

Minimax-optimalisatie voor het modelleren van prestatieruimten

Deze ondertitel mag weggelaten worden

Bert Bibber

Bachelorproef

Bachelor of Science in de industriële wetenschappen: elektromechanica

Promotoren

prof. dr. ir. P. Kumulus, UAntwerpen

prof. dr. ir. E. Snuffel, UAntwerpen

Copromotor

ing. P. Pienter, Cosys, UAntwerpen

Externe Promotor

prof. dr. H. Warwinkel, TNT-Bang, N.V.



Universiteit Antwerpen

**Faculteit Toegepaste
Ingenieurswetenschappen**

Disclaimer Bachelorproef

Dit document is een examendocument dat niet werd gecorrigeerd voor eventueel vastgestelde fouten. Zonder voorafgaande schriftelijke toestemming van zowel de promotor(en) als de auteur(s) is overnemen, kopiëren, gebruiken of realiseren van deze uitgave of gedeelten ervan verboden. Voor aanvragen tot of informatie i.v.m. het overnemen en/of gebruik en/of realisatie van gedeelten uit deze publicatie, wendt u tot de universiteit waaraan de auteur is ingeschreven.

Voorafgaande schriftelijke toestemming van de promotor(en) is eveneens vereist voor het aanwenden van de in dit afstudeerwerk beschreven (originele) methoden, producten, schakelingen en programma's voor industrieel of commercieel nut en voor de inzending van deze publicatie ter deelname aan wetenschappelijke prijzen of wedstrijden.

Dit document is in overeenstemming met het facultair reglement gerelateerd aan dit examendocument en de gedragscode. De tekst werd nagekeken door de promotor en de begeleider.

Inhoudsopgave

Inleiding	3
1 Gezamenlijke bewerking van documenten	5
1.1 Eisen aan het systeem	5
2 Benaderingen voor synchronisatie	7
2.1 Operationele transformatie	7
2.1.1 Transformatiefunctie	7
2.2 Conflictvrije gerepliceerde datatypen	7
3 Ontwerp en implementatie	9
3.1 Opslag van de geschiedenis	9
4 Evaluatie	11
4.1 Bespreking van de resultaten	12
Conclusie	13

Samenvatting
Minimax-optimalisatie voor het modelleren van
prestatieruimten

Deze ondertitel mag weggelaten worden

Bert Bibber

De samenvatting geeft op één pagina een betekenen. Ze staat in twee kolommen onder
overzicht van de hele bachelorproef: de onder- de titel, de ondertitel en de auteur.
zoeksvraag, de aanpak, de resultaten en wat ze

Inleiding

Tekstdocumenten worden tegenwoordig nog maar zelden door één enkele auteur geschreven. Studenten schrijven werkstukken in tweetallen, ontwikkelaars onderhouden samen technische documentatie en onderzoeksteams leggen op het laatste moment vanaf meerdere locaties tegelijk de laatste hand aan artikelen. Om dergelijk schrijven mogelijk te maken, moet de editor waarborgen dat alle deelnemers dezelfde tekst zien en dat geen enkele wijziging verloren gaat.

Het doel van dit werk is twee hoofdbenaderingen voor de synchronisatie van documenten bij gezamenlijke bewerking in realtime te vergelijken, operationele transformatie en conflictvrije gerepliceerde datatypen, en deze te toetsen aan de hand van een prototype van een webeditor. Het eerste hoofdstuk bakt de eisen aan een dergelijk systeem af, het tweede beschrijft beide benaderingen, het derde het ontwerp van het prototype en het vierde de meetresultaten.

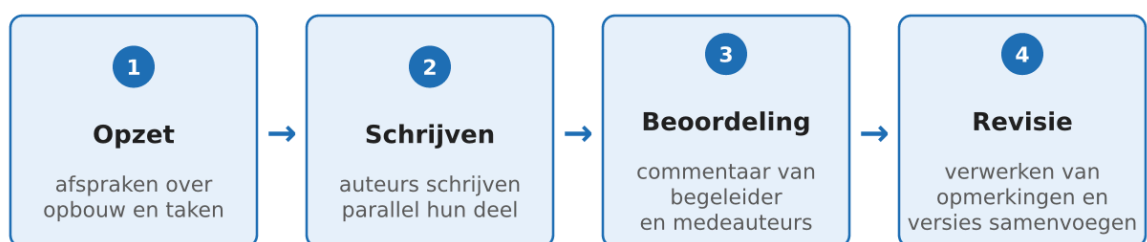
1 Gezamenlijke bewerking van documenten

Onder gezamenlijke bewerking verstaan we de situatie waarin meerdere gebruikers hetzelfde document bewerken en hun wijzigingen bij elkaar doorwerken zonder handmatig samenvoegen. Naargelang de snelheid waarmee wijzigingen zich verspreiden, onderscheiden we drie basale werkwijzen:

- **asynchrone bewerking**, waarbij auteurs volledige versies van het document aan elkaar doorgeven en deze pas achteraf samenvoegen,
- **bewerking met vergrendeling**, waarbij op een gegeven moment slechts één auteur een bepaald deel van het document mag bewerken,
- **bewerking in realtime**, waarbij elke wijziging onmiddellijk zichtbaar wordt voor de andere gebruikers.

Dit werk richt zich uitsluitend op de laatstgenoemde werkwijze. Deze is voor de gebruiker het prettigst, maar stelt de hoogste eisen aan het systeemontwerp, omdat wijzigingen gelijktijdig op veel plaatsen ontstaan en in verschillende volgorde binnenkomen.

Bewerking in realtime verandert daarbij niet alleen de techniek, maar ook het verloop van de hele samenwerking. Het typische verloop van gezamenlijk schrijven, zoals de geïnterviewde gebruikers het beschreven, is weergegeven in Figuur 1.1.



stappen 2 t/m 4 worden herhaald zolang de beoordeling wezenlijke opmerkingen oplevert

Figuur 1.1: Typisch verloop van het gezamenlijk schrijven van een document

1.1 Eisen aan het systeem

Uit interviews met gebruikers van bestaande editors en uit de vakliteratuur hebben we vier eisen afgeleid waaraan een systeem voor gezamenlijke bewerking moet voldoen:

1. convergentie, dat wil zeggen dat na aflevering van alle wijzigingen alle kopieën van het document dezelfde inhoud hebben,

2. behoud van intentie, dat wil zeggen dat het resultaat van een bewerking overeenkomt met wat de auteur op het moment van bewerken zag,
3. responsiviteit, dat wil zeggen dat een lokale bewerking onmiddellijk zichtbaar is zonder op de server te wachten,
4. robuustheid, dat wil zeggen dat een kortstondige verbindingsonderbreking niet leidt tot verlies van tekst waaraan wordt gewerkt.

De eerste twee eisen betreffen correctheid, de overige twee bruikbaarheid. In de praktijk blijkt juist de combinatie van onmiddellijke respons en gegarandeerde convergentie het moeilijkste deel van het hele ontwerp.

2 Benaderingen voor synchronisatie

Systemen voor gezamenlijke bewerking in realtime voeren elke wijziging eerst lokaal uit en verspreiden deze pas daarna onder de andere deelnemers. Ze verschillen vooral in de manier waarop ze omgaan met de situatie waarin twee wijzigingen gelijktijdig ontstaan en elke deelnemer ze in een andere volgorde ontvangt.

2.1 Operationele transformatie

Operationele transformatie gaat terug op het werk van Ellis en Gibbs [1], die als eersten beschreven hoe een gelijktijdige operatie zo kan worden aangepast dat zij ook kan worden uitgevoerd na een andere operatie waarvan haar auteur niet op de hoogte was. Elke bewerking wordt vastgelegd als een invoeg- of verwijderoperatie op een bepaalde positie in de tekst.

Als bijvoorbeeld de ene gebruiker een woord aan het begin van een alinea invoegt en de andere tegelijkertijd een woord aan het einde ervan verwijdert, moet de positie van de verwijdering worden verschoven met de lengte van het ingevoegde woord. Deze verschuiving wordt berekend door de transformatiefunctie.

2.1.1 Transformatiefunctie

De transformatiefunctie krijgt een paar gelijktijdige operaties en geeft daarvan aangepaste versies terug. Om alle kopieën van het document te laten convergeren, moet de functie voldoen aan de zogeheten transformatie-eigenschappen, waarvan de geldigheid voor algemene operaties moeilijk te bewijzen is. De meeste systemen in de praktijk vertrouwen daarom op een centrale server die één enkele volgorde van operaties vastlegt.

2.2 Conflictvrije gerepliceerde datatypen

De tweede benadering wordt gevormd door conflictvrije gerepliceerde datatypen, gezamenlijk aangeduid met de afkorting CRDT [2]. In plaats van operaties achteraf aan te passen, kiezen zij een zodanige representatie van het document dat gelijktijdige operaties commuteren en hun volgorde geen invloed heeft op het resultaat.

Een gerepliceerd datatype is conflictvrij als twee willekeurige replica's die dezelfde verzameling updates hebben ontvangen dezelfde toestand hebben, ongeacht de volgorde waarin die zijn afgeleverd.

Elk teken van het document krijgt een unieke identicator die gedurende het hele bestaan van het teken gelijk blijft. De invoegpositie wordt dan niet bepaald door een getal, maar door een

verwijzing naar de identifier van het naburige teken, zodat gelijktijdige bewerkingen elders in de tekst haar niet kunnen verschuiven.

3 Ontwerp en implementatie

Het prototype van de editor bestaat uit een webclient, een synchronisatieserver en een database waarin de server doorlopend de toestand van de documenten opslaat. Zowel de client als de server is zo geschreven dat de synchronisatielaag kan worden vervangen en beide beschreven benaderingen op dezelfde gebruikersinterface kunnen worden vergeleken.

De client houdt een lokale replica van het document bij en zet elke bewerking van de gebruiker om in een operatie op het gerepliceerde datatype. De operaties worden in batches naar de server gestuurd, zodat het netwerk bij snel typen niet overbelast raakt. De vereenvoudigde verwerking van een binnenkomende batch op de server is weergegeven in het volgende fragment:

```
def verwerk_batch(document, batch):
    for operatie in batch.operaties:
        if operatie.id in document.toegepast:
            continue
        document.pas_toe(operatie)
        document.toegepast.add(operatie.id)
        stuur_naar_anderen(document.id, batch)
```

Omdat de operaties commuteren, hoeft de server de batches niet te ordenen of te transformeren. Het volstaat dat hij ervoor zorgt dat geen enkele operatie twee keer wordt uitgevoerd en dat elke batch alle verbonden clients bereikt.

3.1 Opslag van de geschiedenis

De server slaat niet alleen de uiteindelijke tekst op, maar het volledige operatielogboek. Uit het logboek kan elke eerdere versie van het document worden hersteld en kan de toestand van de replica van een client die langere tijd offline was opnieuw worden opgebouwd. Deze aanpak komt overeen met een architectuur op basis van een onveranderlijk gebeurtenissenlogboek, zoals beschreven door Kleppmann [3]. Om te voorkomen dat het logboek onbeperkt groeit, comprimeert de server het met regelmatige tussenpozen tot snapshots.

4 Evaluatie

We hebben het prototype vergeleken met een referentie-implementatie van operationele transformatie op een verzameling registraties van echte bewerkingssessies. Voor elke sessie maten we de latentie, dat wil zeggen de tijd tussen een bewerking bij de auteur en de weergave ervan bij de andere deelnemers, en de gemiddelde latentie van de sessie berekenden we volgens (4.1):

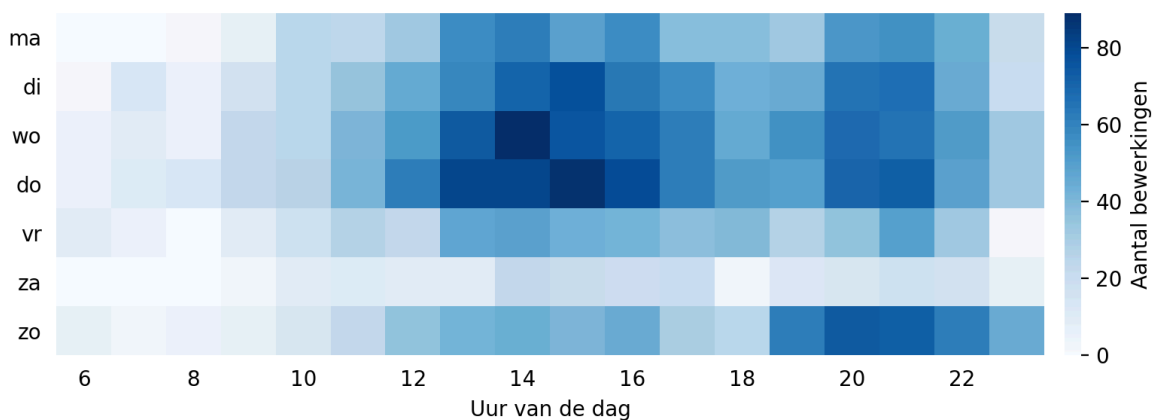
$$\bar{t} = \frac{1}{n} \sum_{i=1}^n (t_i^{\text{aflevering}} - t_i^{\text{bewerking}}) \quad (4.1)$$

waarbij n het aantal operaties in de sessie is. De gemeten waarden voor verschillende aantallen gelijktijdig werkende gebruikers zijn samengevat in Tabel 4.1.

Tabel 4.1: Gemiddelde latentie en omvang van de opgeslagen gegevens naar aantal gebruikers

Gebruikers	Latentie OT (ms)	Latentie CRDT (ms)	Data (kB)
2	38	35	12
5	52	41	19
10	87	48	31
20	164	63	58

Naarmate het aantal gebruikers toeneemt, stijgt de latentie van operationele transformatie aanzienlijk sneller, omdat de server elke operatie moet transformeren ten opzichte van alle gelijktijdige operaties. De prijs voor de lagere latentie van CRDT is een grotere hoeveelheid opgeslagen gegevens, aangezien elk teken zijn eigen identicator meedraagt. De verdeling van de bewerkingen in de tijd, waarop de belastingtest is gebaseerd, is weergegeven in Figuur 4.1.



Figuur 4.1: Aantal bewerkingen per dag van de week en uur in de onderzochte sessies

4.1 Bespreking van de resultaten

De resultaten bevestigen dat beide benaderingen voor kleinere teams gelijkwaardig zijn en dat veeleer de eenvoud van de implementatie de doorslag geeft. Bij een groter aantal gelijktijdige gebruikers heeft CRDT een duidelijk voordeel, zowel in latentie als in robuustheid tegen verbindingsonderbrekingen. Aanvullende interviews met gebruikers, afgenomen volgens de principes van kwalitatief onderzoek [4], lieten zien dat gebruikers een verschil in latentie van minder dan honderd milliseconden niet waarnemen, maar het verlies van tekst waaraan werd gewerkt na een verbindingsonderbreking als een ernstig probleem beschouwen.

Conclusie

Dit werk heeft operationele transformatie en conflictvrije gerepliceerde datatypen vergeleken als twee benaderingen voor de synchronisatie van documenten bij gezamenlijke bewerking in realtime. Met een prototype van een webeditor hebben we aangetoond dat CRDT's een eenvoudigere server mogelijk maken zonder centrale ordening van operaties en bij grotere aantallen gebruikers een lagere latentie bieden.

Het belangrijkste nadeel van CRDT's blijft de grotere hoeveelheid opgeslagen gegevens. Een volgende stap is daarom na te gaan hoe de omvang van de gegevens de periodieke compressie van het logboek tot snapshots beïnvloedt en hoe het prototype zich gedraagt bij langdurig gebruik met tienduizenden documenten.

Literatuurlijst

- [1] C. A. Ellis en 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, pp. 399-407. doi: 10.1145/67544.66963.
- [2] M. Shapiro, N. Preguiça, C. Baquero, en M. Zawirski, “Conflict-free replicated data types”, in *Stabilization, Safety, and Security of Distributed Systems*, Berlin: Springer, 2011, pp. 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.