



EÖTVÖS LORÁND TUDOMÁNYEGYETEM

INFORMATIKAI KAR

PROGRAMOZÁSELMÉLET ÉS SZOFTVERTECHNOLÓGIAI
TANSZÉK

Dolgozat címe

Témavezető:

Témavezető Tamás
egyetemi tanársegéd

Szerző:

Hallgató Hanga
programtervező informatikus BSc

Budapest, 2026

Tartalomjegyzék

Bevezetés	2
1. Dokumentumok közös szerkesztése	3
1.1. A rendszerrel szemben támasztott követelmények	4
2. A szinkronizáció megközelítései	5
2.1. Operációs transzformáció	5
2.1.1. Transzformációs függvény	5
2.2. Konfliktusmentes replikált adattípusok	6
3. Tervezés és megvalósítás	7
3.1. Az előzmények tárolása	8
4. Kiértékelés	9
4.1. Az eredmények megvitatása	10
Összegzés	11
Köszönetnyilvánítás	12
Irodalomjegyzék	13
Ábrajegyzék	14
Táblázatjegyzék	15

Bevezetés

A szöveges dokumentumok ma már csak kivételesen születnek egyetlen szerző kezében. A hallgatók párban írják szemináriumi dolgozataikat, a fejlesztők közösen tartják karban a műszaki dokumentációt, a kutatócsoportok pedig az utolsó pillanatban, egyszerre több helyszínről fejezik be cikkeiket. Ahhoz, hogy az ilyen írás működjön, a szerkesztőnek biztosítania kell, hogy minden résztvevő ugyanazt a szöveget lássa, és egyetlen módosítás se vesszen el.

Jelen dolgozat célja, hogy összehasonlítsa a dokumentumok valós idejű közös szerkesztése során alkalmazott szinkronizáció két fő megközelítését, az operációs transzformációt és a konfliktusmentes replikált adattípusokat, és egy webes szerkesztő prototípusán ellenőrizze őket. Az első fejezet meghatározza az ilyen rendszerrel szemben támasztott követelményeket, a második bemutatja mindkét megközelítést, a harmadik a prototípus tervét, a negyedik pedig a mérések eredményeit.

1. fejezet

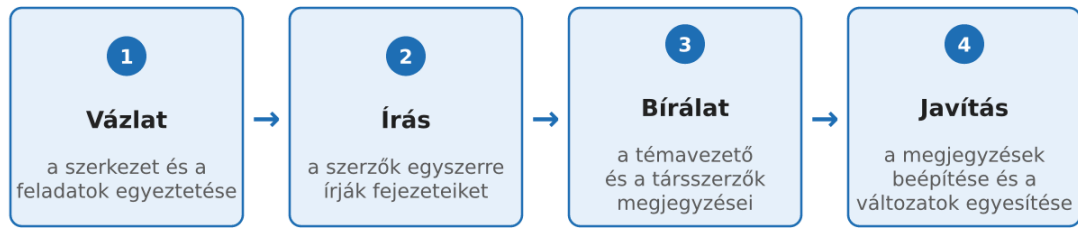
Dokumentumok közös szerkesztése

Közös szerkesztésen azt a helyzetet értjük, amikor több felhasználó módosítja ugyanazt a dokumentumot, és változtatásai kézi összefésülés nélkül jutnak el egymáshoz. Aszerint, hogy a változtatások milyen gyorsan terjednek, három alapvető munkamódot különböztetünk meg:

- **aszinkron szerkesztés**, amelynek során a szerzők a dokumentum teljes változatait adják át egymásnak, és csak utólag fésülik össze őket,
- **zárolásos szerkesztés**, amikor a dokumentum egy adott részét egyszerre csak egy szerző módosíthatja,
- **valós idejű szerkesztés**, amelynek során minden változtatás azonnal megjelenik a többi felhasználónál.

Ez a dolgozat kizárólag az utolsó munkamóddal foglalkozik. Ez a felhasználók számára a legkényelmesebb, ugyanakkor a rendszer tervezésével szemben a legnagyobb követelményeket támasztja, mivel a változtatások sok helyen egyidejűleg keletkeznek, és eltérő sorrendben érkeznek meg.

A valós idejű szerkesztés ráadásul nemcsak a technikát változtatja meg, hanem az egész együttműködés menetét is. A közös írás jellemző folyamatát, ahogyan a megkérdezett felhasználók leírták, az 1.1. ábra szemlélteti.



a 2-4. lépés ismétlődik, amíg a bírálát lényeges megjegyzéseket hoz

1.1. ábra. A dokumentum közös írásának jellemző menete

1.1. A rendszerrel szemben támasztott követelmények

Meglévő szerkesztők felhasználóival készített interjúk és a szakirodalom alapján négy követelményt fogalmaztunk meg, amelyeknek egy közös szerkesztést támogató rendszernek meg kell felelnie:

1. konvergencia, vagyis az összes változtatás kézbesítése után a dokumentum minden másolatának azonos a tartalma,
2. a szándék megőrzése, vagyis a módosítás eredménye megfelel annak, amit a szerző a módosítás pillanatában látott,
3. válaszkészség, vagyis a helyi módosítás azonnal érvényesül, anélkül hogy a szerverre kellene várni,
4. hibatűrés, vagyis a kapcsolat rövid megszakadása nem vezet a félig megírt szöveg elvesztéséhez.

Az első két követelmény a helyességre, a másik kettő a használhatóságra vonatkozik. A gyakorlat azt mutatja, hogy éppen az azonnali válaszkészség és a garantált konvergencia együttes biztosítása a teljes tervezés legnehezebb része.

2. fejezet

A szinkronizáció megközelítései

A valós idejű közös szerkesztést támogató rendszerek minden változtatást először helyben hajtanak végre, és csak ezután küldik el a többi résztvevőnek. Elsősorban abban különböznek, hogyan kezelik azt a helyzetet, amikor két változtatás egyidejűleg keletkezik, és az egyes résztvevők eltérő sorrendben kapják meg őket.

2.1. Operációs transzformáció

Az operációs transzformáció Ellis és Gibbs munkáján alapul [1], akik elsőként írták le, hogyan lehet egy egyidejű műveletet úgy módosítani, hogy egy másik, a szerzője által nem ismert művelet után is végrehajtható legyen. Minden módosítást a szöveg egy adott pozícióján végzett beszúrási vagy törlési műveletként rögzítenek.

Ha például az egyik felhasználó egy szót szúr be egy bekezdés elejére, a másik pedig ugyanekkor töröl egy szót annak végéről, a törlés pozícióját a beszúrt szó hosszával el kell tolni. Ezt az eltolást a transzformációs függvény számítja ki.

2.1.1. Transzformációs függvény

A transzformációs függvény egyidejű műveletek egy párját kapja meg, és azok módosított változatait adja vissza. Ahhoz, hogy a dokumentum összes másolata konvergáljon, a függvénynek teljesítenie kell az úgynevezett transzformációs tulajdonságokat, amelyek teljesülését általános műveletekre nehéz bizonyítani. A legtöbb éles rendszer ezért egy központi szerverre támaszkodik, amely egyetlen műveleti sorrendet határoz meg.

2.2. Konfliktusmentes replikált adattípusok

A második megközelítést a konfliktusmentes replikált adattípusok jelentik, amelyeket összefoglalóan a CRDT rövidítéssel jelölnek [2]. A műveletek utólagos módosítása helyett a dokumentum olyan reprezentációját választják, amelyben az egyidejű műveletek kommutálnak, és sorrendjük nem befolyásolja az eredményt.

Egy replikált adattípus konfliktusmentes, ha bármely két replika, amely ugyanazt a frissítéshalmazt kapta meg, a kézbesítés sorrendjétől függetlenül azonos állapotban van.

A dokumentum minden karaktere egyedi azonosítót kap, amely a karakter teljes élettartama alatt változatlan marad. A beszúrás pozícióját ezután nem egy szám, hanem a szomszédos karakter azonosítójára való hivatkozás határozza meg, így a szöveg más részein végzett egyidejű módosítások nem tolhatják el.

3. fejezet

Tervezés és megvalósítás

A szerkesztő prototípusa egy webes kliensből, egy szinkronizációs szerverből és egy adatbázisból áll, amelybe a szerver folyamatosan menti a dokumentumok állapotát. A kliens és a szerver úgy készült, hogy a szinkronizációs réteg cserélhető legyen, és mindkét ismertetett megközelítés ugyanazon a felhasználói felületen legyen összehasonlítható.

A kliens a dokumentum helyi replikáját tartja fenn, és a felhasználó minden módosítását a replikált adattípuson végzett műveletté alakítja. A műveleteket kötegekben küldi a szervernek, hogy gyors gépelés közben ne terhelje túl a hálózatot. A beérkező köteg szerveroldali feldolgozását egyszerűsítve a következő kódrészlet mutatja be:

```
1 def koteg_feldolgozasa ( dokumentum , koteg ) :
2     for muvelet in koteg . muveletek :
3         if muvelet . id in dokumentum . alkalmazott :
4             continue
5         dokumentum . alkalmaz ( muvelet )
6         dokumentum . alkalmazott . add ( muvelet . id )
7     szetkuldes_a_tobbieknek ( dokumentum . id , koteg )
```

Mivel a műveletek kommutálnak, a szervernek nem kell a kötegeket sorba rendeznie vagy transzformálnia. Elég, ha biztosítja, hogy egyetlen műveletet se hajtson végre kétszer, és minden köteg eljusson az összes csatlakozott klienshez.

3.1. Az előzmények tárolása

A szerver nemcsak a végleges szöveget tárolja, hanem a műveletek teljes naplóját is. A naplóból a dokumentum bármely korábbi változata visszaállítható, és újraépíthető egy hosszabb ideig lecsatlakozott kliens replikájának állapota. Ez a megközelítés a megváltoztathatatlan eseménynaplóra épülő architektúrának felel meg, ahogyan azt Kleppmann leírja [3]. Hogy a napló ne növekedjen korlátlanul, a szerver rendszeres időközönként pillanatképekbe tömöríti.

4. fejezet

Kiértékelés

A prototípust valós szerkesztési munkamenetek felvételeinek halmazán hasonlítottuk össze az operációs transzformáció egy referencia-implementációjával. Minden munkamenetben mértük a késleltetést, vagyis a szerzőnél végzett módosítás és a többi résztvevőnél való megjelenése között eltelt időt, a munkamenet átlagos késleltetését pedig a (4.1) szerint számítottuk ki:

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

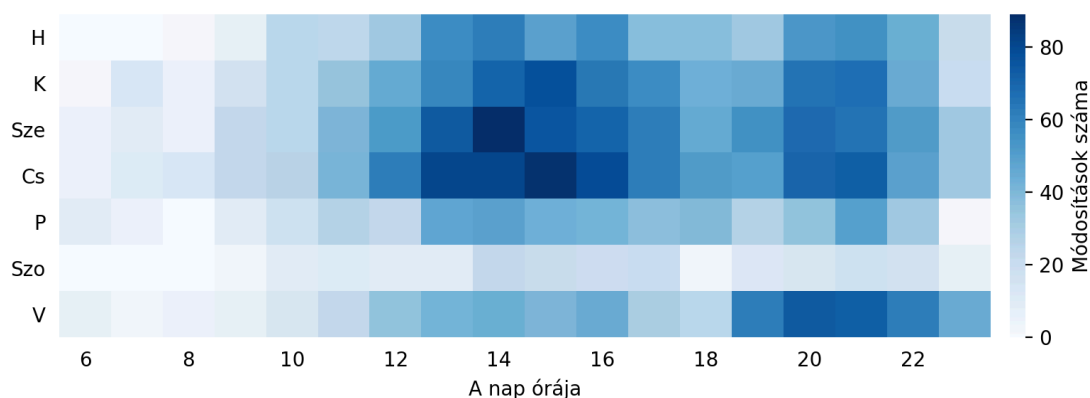
ahol n a munkamenet műveleteinek száma. Az egyidejűleg dolgozó felhasználók különböző számára mért értékeket a 4.1. táblázat foglalja össze.

4.1. táblázat. Átlagos késleltetés és a tárolt adatok mérete a felhasználók számának függvényében

Felhasználók	OT késleltetés (ms)	CRDT késleltetés (ms)	Adatok (kB)
2	38	35	12
5	52	41	19
10	87	48	31
20	164	63	58

A felhasználók számának növekedésével az operációs transzformáció késleltetése jóval gyorsabban nő, mivel a szervernek minden műveletet az összes egyidejű művelettel szemben transzformálnia kell. A CRDT alacsonyabb késleltetésének ára a tárolt adatok nagyobb mennyisége, hiszen minden karakter hordozza a saját azono-

sítóját. A módosítások időbeli eloszlását, amelyre a terheléses teszt épül, a 4.1. ábra mutatja.



4.1. ábra. A módosítások száma a hét napja és az óra szerint a vizsgált munkame-
netekben

4.1. Az eredmények megvitatása

Az eredmények megerősítik, hogy kisebb csapatok esetén a két megközelítés egyenértékű, és inkább a megvalósítás egyszerűsége a döntő. Több egyidejű felhasználó esetén a CRDT egyértelmű előnyben van mind a késleltetés, mind a kapcsolatkiesésekkel szembeni ellenállóképesség tekintetében. A felhasználókkal folytatott kiegészítő interjúk, amelyeket a kvalitatív kutatás elvei szerint végeztünk [4], azt mutatták, hogy a száz milliszekundum alatti késleltetéskülönbséget a felhasználók nem érzékelik, a félig megírt szöveg kapcsolatkiesés utáni elvesztését viszont komoly problémának tartják.

Összegzés

A dolgozat a dokumentumok valós idejű közös szerkesztése során alkalmazott szinkronizáció két megközelítéseként hasonlította össze az operációs transzformációt és a konfliktusmentes replikált adattípusokat. Egy webes szerkesztő prototípusán megmutattuk, hogy a CRDT a műveletek központi sorba rendezése nélküli, egyszerűbb szerveret tesz lehetővé, és több felhasználó esetén alacsonyabb késleltetést nyújt.

A CRDT fő hátránya továbbra is a tárolt adatok nagyobb mennyisége. A következő lépés ezért annak vizsgálata lesz, hogyan befolyásolja az adatmennyiség a napló rendszeres pillanatképekbe tömörítését, és hogyan viselkedik a prototípus hosszú távú üzemben, több tízezer dokumentummal.

Köszönetnyilvánítás

Ha a dolgozat egy projekt vagy az egyetem pénzügyi támogatásával készült, azt jellemzően itt kell feltüntetni. Itt mondhatod köszönetet az oktatóknak, hallgatótársaknak és kollégáknak is, akik segítettek.

Irodalomjegyzék

- [1] Clarence A. Ellis és Simon J. Gibbs. „Concurrency control in groupware systems”. *Proceedings of the 1989 ACM SIGMOD International Conference on Management of Data*. New York: ACM, 1989, 399–407. old. DOI: 10.1145/67544.66963.
- [2] Marc Shapiro és tsai. „Conflict-free replicated data types”. *Stabilization, Safety, and Security of Distributed Systems*. 6976. köt. Lecture Notes in Computer Science. Berlin: Springer, 2011, 386–400. old. DOI: 10.1007/978-3-642-24550-3_29.
- [3] Martin Kleppmann. *Designing data-intensive applications*. Sebastopol: O’Reilly Media, 2017. ISBN: 978-1-4493-7332-0.
- [4] Jan Hendl. *Kvalitativní výzkum: základní metody a aplikace*. Praha: Portál, 2005.

Ábrák jegyzéke

1.1. A dokumentum közös írásának jellemző menete	4
4.1. A módosítások száma a hét napja és az óra szerint a vizsgált munkamenetekben	10

Táblázatok jegyzéke

4.1. Átlagos késleltetés és a tárolt adatok mérete a felhasználók számának függvényében	9
--	---