, aus Komposition hervorgegangene Strukturen die Autorenschaft aller Änderungen. Eine kryptografische Bindung zwischen Änderungen und Autor, bzw. Autorin ist jedoch nicht gegeben. Falls nur schwache Echtzeitanforderungen an ein System gestellt werden, lässt sich dieses Problem einfach lösen, indem z.B. nur kryptografisch signierte Änderungen von anderen Teilnehmerinnen und Teilnehmern akzeptiert werden. Beispielsweise könnten Änderungen immer mindestens über einen Zeitraum von einer Minute akkumuliert und gesammelt signiert verteilt werden. Wirklich praktikabel ist ein solcher Ansatz gerade im Kontext von dezentraler Echtzeitkollaboration jedoch nur in Ausnahmefällen; besonders im Kontext von Textverarbeitung, wenn es auf möglichst verzögerungsfreie Kommunikation von kleinsten Änderungen ankommt, ist es schwierig, einen praxistauglichen Kompromiss zwischen Verzögerung und Overhead zu finden.
Ein weiterer Ansatz, welcher das Problem unverhältnismäßig großer Signaturen im Vergleich zu den kommunizierten Änderungen umgeht, ergibt sich, wenn Effizienz am Netzwerklayer eine stark untergeordnete Rolle spielt und statt eines (effizienten) Broadcasting-Konzepts auf Basis eines Peer-to-Peer-Netzwerks, ein Mesh-Netzwerk eingesetzt wird. Setzt man direkte Verbindungen zwischen allen Kommunikationsteilnehmern und Kommunikationsteilnehmerinnen voraus und unterbindet indirekte Weitergabe von Änderungen über Relaying, bzw. übliche Broadcast-Konzepte, können effiziente kryptografische Verfahren eingesetzt werden, welche die Autorenschaft von Änderungen zweifelsfrei nachvollziehbar machen. Diese Idee lässt sich an Hand eines G-Counters mit zwei Teilnehmern illustrieren und auf größere Teilnehmerzahlen extrapolieren:

1. Partei A und Partei B führen einen Diffie-Hellman-Schlüsselaustausch durch, um einen gemeinsamen geheimen Schlüssel zu erhalten.

2. Beide Parteien bauen einen wechselseitig authentifizierten (verschlüsselten) Kommunikationskanal auf Basis des errechneten Schlüssels zueinander auf. Maximale Effizienz wird in der Regel durch den Einsatz einer Stromchiffre zur Verschlüsselung des Datenverkehrs erreicht.

3. Dieser Kommunikationskanal wird für den Austausch aller Änderungen herangezogen.

4. Partei A akzeptiert nur Änderungen am internen Zähler von Partei B, wenn diese auch tatsächlich über den Kommunikationskanal zwischen A und B eintreffen (und umgekehrt).

[8] https://github.com/automerge
[9] https://redis.io/

5. Dieses Prinzip lässt sich (theoretisch) auf beliebig viele Kommunikationsparteien erweitern, was verhindert, dass Dritte im Namen Anderer (falsifizierte) Änderungen einbringen können.

Ein inhärenter Nachteil dieser Strategie ist die erzwungene quadratische Komplexität betreffend der benötigten Verbindungen und des Netzwerkverkehrs mit der Anzahl der involvierten Parteien. Schließlich muss jede Partei ihre Änderungen jeder anderen Partei kommunizieren und entsprechend viele direkte Verbindungen aufbauen. Ist dezentrale Operation kein Anliegen, lassen jedoch sich nahezu beliebige etablierte Sicherheitskonzepte wie im Rahmen einer jeden traditionellen Client-Server-Anwendung durchsetzen, wodurch derartige Schwierigkeiten gar nicht erst auftreten. Im Kontext dezentraler Systeme sind Sicherheitsfragen jedoch nach wie vor Gegenstand aktiver Forschung. Oftmals stehen jedoch nicht dezentrale Governance-Fragen, sondern der dezentrales Datenaustausch im Vordergrund, weshalb aus aktueller Sicht fraglich ist, welche praktische Relevanz derartige Sicherheitsfragen in der Praxis haben.

4. Fazit

Je nach Anforderung haben sich unterschiedliche Konzepte zur kollaborativen Bearbeitung einer gemeinsamen Datenbasis entwickelt. Insbesondere verteilte Versionskontrollsysteme wie Git erfreuen sich dabei großer Popularität und gemeinsame, browser-basierte Texterarbeitung ist mittlerweile ebenfalls ein etabliertes Arbeitsmodell. Besonders der Produktiveinsatz von letzteren Anwendungen, welche hohe Anforderungen an die Echtzeitfähigkeit der darunterliegenden Systeme stellen, war noch vor wenigen Jahren gänzlich ohne Abhängigkeiten auf zentrale Instanzen nicht vorstellbar. Große Fortschritte im Bereich Conflict-Free Replicated Data Types (CRDTs) machen dies jedoch mittlerweile möglich. Allerdings gibt es dabei noch merklichen Aufholbedarf, was praktikable Sicherheitskonzepte angeht. Dies ist nicht zuletzt der Tatsache geschuldet, dass die Technologie selbst erst vor kurzem einen Entwicklungsstand erreicht hat, der einen Produktiveinsatz ermöglicht. Entsprechend ist zu erwarten, dass Sicherheitsfragen in den kommenden Jahren sowohl in der Forschung als auch in der Praxis ein größerer Stellenwert beigemessen werden wird.

Referenzen

[1] M. Shapiro, N. Preguiça, C. Baquero und M. Zawirski, „Conflict-Free Replicated Data Types," in *Stabilization, Safety, and Security of Distributed Systems*, Springer, 2011, pp. 386-400.

[2] Sylvia Ratnasamy et al, „A Scalable Content-addressable Network," in *Proceedings of the 2001 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, San Diego, ACM, 2001, pp. 161-172.

[3] R. C. Merkle, „A Digital Signature Based on a Conventional Encryption Function," in *Advances in Cryptology — CRYPTO '87*, Springer, 1978, pp. 369-378.

[4] B. Prünster, „Verteilter Datenspeicher in Heterogenen Umgebungen," 11 07 2018. [Online]. Available: https://technology.a-sit.at/verteilter-datenspeicher-in-heterogenen-umgebungen/. [Zugriff am 31 10 2018].

[5] B. Cohen, „The BitTorrent Protocol Specification," 11 10 2013. [Online]. Available: http://www.bittorrent.org/beps/bep_0003.html. [Zugriff am 24 04 2018].

[6] S. Nakamoto, „Bitcoin: A Peer-to-Peer Electronic Cash System," 2008. [Online]. Available: https://bitcoin.org/bitcoin.pdf. [Zugriff am 31 10 2018].

[7] C. Ellis und S. Gibbs, „Concurrency Control in Groupware Systems," in *ACM SIGMOD International Conference on Management of Data*, ACM, 1989, pp. 399-407.

[8] D. Li und R. Li, „An approach to ensuring consistency in peer-to-peer real-time groupeditors," *Computer Supported Cooperative Work (CSCW),* Bd. 17, Nr. 5-6, pp. 553-611, 2008.

[9] M. Kleppmann, „Automerge: Making servers optional for real-time collaboration," https://www.youtube.com/watch?v=PHz17gwiOc8, 2018.