Zachodniopomorski
Uniwersytet Technologiczny
w Szczecinie

Katedra Inżynierii Oprogramowania

kierunek studiów: Informatyka

specjalność: Inżynieria Oprogramowania

Praca dyplomowa inżynierska

IMPLEMENTACJA APLIKACJI DO WSPÓŁTWORZENIA NOTATEK Z WYKORZYSTANIEM TECHNOLOGII PEER TO PEER

IMPLEMENTATION OF A COLLABORATIVE NOTE-TAKING APPLICATION USING PEER-TO-PEER TECHNOLOGY

Oskar Marcin Chybowski

nr albumu: **54941**

Opiekun:

dr inż. Mirosław Mościcki

Katedra Inżynierii Oprogramowania

Szczecin, 2026

Spis treści

Zakres teoretyczny

Systemy współdzielenia dokumentów w czasie rzeczywistym

Budując rozwiązania związane z równoczesnym tworzeniem i modyfikacją tekstu przez więcej niż jednego użytkownika, musimy rozważyć wyzwania napotykane w aktualizacji tworzonego dokumentu, gdzie każdy klient posiada lokalną kopię i nanosi na nie własne zmiany, ale też w międzyczasie musimy nanieść zmiany od pozostałych klientów. W takim systemie mówimy wtedy o zbieżności danych[1] - czyli zapewnieniu tego samego stanu między każdym klientem. W przypadku edycji tekstu skupię się na ewentualnej zbieżności, która uwzględnia posiadanie rozbieżnych kopii tego samego źródła danych u każdego z klientów przez pewien czas. Dopiero gdy zostanie zakończona edycja tekstu, zmiany zostają propagowane i nanoszone do pozostałych klientów. Finalnie każdy klient po czasie posiada identyczną kopię dokumentu. Ze strony doświadczeń użytkowania jest to skuteczna strategia ze względu na możliwość zapewnienia płynności interfejsu graficznego oraz z pomocą złożonych mechanizmów umożliwia rozwiązywanie konfliktów między kopiami.

Wspomniany model nie jest bez wad. Największym problemem jest istnienie konfliktów, których rozwiązanie klienci muszą ustalić za pomocą dodatkowych strategii. Najczęściej wykorzystywaną jest Last-Write-Wins (LWW). Rozstrzyga ona konflikty poprzez nanoszenie tylko tej zmiany, która jest uznawana jako ostatnia w kolejności zbioru konfliktujących operacji. Ustalanie kolejności nie jest jasno tutaj zdefiniowane. W systemach baz danych takich jak Cassandra[9] oraz SQL Server P2P[10] każdy zapis otrzymuje własny znacznik czasowy, na podstawie którego wybierany jest najmłodszy wpis i nim nadpisywane są zmiany w źródle danych. Zmiany ze starszymi znacznikami są porzucane. Zauważalną wadą LWW jest wysokie ryzyko utraty danych w czasie nanoszenia zmian, ponieważ wszystkie konfliktujące starsze zmiany nie są brane pod uwagę.

W przypadku tekstu jako typu danych, istnieje specjalny wariant ewentualnej zbieżności - silna ewentualna zbieżność. Ten model wykorzystuje specjalne struktury danych, które zapewniają bezkonfliktowe nanoszenie zmian, a ich skuteczność opiera się na matematycznych dowodach[2].

Istnieje również model silnej zbieżności, gdzie każdy z klientów musi mieć tą samą kopię danych przez cały czas pracy systemu. Ze względu opóźnienia i potrzebę mechanizmu blokady klientów przed wprowadzaniem zmian, ten model został porzucony w dzisiejszych systemach, ponieważ wspomniane problemy skutkowały gorszą użytecznością w porównaniu do systemów wykorzystujących

ewentualną zbieżność. Istnieją jednak prace, które wskazują na istnienie systemów opartych o model silnej zbieżności, które osiągają bardzo niskie czasy opóźnienia, co redukuje problem używalności takiego rozwiązania w rzeczywistych zastosowaniach.[11]

Algorytmy zapewniające silną ewentualną zbieżność

Podejście do rozwiązania problemu synchronizacji stanów między każdym klientem wymaga zaprojektowania algorytmu od podstaw skupionego na tym zagadnieniu. Przedstawię dwa najczęściej wykorzystywane algorytmy rozwiązujące opisany problem.

Operational Transformation (OT) polega na zamianie każdej wykonanej operacji na kodowalny obiekt, który może być propagowany i nanoszony na kopie w innych replikach. Większość znanych implementacji zakłada też, że istnieje scentralizowany serwer określający kolejność każdej operacji zgłaszanej przez wszystkich klientów i dystrybujący wspomniane zmiany do pozostałych replik. W przypadku wystąpienia konfliktu, operacje są sortowane przez serwer, a następnie każda z nich jest transformowana tak, by uwzględnić zmianę poprzedniej w kolejności operacji. Przykładowymi propozycjami algorytmów OT, których skuteczność nie została obalona dowodami, są: Jupiter[3], SOCT3/4[4] oraz TTF[5]. Jupiter oraz SOCT3/4 wymagają wcześniej wspomnianego scentralizowanego serwera. Google Docs początkowo korzystał z Operational Transformation[12]. Ze względu na złożność implementacji większość systemów nie korzysta dziś z tych algorytmów do synchronizacji danych, gdzie przykładem takiej decyzji jest Figma[13].

Następcą Operational Transformation są bezkonfliktowe replikowane typy danych (Conflict-free replicated data types - CRDT). Pozwalają one na nanoszenie zmian na własne kopie przez klientów, bez potrzeby uzgadniania tego z innymi klientami. Tak jak każdy algorytm ewentualnej zbieżności, pozwala na tymczasową rozbieżność między replikami, ale po otrzymaniu wszystkich zmian przez każdego klienta, dane zawsze pozostają zbieżne. Przykładowymi algorytmami są RGA[6], WOOT[7] oraz TreeDoc[8].

Spis literatury

Artykuły

[1] Werner Vogels. „Eventually consistent”. W: 52.1 (2009-01), s. 40–44.

[2] Victor B. F. Gomes, Martin Kleppmann, Dominic P. Mulligan i Alastair R. Beresford. „Verifying strong eventual consistency in distributed systems”. W: 1.OOPSLA (2017-10).

[3] David A. Nichols, Pavel Curtis, Michael Dixon i John Lamping. „High-latency, low-bandwidth windowing in the Jupiter collaboration system”. W: (1995), s. 111–120.

[4] Nicolas Vidot, Michelle Cart, Jean Ferrié i Maher Suleiman. „Copies convergence in a distributed real-time collaborative environment”. W: (2000), s. 171–180.

[5] Gérald Oster, Pascal Molli, Pascal Urso i Abdessamad Imine. „Tombstone Transformation Functions for Ensuring Consistency in Collaborative Editing Systems”. W: (2006), s. 1–10.

[6] Hyun-Gul Roh, Myeongjae Jeon, Jin-Soo Kim i Joonwon Lee. „Replicated abstract data types: Building blocks for collaborative applications”. W: 71.3 (2011), s. 354–368.

[7] Gérald Oster, Pascal Urso, Pascal Molli i Abdessamad Imine. „Data consistency for P2P collaborative editing”. W: (2006), s. 259–268.

[8] Nuno Preguica, Joan Manuel Marques, Marc Shapiro i Mihai Letia. „A Commutative Replicated Data Type for Cooperative Editing”. W: (2009), s. 395–403.

Źródła internetowe i inne

[9] Apache. *Data versioning - Apache Cassandra Documentation*. URL: `https://cassandra.apache.org/doc/latest/cassandra/architecture/dynamo.html#data-versioning`.

[10] Microsoft. *Peer To Peer Transactional Replication - Microsoft SQL Server documentation*. URL: `https://learn.microsoft.com/en-us/sql/relational-databases/replication/transactional/peer-to-peer-transactional-replication?view=sql-server-ver17`.

[11] Weixin Yu i Kaylee Xie. *A Paxos-Based Strongly Consistent Live Document Editor*. URL: `https://www.scs.stanford.edu/26wi-cs244c/proj/live_document_editor.pdf`.

[12] John Day-Richter. *What's Different About New Google Docs*. URL: `https://drive.googleblog.com/2010/09/whats-different-about-new-google-docs.html`.

[13] Evan Wallace. *How Figma's Multiplayer Technology Works*. URL: `https://www.figma.com/blog/how-figmas-multiplayer-technology-works/`.