



POLITECHNIKA POZNAŃSKA

WYDZIAŁ INFORMATYKI I TELEKOMUNIKACJI
Instytut Informatyki

Praca dyplomowa licencjacka

TYTUŁ PRACY DYPLOMOWEJ

Imię i nazwisko pierwszego autora, 111111

Imię i nazwisko drugiego autora, 222222

Promotor

prof. dr hab. inż. Imię Nazwisko

POZNAŃ 2026

Tutaj będzie karta pracy dyplomowej;
oryginał wstawiamy do wersji dla archiwum PP, w pozostałych kopiach wstawiamy ksero.

Streszczenie

Tutaj napisz streszczenie pracy.

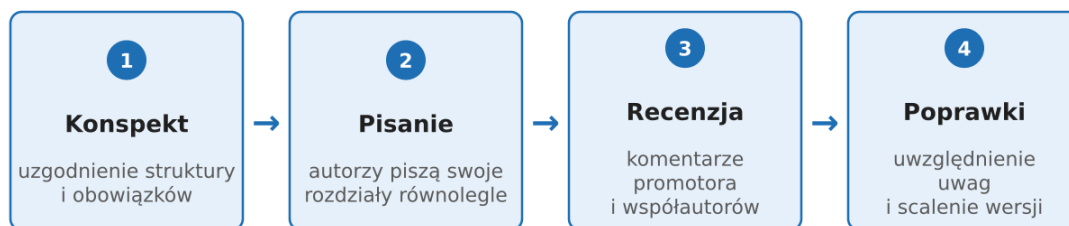
Spis treści

Wstęp	1
1 Wspólna edycja dokumentów	2
1.1 Wymagania wobec systemu	2
2 Podejścia do synchronizacji	4
2.1 Transformacja operacyjna	4
2.1.1 Funkcja transformacji	4
2.2 Bezkonfliktowe replikowane typy danych	4
3 Projekt i implementacja	5
3.1 Przechowywanie historii	5
4 Ewaluacja	6
4.1 Dyskusja wyników	7
Zakończenie	7
Bibliografia	8

Wstęp

Dokumenty tekstowe tylko wyjątkowo powstają dziś w rękach jednego autora. Studenci piszą prace seminaryjne w parach, programiści wspólnie utrzymują dokumentację techniczną, a zespoły badawcze kończą artykuły w ostatniej chwili z kilku miejsc jednocześnie. Aby takie pisanie było możliwe, edytor musi zapewnić, że wszyscy uczestnicy widzą ten sam tekst i że żadna zmiana nie zostanie utracona.

Celem niniejszej pracy jest porównanie dwóch głównych podejść do synchronizacji dokumentów podczas wspólnej edycji w czasie rzeczywistym, transformacji operacyjnej i bezkonfliktowych replikowanych typów danych, oraz ich weryfikacja na prototypie edytora internetowego. Pierwszy rozdział określa wymagania wobec takiego systemu, drugi opisuje oba podejścia, trzeci projekt prototypu, a czwarty wyniki pomiarów.



kroki od 2 do 4 powtarzają się, dopóki recenzja przynosi istotne uwagi

Rysunek 1.1. Typowy przebieg wspólnego pisania dokumentu

Rozdział 1

Wspólna edycja dokumentów

Przez wspólną edycję rozumiemy sytuację, w której wielu użytkowników modyfikuje ten sam dokument, a ich zmiany wzajemnie się propagują bez ręcznego scalania. W zależności od tego, jak szybko rozchodzą się zmiany, wyróżniamy trzy podstawowe tryby pracy:

- **edycja asynchroniczna**, w której autorzy przekazują sobie całe wersje dokumentu i scalają je dopiero później;
- **edycja z blokadami**, w której w danej chwili określoną część dokumentu może modyfikować tylko jeden autor;
- **edycja w czasie rzeczywistym**, w której każda zmiana jest natychmiast widoczna dla pozostałych użytkowników.

Niniejsza praca dotyczy wyłącznie ostatniego trybu. Jest on dla użytkowników najwygodniejszy, ale stawia największe wymagania projektowi systemu, ponieważ zmiany powstają współbieżnie w wielu miejscach i docierają w różnej kolejności.

Edycja w czasie rzeczywistym zmienia przy tym nie tylko technikę, lecz także przebieg całej współpracy. Typowy proces wspólnego pisania, opisany przez ankietowanych użytkowników, przedstawia Rysunek 1.1.

1.1 Wymagania wobec systemu

Na podstawie wywiadów z użytkownikami istniejących edytorów oraz literatury przedmiotu sformułowaliśmy cztery wymagania, które musi spełniać system do wspólnej edycji:

1. **zbieżność**, czyli po dostarczeniu wszystkich zmian wszystkie kopie dokumentu mają tę samą zawartość;
2. **zachowanie intencji**, czyli wynik zmiany odpowiada temu, co autor widział w chwili jej wprowadzania;
3. **responsywność**, czyli lokalna zmiana jest widoczna natychmiast, bez oczekiwania na serwer;

4. odporność, czyli krótkotrwała przerwa w połączeniu nie prowadzi do utraty wprowadzanego tekstu.

Pierwsze dwa wymagania dotyczą poprawności, pozostałe dwa użyteczności. W praktyce okazuje się, że właśnie połączenie natychmiastowej responsywności i gwarantowanej zbieżności stanowi najtrudniejszą część całego projektu.

Rozdział 2

Podjęcia do synchronizacji

Systemy wspólnej edycji w czasie rzeczywistym każdą zmianę najpierw wykonują lokalnie, a dopiero potem rozsyłają ją pozostałym uczestnikom. Różnią się przede wszystkim tym, jak rozwiązują sytuację, w której dwie zmiany powstają współbieżnie, a każdy uczestnik otrzymuje je w innej kolejności.

2.1 Transformacja operacyjna

Transformacja operacyjna wywodzi się z pracy Ellisa i Gibbsa [1], którzy jako pierwsi opisali, jak zmodyfikować współbieżną operację tak, aby można ją było wykonać również po innej operacji, o której jej autor nie wiedział. Każda zmiana jest zapisywana jako operacja wstawienia lub usunięcia na określonej pozycji w tekście.

Jeśli na przykład jeden użytkownik wstawi słowo na początku akapitu, a drugi w tym samym czasie usunie słowo na jego końcu, pozycja usunięcia musi zostać przesunięta o długość wstawionego słowa. To przesunięcie oblicza funkcja transformacji.

2.1.1 Funkcja transformacji

Funkcja transformacji otrzymuje parę współbieżnych operacji i zwraca ich zmodyfikowane wersje. Aby wszystkie kopie dokumentu były zbieżne, funkcja musi spełniać tak zwane własności transformacji, których zachowanie jest trudne do udowodnienia dla operacji ogólnych. Większość wdrożonych systemów opiera się zatem na serwerze centralnym, który ustala jedną kolejność operacji.

2.2 Bezkonfliktowe replikowane typy danych

Drugie podejście stanowią bezkonfliktowe replikowane typy danych, określane łącznie skrótem CRDT [2]. Zamiast późniejszej modyfikacji operacji wybierają one taką reprezentację dokumentu, w której współbieżne operacje są przemienne, a ich kolejność nie wpływa na wynik.

„Replikowany typ danych jest bezkonfliktowy, jeżeli dowolne dwie repliki, które otrzymały ten sam zbiór aktualizacji, mają ten sam stan niezależnie od kolejności ich dostarczenia.”

Każdy znak dokumentu otrzymuje jednoznaczny identyfikator, który pozostaje niezmienny przez cały czas istnienia znaku. Pozycja wstawienia nie jest wówczas określana liczbą, lecz odwołaniem do identyfikatora sąsiedniego znaku, dzięki czemu współbieżne zmiany w innych miejscach tekstu nie mogą jej przesunąć.

Rozdział 3

Projekt i implementacja

Prototyp edytora składa się z klienta webowego, serwera synchronizacji i bazy danych, w której serwer na bieżąco zapisuje stan dokumentów. Klient i serwer zostały napisane tak, aby można było wymienić warstwę synchronizacji i porównać oba opisane podejścia przy tym samym interfejsie użytkownika.

Klient utrzymuje lokalną replikę dokumentu i każdą zmianę użytkownika przekształca w operację na replikowanym typie danych. Operacje są wysyłane do serwera w paczkach, aby przy szybkim pisaniu nie przeciążać sieci. Uprozczone przetwarzanie przychodzącej paczki na serwerze przedstawia poniższy listing:

```
def przetworz_paczke(dokument, paczka):
    for operacja in paczka.operacje:
        if operacja.id in dokument.zastosowane:
            continue
        dokument.zastosuj(operacja)
        dokument.zastosowane.add(operacja.id)
    rozeslij_pozostalym(dokument.id, paczka)
```

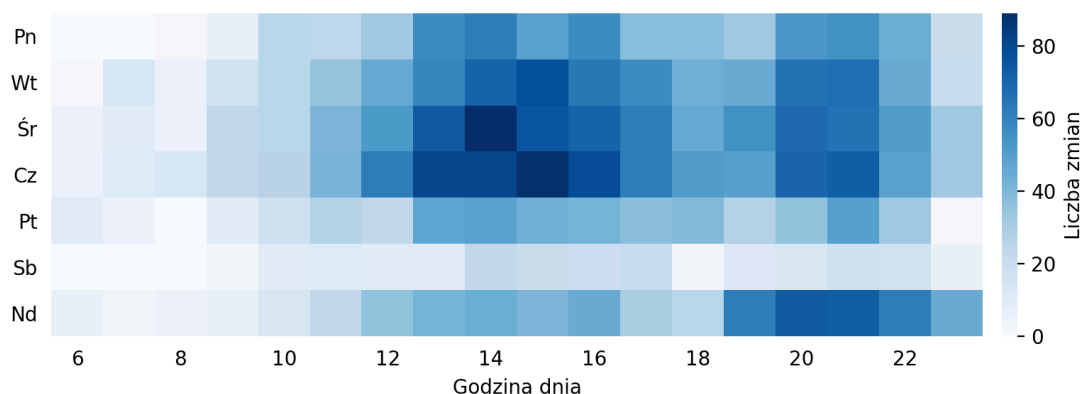
Ponieważ operacje są przemienne, serwer nie musi paczek porządkować ani transformować. Wystarczy, że zapewni, iż żadna operacja nie zostanie wykonana dwukrotnie i że każda paczka dotrze do wszystkich podłączonych klientów.

3.1 Przechowywanie historii

Serwer zapisuje nie tylko wynikowy tekst, lecz cały log operacji. Z logu można odtworzyć dowolną wcześniejszą wersję dokumentu i ponownie zbudować stan repliki klienta, który przez dłuższy czas był odłączony. Podejście to odpowiada architekturze opartej na niezmiennym zapisie zdarzeń, jak opisuje ją Kleppmann [3]. Aby log nie rósł bez ograniczeń, serwer w regularnych odstępach czasu kompaktuje go w migawki.

Tabela 4.1. Średnie opóźnienie i rozmiar zapisanych danych w zależności od liczby użytkowników

Użytkownicy	Opóźnienie OT (ms)	Opóźnienie CRDT (ms)	Dane (kB)
2	38	35	12
5	52	41	19
10	87	48	31
20	164	63	58



Rysunek 4.1. Liczba zmian według dnia tygodnia i godziny w badanych sesjach

Rozdział 4

Ewaluacja

Prototyp porównaliśmy z referencyjną implementacją transformacji operacyjnej na zbiorze zapisów rzeczywistych sesji edycyjnych. Dla każdej sesji mierzyliśmy opóźnienie, czyli czas między wprowadzeniem zmiany przez autora a jej wyświetleniem u pozostałych uczestników, a średnie opóźnienie sesji obliczyliśmy w sposób, który przedstawia (4.1):

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

gdzie n oznacza liczbę operacji w sesji. Zmierzone wartości dla różnej liczby równocześnie pracujących użytkowników zestawia Tabela 4.1.

Wraz ze wzrostem liczby użytkowników opóźnienie transformacji operacyjnej rośnie znacznie szybciej, ponieważ serwer musi transformować każdą operację względem wszystkich operacji współbieżnych. Ceną za niższe opóźnienie CRDT jest większa ilość zapisanych danych, gdyż każdy znak przechowuje swój identyfikator. Rozkład zmian w czasie, na którym opiera się test obciążeniowy, przedstawia Rysunek 4.1.

4.1 Dyskusja wyników

Wyniki potwierdzają, że dla mniejszych zespołów oba podejścia są równorzędne, a o wyborze decyduje raczej prostota implementacji. Przy większej liczbie równoczesnych użytkowników CRDT ma wyraźną przewagę zarówno pod względem opóźnienia, jak i odporności na przerwy w połączeniu. Uzupełniające wywiady z użytkownikami, przeprowadzone zgodnie z zasadami badań jakościowych [4], wykazały, że różnicy opóźnienia poniżej stu milisekund użytkownicy nie dostrzegają, natomiast utratę wprowadzanego tekstu po przerwie w połączeniu uważają za poważny problem.

Zakończenie

W pracy porównano transformację operacyjną i bezkonfliktowe replikowane typy danych jako dwa podejścia do synchronizacji dokumentów podczas wspólnej edycji w czasie rzeczywistym. Na prototypie edytora internetowego wykazaliśmy, że CRDT umożliwiają prostszy serwer bez centralnego porządkowania operacji, a przy większej liczbie użytkowników zapewniają niższe opóźnienie.

Główną wadą CRDT pozostaje większa ilość przechowywanych danych. Kolejnym krokiem będzie zatem sprawdzenie, jak rozmiar danych wpływa na regularną kompaktację logu do migawek oraz jak prototyp zachowuje się podczas długotrwałej eksploatacji z dziesiątkami tysięcy dokumentów.

Bibliografia

- [1] C. A. Ellis i S. J. Gibbs, „Concurrency control in groupware systems”, w *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, i M. Zawirski, „Conflict-free replicated data types”, w *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.



© 2026 Imię i nazwisko pierwszego autora, Imię i nazwisko drugiego autora

Instytut Informatyki, Wydział Informatyki i Telekomunikacji
Politechnika Poznańska

Skład w TommDocu według szablonu put-thesis