

Real-Time Collaborative Editing of Documents

A Comparison of Two Approaches

DISSERTATION

zur Erlangung des akademischen Grades

doctor rerum naturalium

(Dr. rer. nat.)

im Fach Informatik

eingereicht an der

Mathematisch-Naturwissenschaftlichen Fakultät
der Humboldt-Universität zu Berlin

von

M. Sc. Vorname Name

Präsidentin der Humboldt-Universität zu Berlin:

Prof. Dr. Vorname Name

Dekan der Mathematisch-Naturwissenschaftlichen Fakultät:

Prof. Dr. Vorname Name

Gutachter/innen:

1. Prof. Dr. Vorname Name

2. Prof. Dr. Vorname Name

Tag der mündlichen Prüfung: 15. Dezember 2026

Zusammenfassung

Diese Arbeit vergleicht zwei Ansätze zur Synchronisierung eines Dokuments, das mehrere Personen gleichzeitig bearbeiten, die operationale Transformation und konfliktfreie replizierte Datentypen, und bewertet beide an einem Prototyp eines Web-Editors.

Schlagwörter Synchronisierung, Echtzeit, Web-Editor

Abstract

This thesis compares two approaches to synchronising a document that several people edit at the same time, operational transformation and conflict-free replicated data types, and evaluates both on a prototype of a web editor. The abstract should summarise the problem, the method, the results and the conclusions of the work, and be readable on its own.

Keywords Synchronisation, real time, web editor

Inhaltsverzeichnis

Zusammenfassung	i
Abstract	iii
Einleitung	1
1. Kollaborative Bearbeitung von Dokumenten	3
1.1. Anforderungen an das System	3
2. Ansätze zur Synchronisation	5
2.1. Operationale Transformation	5
2.1.1. Transformationsfunktion	5
2.2. Konfliktfreie replizierte Datentypen	5
3. Entwurf und Implementierung	7
3.1. Speicherung der Historie	7
4. Evaluation	9
4.1. Diskussion der Ergebnisse	10
Fazit	11
A. Supplementary Material	13
Selbständigkeitserklärung	17

Abbildungsverzeichnis

1.1. Typischer Ablauf des gemeinsamen Schreibens eines Dokuments	3
4.1. Anzahl der Änderungen nach Wochentag und Uhrzeit in den untersuchten Sitzungen	9

Tabellenverzeichnis

4.1. Mittlere Latenz und Größe der gespeicherten Daten nach Anzahl der Benutzer 9

Einleitung

Textdokumente entstehen heute nur noch selten in den Händen eines einzigen Autors. Studierende schreiben Seminararbeiten zu zweit, Entwickler pflegen gemeinsam technische Dokumentation, und Forschungsteams stellen Artikel in letzter Minute von mehreren Orten aus gleichzeitig fertig. Damit ein solches Schreiben funktioniert, muss der Editor sicherstellen, dass alle Beteiligten denselben Text sehen und keine Änderung verloren geht.

Ziel dieser Arbeit ist es, die zwei wichtigsten Ansätze zur Synchronisation von Dokumenten bei der kollaborativen Echtzeitbearbeitung, die operationale Transformation und konfliktfreie replizierte Datentypen, zu vergleichen und an einem Prototyp eines Web-Editors zu überprüfen. Das erste Kapitel legt die Anforderungen an ein solches System fest, das zweite beschreibt beide Ansätze, das dritte den Entwurf des Prototyps und das vierte die Messergebnisse.

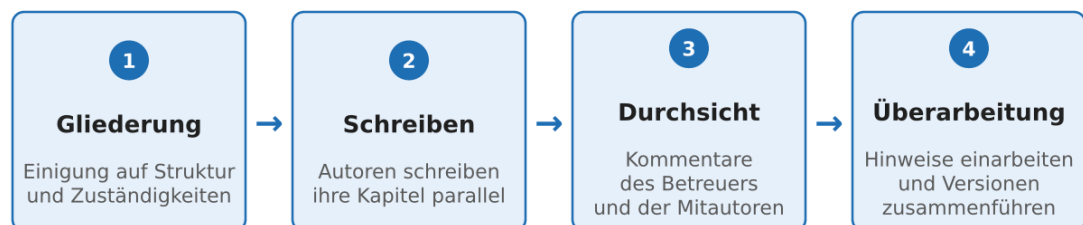
1. Kollaborative Bearbeitung von Dokumenten

Unter kollaborativer Bearbeitung verstehen wir eine Situation, in der mehrere Benutzer dasselbe Dokument bearbeiten und ihre Änderungen sich gegenseitig ohne manuelles Zusammenführen übertragen. Je nachdem, wie schnell sich die Änderungen verbreiten, unterscheiden wir drei grundlegende Arbeitsweisen:

- **asynchrone Bearbeitung**, bei der die Autoren ganze Versionen des Dokuments austauschen und erst nachträglich zusammenführen,
- **Bearbeitung mit Sperren**, bei der einen bestimmten Teil des Dokuments jeweils nur ein Autor bearbeiten darf,
- **Echtzeitbearbeitung**, bei der jede Änderung sofort bei den übrigen Benutzern sichtbar wird.

Diese Arbeit befasst sich ausschließlich mit der letzten Arbeitsweise. Sie ist für die Benutzer am bequemsten, stellt aber die höchsten Anforderungen an den Systementwurf, da Änderungen gleichzeitig an vielen Stellen entstehen und in unterschiedlicher Reihenfolge eintreffen.

Die Echtzeitbearbeitung verändert dabei nicht nur die Technik, sondern auch den Ablauf der gesamten Zusammenarbeit. Den typischen Ablauf des gemeinsamen Schreibens, wie ihn die befragten Benutzer beschrieben haben, zeigt Abbildung 1.1.



die Schritte 2 bis 4 wiederholen sich, solange die Durchsicht wesentliche Anmerkungen ergibt

Abbildung 1.1.: Typischer Ablauf des gemeinsamen Schreibens eines Dokuments

1.1. Anforderungen an das System

Aus Interviews mit Benutzern bestehender Editoren und aus der Fachliteratur haben wir vier Anforderungen abgeleitet, die ein System für die kollaborative Bearbeitung erfüllen muss:

1. Konvergenz, d. h. nach der Zustellung aller Änderungen haben alle Kopien des Dokuments denselben Inhalt,

2. Wahrung der Absicht, d. h. das Ergebnis einer Änderung entspricht dem, was der Autor im Moment der Änderung gesehen hat,
3. Reaktionsfähigkeit, d. h. eine lokale Änderung wirkt sich sofort aus, ohne auf den Server zu warten,
4. Robustheit, d. h. ein kurzzeitiger Verbindungsausfall führt nicht zum Verlust des begonnenen Textes.

Die ersten beiden Anforderungen betreffen die Korrektheit, die übrigen beiden die Benutzbarkeit. In der Praxis zeigt sich, dass gerade die Kombination aus sofortiger Reaktion und garantierter Konvergenz der schwierigste Teil des gesamten Entwurfs ist.

2. Ansätze zur Synchronisation

Systeme für die kollaborative Echtzeitbearbeitung führen jede Änderung zunächst lokal aus und verteilen sie erst danach an die übrigen Beteiligten. Sie unterscheiden sich vor allem darin, wie sie mit der Situation umgehen, in der zwei Änderungen gleichzeitig entstehen und jeder Beteiligte sie in einer anderen Reihenfolge erhält.

2.1. Operationale Transformation

Die operationale Transformation geht auf die Arbeit von Ellis und Gibbs [1] zurück, die als Erste beschrieben, wie eine nebenläufige Operation so angepasst werden kann, dass sie auch nach einer anderen Operation ausgeführt werden kann, von der ihr Autor nichts wusste. Jede Änderung wird als Einfüge- oder Löschoption an einer bestimmten Position im Text erfasst.

Fügt beispielsweise ein Benutzer ein Wort am Anfang eines Absatzes ein und löscht ein anderer gleichzeitig ein Wort an dessen Ende, muss die Position der Löschung um die Länge des eingefügten Wortes verschoben werden. Diese Verschiebung berechnet die Transformationsfunktion.

2.1.1. Transformationsfunktion

Die Transformationsfunktion erhält ein Paar nebenläufiger Operationen und gibt deren angepasste Versionen zurück. Damit alle Kopien des Dokuments konvergieren, muss die Funktion die sogenannten Transformationseigenschaften erfüllen, deren Einhaltung sich für allgemeine Operationen nur schwer beweisen lässt. Die meisten eingesetzten Systeme verlassen sich daher auf einen zentralen Server, der eine einzige Reihenfolge der Operationen festlegt.

2.2. Konfliktfreie replizierte Datentypen

Den zweiten Ansatz bilden konfliktfreie replizierte Datentypen, zusammenfassend mit der Abkürzung CRDT bezeichnet [2]. Statt Operationen nachträglich anzupassen, wählen sie eine Darstellung des Dokuments, in der nebenläufige Operationen kommutieren und ihre Reihenfolge keinen Einfluss auf das Ergebnis hat.

Ein replizierter Datentyp ist konfliktfrei, wenn zwei beliebige Replikate, die dieselbe Menge von Aktualisierungen erhalten haben, unabhängig von der Reihenfolge ihrer Zustellung denselben Zustand aufweisen.

Jedes Zeichen des Dokuments erhält einen eindeutigen Bezeichner, der während der gesamten Lebensdauer des Zeichens unverändert bleibt. Die Einfügeposition wird dann nicht durch eine Zahl, sondern durch einen Verweis auf den Bezeichner des benachbarten Zeichens bestimmt, sodass nebenläufige Änderungen an anderen Stellen des Textes sie nicht verschieben können.

3. Entwurf und Implementierung

Der Prototyp des Editors besteht aus einem Web-Client, einem Synchronisationsserver und einer Datenbank, in der der Server fortlaufend den Zustand der Dokumente speichert. Client und Server sind so geschrieben, dass sich die Synchronisationsschicht austauschen lässt und beide beschriebenen Ansätze auf derselben Benutzeroberfläche verglichen werden können.

Der Client hält ein lokales Replikat des Dokuments vor und überführt jede Änderung des Benutzers in eine Operation auf dem replizierten Datentyp. Die Operationen werden stapelweise an den Server gesendet, damit das Netz bei schnellem Tippen nicht überlastet wird. Die vereinfachte Verarbeitung eines eingehenden Stapels auf dem Server zeigt das folgende Listing:

```
def verarbeite_stapel(dokument, stapel):
    for operation in stapel.operationen:
        if operation.id in dokument.angewendet:
            continue
        dokument.anwenden(operation)
        dokument.angewendet.add(operation.id)
    an_andere_verteilen(dokument.id, stapel)
```

Da die Operationen kommutieren, muss der Server die Stapel weder ordnen noch transformieren. Es genügt, wenn er sicherstellt, dass keine Operation zweimal ausgeführt wird und jeder Stapel alle verbundenen Clients erreicht.

3.1. Speicherung der Historie

Der Server speichert nicht nur den resultierenden Text, sondern das gesamte Operationsprotokoll. Aus dem Protokoll lässt sich jede frühere Version des Dokuments wiederherstellen und der Replikatzustand eines Clients, der längere Zeit getrennt war, neu aufbauen. Dieser Ansatz entspricht einer Architektur auf Grundlage eines unveränderlichen Ereignisprotokolls, wie sie Kleppmann beschreibt [3]. Damit das Protokoll nicht unbegrenzt wächst, verdichtet der Server es in regelmäßigen Abständen zu Snapshots.

4. Evaluation

Wir haben den Prototyp mit einer Referenzimplementierung der operationalen Transformation anhand einer Reihe von Aufzeichnungen realer Bearbeitungssitzungen verglichen. Für jede Sitzung haben wir die Latenz, also die Zeit zwischen einer Änderung beim Autor und ihrer Anzeige bei den übrigen Beteiligten, erfasst und die mittlere Latenz der Sitzung nach (4.1) berechnet:

$$\bar{t} = \frac{1}{n} \sum_{i=1}^n \left(t_i^{\text{Zustellung}} - t_i^{\text{Änderung}} \right) \quad (4.1)$$

wobei n die Anzahl der Operationen in der Sitzung ist. Die gemessenen Werte für unterschiedlich viele gleichzeitig arbeitende Benutzer fasst Tabelle 4.1 zusammen.

Tabelle 4.1.: Mittlere Latenz und Größe der gespeicherten Daten nach Anzahl der Benutzer

Benutzer	Latenz OT (ms)	Latenz CRDT (ms)	Daten (kB)
2	38	35	12
5	52	41	19
10	87	48	31
20	164	63	58

Mit steigender Anzahl der Benutzer wächst die Latenz der operationalen Transformation deutlich schneller, da der Server jede Operation gegen alle nebenläufigen Operationen transformieren muss. Der Preis für die niedrigere Latenz von CRDT ist ein größeres Volumen gespeicherter Daten, da jedes Zeichen seinen eigenen Bezeichner trägt. Die zeitliche Verteilung der Änderungen, auf der der Lasttest beruht, zeigt Abbildung 4.1.

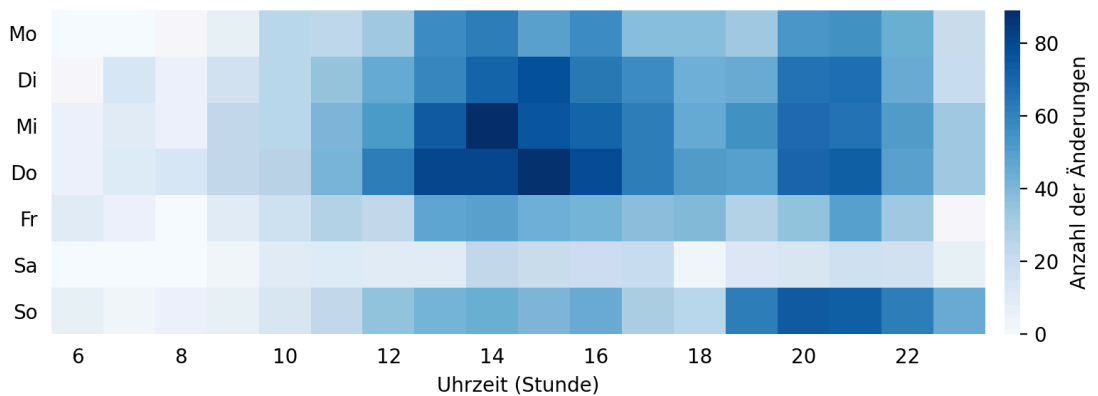


Abbildung 4.1.: Anzahl der Änderungen nach Wochentag und Uhrzeit in den untersuchten Sitzungen

4.1. Diskussion der Ergebnisse

Die Ergebnisse bestätigen, dass beide Ansätze für kleinere Teams gleichwertig sind und eher die Einfachheit der Implementierung den Ausschlag gibt. Bei einer größeren Anzahl gleichzeitiger Benutzer hat CRDT einen klaren Vorteil sowohl bei der Latenz als auch bei der Robustheit gegenüber Verbindungsausfällen. Ergänzende Interviews mit Benutzern, die nach den Grundsätzen der qualitativen Forschung geführt wurden [4], zeigten, dass die Benutzer einen Latenzunterschied unter hundert Millisekunden nicht wahrnehmen, den Verlust des begonnenen Textes nach einem Verbindungsausfall hingegen als gravierendes Problem ansehen.

Fazit

Die Arbeit hat die operationale Transformation und konfliktfreie replizierte Datentypen als zwei Ansätze zur Synchronisation von Dokumenten bei der kollaborativen Echtzeitbearbeitung verglichen. Am Prototyp eines Web-Editors haben wir gezeigt, dass CRDT einen einfacheren Server ohne zentrale Ordnung der Operationen ermöglichen und bei einer größeren Anzahl von Benutzern eine niedrigere Latenz bieten.

Der wichtigste Nachteil von CRDT bleibt das größere Volumen der gespeicherten Daten. Der nächste Schritt wird daher sein, zu überprüfen, wie sich die regelmäßige Verdichtung des Protokolls zu Snapshots auf die Datengröße auswirkt und wie sich der Prototyp im Langzeitbetrieb mit Zehntausenden von Dokumenten verhält.

A. Supplementary Material

An appendix holds material that supports the text but is not needed to follow it, such as the questionnaire of a survey, measurement data or a longer listing.

Literaturverzeichnis

- [1] C. A. Ellis und S. J. Gibbs, „Concurrency control in groupware systems“, in *Proceedings of the 1989 ACM SIGMOD International Conference on Management of Data*, New York: ACM, 1989, S. 399–407. doi: 10.1145/67544.66963.
- [2] M. Shapiro, N. Preguiça, C. Baquero, und M. Zawirski, „Conflict-free replicated data types“, in *Stabilization, Safety, and Security of Distributed Systems*, Berlin: Springer, 2011, S. 386–400. doi: 10.1007/978-3-642-24550-3_29.
- [3] M. Kleppmann, *Designing data-intensive applications*. Sebastopol: O’Reilly Media, 2017.
- [4] J. Hendl, *Kvalitativní výzkum: základní metody a aplikace*. Praha: Portál, 2005.

Selbständigkeitserklärung

Ich erkläre, dass ich die vorliegende Arbeit selbständig und nur unter Verwendung der angegebenen Literatur und Hilfsmittel angefertigt habe.

Berlin, den 1. Oktober 2026

M. Sc. Vorname Name