Elosztott adatfeldolgozás

Tanulmány az Adatbázisok haladóknak c. tárgyhoz

Bujáki Attila

4. évf. BSc. Mérnök informatikus szak

2012/2013 tanév I. félév

Tartalomjegyzék

Bevezetés

Az elmúlt években a nagyméretű földrajzilag is elosztott működésű, cloud computing paradigmán alapuló alkalmazások a mindennapokban egyre fontosabbá és használatuk is egyre elterjedtebbé vált, így egyre nagyobb helyet kaptak a mindennapi életünk számos területén is. Az ilyen alkalmazásokat tekintve a rendelkezésre állás, a hibatűrés, és a megfelelő teljesítmény biztosításának kérdésköre egyáltalán nem triviális kérdés. Ma ez egy rendkívül aktív kutatási területet képvisel. Bizonyos tulajdonságok biztosítása érdekében gyakran másokról le kell mondani, gyakran mindezt pedig úgy, hogy a megoldás eredménye a lehető leginkább szolgálja üzleti célokat.
A következőkben összefoglalom az elosztott működés főbb kérdéseit, az elosztottság módozatait, a nagyméretű elosztott rendszerek adattárolásával kapcsolatban a konzisztencia kérdéskörét, különféle adat konzisztencia modellek is bemutatásra kerülnek, illetve végezetül bemutatok néhány gyakorlati implementációját a különböző bemutatott adat konzisztencia modelleknek, amelyek már éles helyzetekben is bizonyítottan helytálltak.

Előzmények

Korábban az adatbázisok tárgy keretein belül már számos az elosztott adatfeldolgozáshoz kapcsolódó megoldást, algoritmust és lehetőséget megismerhettünk. A laboratóriumi gyakorlatok során pedig közelebbi viszonyba is kerülhettünk az adatbáziskezelő rendszerekkel, viszont bizonyos, a manapság már nap, mint nap használt rendszereinket jellemző földrajzilag is elosztott működés gyakorlati problémáinak bizonyos aspektusaira és konkrét megvalósításokra sor és idő sajnos már nem került.
A munkaerő piaci követelmények is megkövetelik ezen új technológiák ismeretét is. Állásinterjú keretein belül már én magam is szembesültem azzal, hogy ezek a témák bizony-bizony előjönnek. Ez pedig nagyon is fontos, hogy az ember egy ilyen kritikus és döntő helyzetben pozitív képet fessen magáról a leendő munkáltatóban. Az ilyen felmerülő technológiák előzetes ismerete egy fontos többletet ad, amely majd megkülönböztethet minket a többi versenytársunktól. Az én konkrét esetemben a NoSQL adatbáziskezelők merültek fel egy interjú során, ahol egy fejlesztői pozícióra pályáztam.
Ezeknek a modern elosztott megoldásoknak a kellő megértéséhez viszont tisztában kell lenni alapvető ismeretekkel az elosztott működéssel kapcsolatban és az elosztott környezet számos problémájával és a lehetséges megoldásokkal, hogy átfogó képet kaphassunk az új technológiák miben létéről is.
A különféle elosztott algoritmusok ismerete, azoknak tervezésének kérdéseinek ismeretei is sokat segíthetnek a témakör jobb megértésében, ráadásul ezeket fejlesztőként amúgy is hasznosíthatjuk. Az elosztott algoritmusokról egy mély és meglehetősen átfogó alkotás a Desing and Analysis of Distributed Algorithms, Nicola Santoro-tól. [6] Ez a mű általánosan és nem adatbázis specifikusan mutatja be a különféle elosztott algoritmusokat, de az általános tárgyalásnak köszönhetően rendkívül jól használható tudásra tehetünk szert a könyv oldalait lapozgatva.

Kontextus

Bevezetésképp szeretném elhelyezni a tanulmányomban kifejtett témákat az eddig tanultak és a többi, a tárgy keretein belül bemutatott téma környezetében.
Szerettem volna olyan szemelvényeit kiválasztani ennek a meglehetősen tág témakörnek, amelyek az adatbázisok tárgyból tanultakra építve a többi haladóbb témakör felé nyit utat, illetve egyéb területekre ad betekintést.
A NoSQL témakörről ugye már korábban hallhattunk a félév során, a Map/Reduce-os téma és az elosztott architektúrákról szóló előadás pedig ezután fog következni. Ezeken a területeken kívül is próbáltam minél érdekesebb és több irányba nyitni.
Ahhoz, hogy elosztott adatfeldolgozásról beszélhessünk először is beszéljük meg, hogy mit értünk jelen esetben elosztottság alatt. Az adatbázisokból már tanultaknak megfelelően:

Definíció: *elosztott adatbázis:* csomópontok (node, site) összessége, amelyekben egy-egy számítógép üzemel saját CPU-val, háttértárral és adatbáziskezelővel. A csomópontok egymással össze vannak kötve adatcserére alkalmas módon úgy, hogy az adatbáziskezelők felhasználói csupán egyetlen, logikailag egységes adatbázist érzékelnek. [1]

A problémáink itt, a nagyméretű földrajzilag is elosztott alkalmazásoknál, majd a definíció utolsó részével lesznek. A logikai egység kérdésének megvalósítása rendkívül sarkalatos pontja az ilyen rendszerek tervezésének, mivel a teljes logikai egység, a teljes konzisztencia és a DDBMS teljesen transzparens működése (ez összefügg a konzisztenciával is) rendkívül költséges lesz bizonyos esetekben, és ez sokszor nem megengedhető különféle üzleti célok miatt. [5]

Elosztott működés

Az adathalmazok elosztása (az elosztott működéshez) a csúcspontok között különbözőképpen történhet. [2]
Egyik lehetőség a **felbontás** (fragmentation). Ez azt jelenti, hogy az adathalmazt bizonyos dimenziók mentén részekre bontjuk és ezeket a részeket külön csomópontokban tároljuk el.

Relációs adatmodellt feltételezve ez jelentheti például azt, hogy egy adott reláció bizonyos sorait tároljuk csak egy csúcsban, míg más sorokat esetleg több (, különböző) másik csúcsban tárolunk el.

A **függőleges felbontás** a másik kínálkozó lehetőség, ami kínálkozik a relációs adatmodellnél. Ez azt jelenti, hogy egy reláció bizonyos oszlopait választjuk ki az egy-egy csúcsban való tároláshoz. Ha ezt választjuk nem szabad megfeledkeznünk arról, hogy a reláció helyreállíthatóságához figyelembe kell vennünk azt is, hogy az egy-egy csomópontba kerülő „reláció darabok" egyértelműen azonosíthatóak legyenek, hogy mely sorhoz tartoznak. Ennek a biztosítására egy könnyű megoldás lehet, ha felveszünk egy plusz mezőt minden egyes csomópontba kerülő oszlophalmazhoz, amely csak arra szolgál, hogy azonosítsa a reláció sorait, így biztosítva, hogy a felbontást követően is veszteségmentesen visszaállítható legyen a reláció. A probléma nagyon hasonló az adatbázisokból már tanult veszteségmentes felbontás témaköréhez. [1] *(Itt is hasonlóan „feltördelt" darabokból kell tudnunk visszaállítani az eredeti adathalmazt, információveszteség nélkül.)*
A két felbontási módszert természetesen **keverhetjük** is. Ilyenkor egy csúcsba csupán egy teljes relációnak, a bizonyos oszlopaihoz tartozó értékek közül is csak a reláció bizonyos soraihoz tartozók kerülnek. Itt is ügyelni kell a függőleges felbontás kapcsán felmerülő problémára, hogy ne történjen adatvesztés a felbontás következtében.
A felbontásnak köszönhetően a csúcsok között megoszthatjuk a terhelést. Bizonyos adathalmazokat akár közelebb is vihetünk a felhasználási helyükhöz, így csökkentve a válaszidőket a hálózati késleltetés csökkentésével. Ez például egy elosztott vállalati infrastruktúránál vonzó lehetőség lehet.
Például az adott ügyfélszolgálatokon helyben rendelkezésre állhat az ottani működéshez szükséges adathalmaz, így gyorsítva a hozzáférést, lecsökkentve a nagy távolságok áthidalásából következő hálózati késleltetés okozta magasabb válaszidőket.
Ha csak felbontást használunk, akkor az ilyen esetben távolabbi csúcsokból történő elérések viszont akár még lassulhatnak is egy központosított megoldáshoz képest.

Például ha egy központi irodában adatelemzést szeretnénk végrehajtani, akkor sokkal üdvösebb lenne számunkra, ha a központban rendelkezésre állna minden információ a gyors működéshez, és nem kellene az egyes irodák szervereihez fordulni (a WAN-on).

A replikáció röviden annyit jelent, hogy egy bizonyos adathalmazból több másolati példányt is tartunk. Az ezzel kapcsolatosan felmerülő problémákról már sok szó esett az adatbázisok tananyagban is. A módosításokat valahogy el kell jutatnunk a többi példányhoz is, ennek megoldása pedig számos különböző módon történhet, ezeknek megfelelő megoldások különböző előnyökkel és hátrányokkal járhat.
Alapvetően három kategóriába rendezhetjük a „replikáltság mértékét".

- Lehet szó arról, hogy egy adathalmaz, mondjuk egy reláció bizonyos sorai több csomópontban megtalálhatóak – ekkor beszélünk replikációról.
- Lehet szó **teljes replikáció**ról. Ilyenkor az adott adathalmaz minden eleme megtalálható több különböző csomópontban is.
- **Teljesen redundáns adattárolás** is megvalósítható replikációval. Ilyenkor minden egyes csúcsban megtalálható a teljes adathalmaz. Ez nyilván jó, ha olvasásokról van szó (a terhelés eloszthető, a hálózati késleltetési időkből származó válaszidő növekedés minimalizálható), de ha írásról van szó, akkor jelentős többlet feladatot a változásokat mindenhova eljuttatni. Természetesen ez jelentős többlet tárhelyet is igényel, a többlet példányok tárolása miatt.
- És természetesen lehet kombinálni a különböző replikációs lehetőségeket és a felbontás különböző változatait is különféleképpen, akár egyszerre.

Adatreplikáció

Az adatreplikáció nem egy új keletű dolog. Számos célszoftver és célhardver létezik már. Az információs rendszerek üzemeltetése tárgyból is számos ezzel kapcsolatos technológiát megismertünk. Ide tartozik a biztonsági mentések készítése, az archiválás, illetve az adatbázistartalmak replikálása is

idetartozhat. De még bizonyos RAID technológiák alkalmazása is a tárolt adatok redundáns tárolásával biztosítják a magasabb megbízhatóságot és a hibatűrést. [3][4]
De most a mi szempontunkból az adatbázisok témakörét tekintve vizsgálódunk.
Az elmúlt években a cloud computing, a számítási felhőket alkalmazó alkalmazások lenyűgöző fejlődését láthattuk. A mindennapi életünk részévé vált az új felhő paradigmán alapuló alkalmazások használata, legyen szó egy google szolgáltatás igénybevételéről, vagy könyvvásárlásról az Amazon-on vagy éppen az „online társasági életünkről" a facebook-on. Ezekben az alkalmazásokban az a közös, hogy földrajzilag elosztott módon működnek, a világ bármely pontjáról elérhetőek és hatalmas felhasználói kört kiszolgálnak, amely ráadásul egyre csak tovább növekszik. A felhasználói élményt tekintve a válaszidők meghatározó fontosságúak, ha ezek az alkalmazások nem támaszkodnának a replikáció nyújtotta lehetőségekre, jelentősen nagyobb válaszidőkkel kellene számolni, és így a felhasználói bázisuk jelentős részét el is vesztenék. Ezeknek a nagyméretű elosztott georeplikált adathalmazokra támaszkodó alkalmazásoknak szembe kell nézniük azzal a problémával is, amit már az adatbázisok tárgyban is részletesen megismertünk, hogy a különböző csomópontokban, a különböző adattárházakban található adathalmazok közötti konzisztenciát fent kell tartani [1] (, „többé-kevésbé", mint látni fogjuk. Erről lesz szó a későbbiekben.). Mindezt úgy szeretnénk elérni, hogy ez megvalósítás az egész adathalmazt érintő nagyszámú párhuzamos módosítás ellenére is a megfelelő funkcionalitást biztosítsa.
Az ilyen földrajzilag elosztott környezetben történő replikációt nevezzük georeplikációnak. Ez a fogalom nagyon szorosan kötődik a felhő infrastruktúrán alapuló alkalmazások fejlesztésének témaköréhez.
A paradigmához köthető szlogenek, hogy igény szerint, vagy hogy bárhonnan elérhető erőforrások használata, bizony nem azt jelenti, hogy tényleg mindegy, hogy hol tároljuk az adatainkat.
A korábban már említett szolgáltatások a világ bármely pontjáról elérhetőek. A felhasználók jelentős elvárásokkal fordulnak a szolgáltatáshoz. A válaszidők pedig meghatározóak a felhasználói élményben, így ezeket a minimálisra kell csökkenteni, hogy minél több felhasználót csábíthassanak ezek a szolgáltatók magukhoz. Így egyre és egyre több felhasználót kell kiszolgálniuk, pedig a felhasználóik száma már ma is hatalmas. Ez pedig csak úgy lehetséges ha a felhasználókhoz „közelebb visszük" az általuk használt adathalmazokat, ezáltal csökkentve a válaszidőket.
Ez egy olyan elrendezést eredményez ahol az adathalmazok számos adatközpontban találhatók meg, különböző földrajzi helyeken. Az adat replikációra kerül a különböző adattárházak között és azokon belül is.
Ideális esetben a futásidőben, a köztes réteg feladataként, a fejlesztők elől teljesen el lenne rejtve, hogy valójában egy elosztott adathalmazon dolgoznak. Nem kellene a szoftverben különleges esetekkel törődniük, úgy látszódna számukra az adathalmaz mintha lokálisan elérhető lenne. Nem kellene a csomópontok és a linkek változatos hibáival, vagy az inkonzisztenciával törődni. Ez lenne a DDBMS feladata, hogy mindezt elrejtse a fejlesztők elől is. Ezt nevezzük transzparens működésnek. (Átlátszó, mintha ott se lenne.)
Az izolációt is biztosítania kellene a DDBMS-nek, hogy az alkalmazás konkurens, vagy a különböző alkalmazáspéldányok konkurens hozzáférései izolált módon történhessenek.
Ezeknek a biztosítására két plusz fő feladata lesz a DDBMS-nek:
- A konkurens hozzáférés szabályozást biztosítania kell, ill.
- A replikáció kezelést is biztosítania kell.

Látni fogjuk, hogy a magasabb konzisztencia biztosítása jelentős költségeket jelent a georeplikált környezetben, mivel az a csomópontok komoly együttműködését igénylik és az egy másolati példány szintű konzisztencia biztosítása az ilyen elrendezésben egyszerűen nem megengedhető.
- Ezzel kapcsolatban szó lesz számos adat konzisztencia modellről, majd bizonyos konkrét implementációkról.

A georeplikáció magában is hatalmas üzlet. Olyan cégek mint az iOra, és a Syntergy foglalkoznak vele. A hajókon lévő rendszerek és a part közötti adatátvitel, illetve a különböző katonai célok a legjellemzőbb területek, de még számos információt megtudhatunk erről is a honlapjaikon keresgélve.
A geopreplikált környezet problémakörét nagyon szépen bemutatja az irodalomjegyzékben szereplő [5] tanulmány. Munkám során sokat merítettem belőle.

Konkurens hozzáférés vezérlés

Röviden beszéljünk kicsit a konkurens hozzáférés vezérlésről. Ha mondjuk, van két műveletünk és ezeket végrehajtjuk, úgy hogy az egyik végeztével indítjuk a másikat, akkor soros végrehajtásról beszélünk. Ezzel valószínűleg senkinek sem mondtam újat. A soros végrehajtást könnyen tudjuk követni, viszont ha konkurens módon hajtjuk végre a műveleteket, akkor viszont a végeredmény sokszor nem ilyen egyértelmű meghatározni. Ha veszünk egy egyszerű FIFO adatstruktúrát is, már

akkor is bajba lehetünk a konkurens működést tekintve. A problémát a tranzakciók fogalmának a bevezetésével probáltuk megfogni. Elosztott környezetben ezeknek a megvalósítása nehéz. Bizonyos esetekben ennek megfelelően ezeket nem is valósítják meg, így például egyes NoSQL adatbáziskezelőkben – pl.: MondoDB, ahogy a NoSQL-es előadáson elhangzott.
Egy kézenfekvő és egyszerű megoldás, hogy ne is legyen konkurens hozzáférés. Ez nyilvánvalóan nem kielégítő megoldás. A legtöbb esetben a műveleteket különböző adathalmazokon végezzük. Ezek egyáltalán nem zavarják meg egymás hatását, így ha ezt választanánk, akkor jelentős teljesítmény visszaesést tapasztalnánk, illetve rendkívül hosszú végrehajtási időket.
Ez nyílván nem megengedhető. Így muszáj lesz az adatbáziskezelő rendszeinknek elosztott esetben is megvalósítani valamiféle konkurens hozzáférés vezérlést, hogy a konkurens hozzáférések ellenére is valamiféle „elvárt, elfogadható” eredményt kapjunk a műveletek elvégzését követően.
Ezeket az elfogadott eredményeket definiálja majd számunkra, hogy melyik végrehajtása lesz korrekt, érvényes az adott konkurens műveleteknek és melyik nem.
Ezeknek a konkurens műveletek végrehajtásának a korrektségének a biztosítására kerülnek implementációra a különböző konkurens hozzáférés vezérlő mechanizmusok.
Ahogy korábban megbeszéltük a soros sorrendeket könnyebben átlátjuk. Ennek megfelelően azt szeretnénk, ha úgy hajtódnának végre az általunk kért műveletek az adatokon, mintha valamiféle soros sorrendben történtek volna. Ezt próbálja megfogni a linearizálhatóság és a sorosíthatóság fogalma. Ezt a két modellt gyakran nevezzük erős konzisztencia kritériumoknak.
Sajnos, ahogy látni fogjuk az erős konzisztencia modellek megvalósítása a georeplikált rendszerekben rendkívül költséges, mivel az szoros együttműködést igényel az adattárházak között, amely együttműködés pedig élénk kommunikációt igényel, amelyet a nagy méretű hálózatokban megfigyelhető nagy hálózati késleltetés jelentősen lassít. Továbbá annak érdekében, hogy a megfelelő teljesítményt tudják nyújtani ezek a rendszerek, gyakran lazább adatkonzisztencia modelleket kell megvalósítaniuk, amelyek a fejlesztők számára is felfedik az elosztott működést, viszont kisebb költségeket jelentenek.

Replikáció menedzsment

A replikáció több célt is szolgál a nagyméretű elosztott rendszerekben. Egy részről lehetővé teszi, hogy elosszuk az olvasás jellegű műveletek által keletkező terhelést több csomópontra a replikák között, azon az áron, hogy az írás jellegű műveletek során többlet koordinációra van szükség a konzisztencia helyreállítására. Sok konkrét alkalmazás esetében az olvasás jellegű műveletek száma a meghatározó az írásokéval szemben, így a replikáció általában teljesítmény szempontból is kifizetődik.
Továbbá a replikáció alkalmazásával növelhető a rendszer megbízhatósága és hibatűrése.
Ha egy adott csúcspont kiesik a rendszer tovább működhet, akár zavartalanul is, ha még az adat elérhető egy másik csúcsban. Így akár jelentős természeti katasztrófákat is átvészelhetünk úgy, hogy a szolgáltatás és a rendszer elérhető marad. Egész adattárházak semmisülhetnek meg tűzben, vagy földrengésben, de a szolgáltatásunk tovább működhet. (Hibatűrés.)
Teljesítmény romlás persze előfordulhat, mert lehetséges, hogy az új csomópont már nagyobb földrajzi távolságra van a kérés helyétől.
A válaszidők javításában is jelentős szerepet játszik a replikációk megléte. A korábbiakban már kiemeltem, hogy üzleti szempontból ez egy jelentős tényező. A replikáció segítségével az adathalmazokat a felhasználóikhoz közelebb is elhelyezhetjük.
A fő kihívás amivel találkozunk a replikáció során az, az, hogy hogyan tartsuk a különböző replikákat konzisztens állapotban.
Ismételten az a helyzet, hogy ideális esetben fenntarthatnánk azt a látszatot mintha csak lokális adatpéldányokhoz férnénk hozzá a távoli esetben is és az egész replikációs és a földrajzilag elosztott működést elfedhetnénk. Könnyebb dolga lenne a fejlesztőknek is, mivel nem kellene egy „csomó” különleges esettel foglalkozniuk. Sajnos ez földrajzilag elosztott esetben sokszor nem lehetséges, mivel a WAN-ra jellemző nagy késleltetési idők miatt csak gyengébb konzisztencia modelleket tudunk használni.
Az egy másolati példánynak megfelelő konzisztencia biztosítására számos módszert tanultunk az adatbázisok tárgy keretében.
CAP – válassz kettőt. Erről is szó volt már korábbi előadásokon, ezért ezzel nem foglalkozok. Forrás: [5]

Tanulságok

Az előzőekben megállapítottuk, hogy valóban előnyös a replikáció. Hasonlóan a konkurrens hozzáférés szabályozás is. Erős érvek szolnak mind a kettő alkalmazása mellett, és számos esetben ezek már

bizonyítottak a gyakorlatban is. Viszont a georeplikációt jellemző magas késleltetési idők miatt a csúcsok koordinációja rendkívül költséges lehet. A hálózat esetleg részekre is szakadhat. Ezeknek a következményeképpen nem lehetséges az erős konzisztencia fenntartása. Enyhébb modelleket kell követnünk. Ilyenkor viszont a fejlesztő számára nem transzparens az elosztott, georeplikált adatkezelés. Különféle anomáliákat tapasztalhat és különleges esetekre kell felkészülnie, amelyek az inkonzisztenciából fakadnak és abból, hogy nem kerül elrejtésre előle teljes mértékben az adatok georeplikált létezése, illetve hogy a korrektnek elfogadott futása eredmények halmaza szélesebb.
Mindezt tovább tetézi az a tény, hogy az ilyen georeplikált rendszerek általában hatalmas számú felhasználót szolgálnak ki, ezek pedig több ezer kéréssel fordulnak a rendszerhez másodpercenként, amelyeket a lehető legrövidebb időn belül meg kell válaszolni a megfelelő felhasználói élmény fenntartása érdekében, így az üzleti célokat is szolgálva. Ez az infrastruktúrát mindinkább egy olyan irányba viszi el, hogy nagyon nagy méretekben is kis késleltetéssel végezze a műveleteket. Ezeknek a követelményeknek a kielégítése csak úgy lehetséges ha nagyon **hatékony algoritmusokat alkalmazunk, amelyek gyakran azt jelentik, hogy a konzisztencia kritériumainkat is enyhíteni kell.**

Forrás: [5]

Adat konzisztencia modellek

Feltevések

Mielőtt még a konkrét, adat konzisztencia modellekkel foglalkoznánk, tisztázzunk pár feltevést a diszkussziónkkal kapcsolatban. Jelen esetben csak regiszter típusú objektumokról fogunk beszélni. Ha másról lesz szó, akkor azt külön kiemelem. A regiszter az egyik legegyszerűbb adatelem. Csak írás és olvasás műveletet hajthatunk végre rajta. Az írás felülírja a korábbi tartalmat.
Vannak összetettebb működést lehetővé tevő adatelemek is. (Összetettebb szemantikával bírók.) Ilyenek lennének például a halmazok ahol még hozzáadás művelet is van. Ez más szemantikát takar. A végrehajtás kommutatív.

Bizonyos adat konzisztencia modellek lehetővé teszik azt, hogy több replikált példányt módosítsunk egyszerre, így konfliktusokat előidézve.

Pl.: Van egy regiszterünk amiből több replikált példány van. Ezek közül többet felülírunk. Melyik értéket tekintjük érvényesnek? Ezek közül valamelyiket ki kell választani. Általában azon egyszerű szabályt alkalmazzák, hogy az utolsó író által beírt tartalom az érvényes. Ez regiszterekre jól működik, de más, komplexebb szemantikával bíró adatobjektum típusokra ez már korántsem ilyen egyszerű.

Léteznek az objektumok szemantikáját figyelembe vevő rendszerek is. Ezeknél például halmaz objektumnál a több hozzáadás művelet eredményét össze kell fésülni, ezzel felismerve azt, hogy a hozzáadás ebben az esetben kommutatív művelet.
A konfliktus megoldás lehet a rendszerben megvalósított funkció is, de akár egy külső komponens is végezheti.

Adat konzisztencia modell típusok

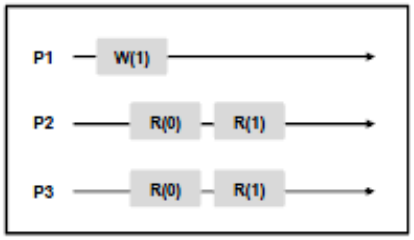
A következő részben a tárgyalásunkat különböző adat konzisztencia modellek megismerésével folytatjuk. Ezeket az szerint csoportosítjuk, hogy tranzakiók kezelésével megvalósítják-e műveletsorozatok izolációját, vagy pedig csak adott műveletekre vonatkoznak és a műveletsorozatokat követő állapotokról nem határoznak meg konkrétan külön semmit.
Először tekintsük a nem tranzakciós adat konzisztencia modelleket.
Ahogy már említettük a tárgyalás során a megosztott adatelemek egyszerű regiszterek lesznek.
A tárgyalásunkat kezdjük a nem tranzakciós modellekkel.

Nem tranzakciós adat konzisztencia modellek

Linearizálhatóság (Atomi konzisztencia modell)

A linearizálhatóság, avagy az atomi konzisztencia modell a legerősebb garanciát nyújtja a tárgyalt nem tranzakciós adat konzisztenia modellek közül, az alkalmazás számára. Tény, hogy a legközelebb ez áll

az idealizált memória absztrakcióhoz, ahol minden művelet azonnal és oszthatatlan módon történik meg.
Továbbá a modellben helyet kap még az is az, hogy minden művelet adott ideig tarthat. A lenti ábrán láthatjuk ezt is. Az írás és olvasás műveletek is a téglalapokkal jelölt időintervallumot igénylik a végrehajtáshoz, a meghívástól a végrehajtásig. Ebben a modellben a végrehajtást követően, azaz az idővalon a téglalap jobb szélétől kezdve viszont a művelet hatásait minden más ezt követően indult művelet azonnal észleli.
A linearizálhatóságnak van egy **kompózabilitás** nevű tulajdonsága, amely azt jelenti, hogy egy rendszer csak akkor és csakis akkor linearizálható, ha minden egyes eleme szintén az. Így fordítva is. Linearizálható komponensekből álló rendszer linearizálható lesz.
Ha van bármely más ezt a tulajdonságot elrontó eleme, akkor már a nagy rendszerre sem lesz igaz ez a tulajdonság.
Sajnálatos módon az, hogy egy hibatűrő replikált rendszerünk a linearizálható adat konzisztencia modell szerint működjön az nagyon költséges.
Ennek megvalósításának az egyik módja az lenne, hogy minden egyes példányt zárolunk írás során. De ez például hibák jelenlétében nem megoldható. Ha például szétszakad a hálózat különböző részekre, akkor a szétszakadt részek egyáltalán nem tudnak kommunikálni, így a zárolás sem megoldható.

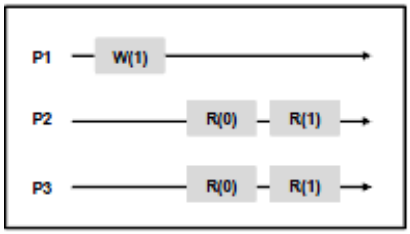


1. ábra - Linearizálhatóság - Egy korrekt lefolyás.

Pl.: A fenti ábrán látható egy linearizálhatóságnak megfelelő lefolyás. Láthatjuk, hogy az írás lezárultával minden egyes végrehajtási szálon már azonnal az új értéket olvassuk. Az írás lezárulása előtt még pedig a korábbi értéket.
Az ábrák értelmezéséhez tudni kell, hogy a kezdeti érték a 0. A W utasításokkal írunk a tárolóba, míg az R-rel olvasunk. Az R után a zárójelben lévő értéket olvassuk ki.
A P1, P2, P3 folyamatok a különböző végrehajtási szálak. Ezek elosztott rendszerben akár külön csúcsokon is futhatnak, különböző replikákat elérve.

Szekvenciális konzisztencia modell

A szekvenciális konzisztencia modellben az kerül megfogalmazásra, hogy több szálon való végrehajtásnak azonos eredményre kell vezetnie, mintha egy azonos szálon hajtanánk végre ezeket a műveleteket valamilyen sorrendben.
Nézzünk erre egy példát:



2. ábra - Egy szekvenciálisan konzisztens, korrekt végrehajtás

A példán látható, hogy az írás hatására már az új érték lesz kiolvasható bizonyos idő elteltével. De azt követően már csak az új értéket olvashatjuk. Ettől szekvenciális.

A linearizálhatósággal ellentétben a szekvenciális konzisztencia nem kompóziábilis.
Ennek következtében helyi példányon végrehajtott szekvenciálisan konzisztens műveletek a rendszer egészére tekintve nem feltétlenül lesznek szekvenciálisan konzisztensek.
Ez azt jelenti, hogy a linearizálhatósággal ellentétben itt hiába hajtok végre csupa ilyen műveletet lokálisan, globálisan ez nem biztos, hogy elegendő lesz a konzisztencia kritérium fenntartásához.

A szekvenciális konzisztencia gyengébb kritérium mint a linearizálhatóság.
Enyhített követelményeket tartalmaz a konzisztenciára nézve.
Itt nem szükséges, hogy az írás műveletek hatása azonnal látható legyen minden olvasás művelet számára az írás befejeztével. (Elosztott georeplikált rendszerben, főleg ha földrajzilag nagy távolságokról beszélünk ez amúgy is nagyon hosszú írás műveleteket eredményezne, mivel az írás hatását minden egyes replikára ki kellene terjeszteni mielőtt a művelet bezárulna, hogy biztosítsuk azt, hogy az olvasás műveletek valóban az új értéket látják – innét a nagy költsége a lineraziálhatóságnak.)
Az ábrán láthatjuk, hogy egy nem linearizálható, szekvenciális konzisztencia modellnek megfelelő rendszerben az írás művelet befejezése után is előfordulhat még, hogy a régi értéket olvassuk. Később már persze az új érték is elérhetővé válik, de az írás művelet befejezésének a hatására még nem látszódik annak hatása minden olvasás számára azonnal.

Rekordonkénti idővonal konzisztencia

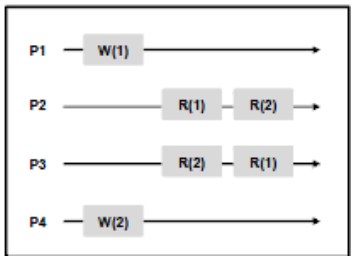
A rekordonkénti idővonal konzisztencia modell a szekvenciális konzisztencia enyhítéséből adódik.
Ez az első modell, amely már felfedi a fejlesztő számára, hogy replikáció van a háttérben.
Abból a felismerésből adódik ez a modell, hogy általában egy adott adattárház, egy adott objektumához fordulunk csak. Ennek megfelelően itt csak egy objektumra biztosítja a korábban bemutatott szekvenciális konzisztenciát. Objektumonként külön idővonal van. Ez azt eredményezi, hogy a korábbi olvasások az adott objektumról mindig egy konzisztens képet mutatnak, de nem feltétlen a legfrissebbet. A korábbi olvasások hatására a későbbiek számára lehet, hogy egy régebbi állapot lesz látható, annak érdekében, hogy egy konzisztens képet láttasson. Azonban viszont ha egy végrehajtási szálból több különböző objektumot vagy adattárat próbálunk elérni, akkor nem biztosítja számunkra a modell a szekvenciális konzisztenciát.
Annak köszönhetően, hogy ez a jellemző hozzáférés, hogy egy adattárházon belül egy adott replikához fordul a kéréseivel a végrehajtási szál, így a replikák esetleges inkonzisztenciái (amelyek a külön idővonalakból következnek) nem okoznak problémát. Általuk nem keletkeznek a fejlesztő által megfigyelhető anomáliák. DE, azonban ha elkezd több adattárhoz fordulni, vagy különböző replikákhoz, akkor anomáliákat figyelhet meg a felhasználó.

Kauzális konzisztencia modell

A kauzális konzisztencia modell azt biztosítja számunkra, hogy a végrehajtott műveletek hatására az eredmények mindig konzisztensek lesznek a Lamport féle definíciójával a „korábban történt” tualjdonságnak, amely egy részleges rendezést biztosít az események között.
Ez informálisan leírva a következő:

- Azonos végrehajtási szálon könnyű meghatározni. Nyílván ami előbb történt a szálon belül az van korábban.
- Ha A írás, és B olvasás ami az A által írt értéket olvassa, akkor A a B előtt történt. (Nyílván mivel már az általa a beírt értéket olvassa ki.)
- Ez a kapcsolat tranzitív. Ha A a B előtt történt, B pedig a C előtt, akkor elmondhatjuk azt is, hogy A C előtt történt.

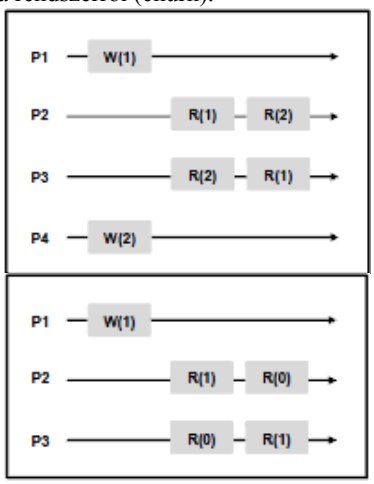
DE – figyeljük meg, hogy így a végrehajtási szálaknak, pont, mint a szekvenciális konzisztenciánál, nem szükséges a legutoljára beírt értéket kiolvasnia.
A konkurens írások sorrendje a különböző végrehajtási szálakon különböző eredményre vezethet.
Gondoljunk csak bele, így is megmarad a korábban történt definícióval a konzisztencia. Az ábrán is egy ilyen eset látható, amikor két írás hatására az egyes replikák más állapotba kerülnek. A replikák ezáltal divergálnak/divergálhatnak.



3. ábra - Korrekt eredmény - kauzális modell alatt

Esetleges konzisztencia modell

Az esetleges konzisztencia (Eventual Consistency) egy olyan fogalom, amelyet az olyan konzisztencia modellekre használunk, amelyek megengedik azt, hogy az írási műveletek hatásait a különböző végrehajtási szálak más sorrendben figyelhessék meg, akár egy hosszabb időtartamra is. Mindazonáltal minden végrehajtási szál számára azért annyit garantál, hogy **esetlegesen** azonos értékeket olvassanak, ha az írási műveleteket felfüggesztjük.
Azt az időintervallumot amire szükség van az írási műveletek végrehajtásától addig a pontig amíg az összes replikára hatással lesz az írási művelet **inkonzisztencia ablak**nak nevezzük. Az ablak méretét sok tényező befolyásolja. Ilyenek például a kommunikációs késleltetések, a különböző hibák, a rendszer terhelése, a hálózat részekre szakadása, illetve az a jelenség, hogyha csúcsok sűrűn csatlakoznak és kapcsolódnak le a rendszerről (churn).



4. ábra - Korrekt eredmények az esetleges konzisztencia modell alatt

Kauzális+ konzisztencia modell

A kauzális konzisztecia problémáját már a korábbiakban szemléltettem. A replikák akár örökké is széttarthatnak és sose érnek el egy konzisztens állapotot, amennyiben konkurens írásokat is végrehajtunk több különböző csúcson. Ezeknek a hatásai a Lamport féle rendezés definíciójából következően különböző sorrendben is érvényesülhetnek a különböző csúcsokban. Ennek a problémának az orvoslására használhatunk konvergens konfliktuskezelést. Ennek az egyik egyszerű megvalósítása regiszterek esetén az, hogy az utolsó írást tekintjük érvényesnek. Az utolsó írás által beírt tartalmat replikáljuk. Ez felveti az adatbázisok keretében már sokat taglalt problémát, hogy végülis melyik lesz az utolsó írás. (Órák szinkronizálásának problémája, verziózás…)
Egy másik nagy előnye ennek a modellnek, hogy bonyolultabb adat objektumok esetén szemantikus konfliktus feloldást is lehetővé tesz. Ennek megfelelően felismerhetők a kommutatív műveletek és a hatásaikat összefésülve alkalmazhatjuk a csúcsokban.

Tranzakciós konzisztencia modellek

A korábbi, nem tranzakciós konzisztencia modellek csupán a műveletek izolációjával foglalkoztak. Itt már lesznek tranzakcióink, amely műveletek sorozata, amelyek közül vagy mind hatásos, vagy egyik sem. [1]

Sorosítható

A sorosíthatóságról adatbázisokból szintén tanultuk. Georeplikált környezetben ez egyszerűen nem fog menni.
Túl sok együttműködést igényelne a csúcspontoktól, ez rengeteg kommunikációval járnak, ami pedig nagyon költséges.

Snapshot Isolation

A Snapshot Izoláció konzisztencia modell jól elterjedt. Az Oracle és a SQLServer is megvalósítja. A sorosítható enyhítéseként kapjuk.
Az elképzelés alapja az, hogy minden egyes tranzakció egy korrekt állapotból, egy pillanatképből indul. A tranzakció lefutása után egy másik korrekt állapotba kerülünk. Ez azt eredményezi, hogy tranzakció nem függhet „félig lefuttott" tranzakciók által írt „piszkos" adatoktól.
Hogy jobban megértsük a snapshot izolációt vegyük a kovetkező példát:

Két konkurens tranzakció egy metszettel rendelkező adathalmazt olvas be.
A beolvasás után a tranzakciók a beolvasott adatoknak két diszjunk halmazát módosítják, majd ezt írják vissza.

A két tranzakció nem fog tudni a másik tranzakcióban történt módosításokról.
Ez egy jellemző anomália a snapshot izolációnál, „write skew"-nak szokás nevezni. Ez is mutatja, hogy a Snapshot Isolation gyengébb kritérium, mint a sorosíthatóság, ott ez nem történhetne meg konkurens módon.

Parallel Snaphot Isolation

Annak ellenére, hogy gyengébb mint a sorosíthatóság konzisztencia modell, a Snapshot Izolációnál is az összes írás-írás konfliktust fel kell deríteni. Elosztott esetben ez egy költséges feladat. Hát még georeplikált esetben, amikor az együttműködést a magas költségű kommunikáció nagyon költségessé teszi. Nem régiben bemutatásra került ennek a modellnek egy módosított változata. Ez a modell a snapshot izolációs modell enyhített változata, amely direktbe a georeplikált rendszereket célozza meg. Ez lehetővé teszi azt, hogy a különböző replikákra a tranzakciók különböző sorrendbe fejthessék ki a hatásukat.
Adott csúcson belül is futhatnak konkurens tranzakciók. Ezeket a snapshot izoláció szerint kezeljük.
Különböző csúcsokon is futhatnak konkurens tranzakciók. Ezeket enyhébb módon kezeljük.
Ez úgy működik, hogy egy csúcspontban a snapshot izolációnak megfelelő működés megy végbe, viszont a különböző csúcsok között detektált írás-írás konfliktusok hatására csak akkor abortáljuk a tranzakciót ha még az nem zárult le. Ha már mindkét csúcsban végrehajtott tranzakció lezárult, akkor a tranzakciók hatásait valamilyen előre meghatározott sorrendben alkalmazzuk az összes replikára.

Forrás: [5][8]

Adat replikációs stratégiák

Aktív replikáció

Aktív replikációnál arról kell gondoskodnunk, hogy a műveleteket az összes replika végrehajthassa egy meghatározott sorrendben és biztosan. Ennek megfelően az összes replika szinkronban tud maradni. Ennek megvalósításához egy előre determinált sorrend szükséges, szinkron végrehajtással. Biztosítanunk kell egy atomi üzenetszórást megvalósító módszert. Az üzeneteket minden csúcspontnak meg kell kapnia, ráadásul biztosítani kell azt is, hogy mind azonos sorrendben kapják meg ezeket, így azonos sorrendben hajthassák végre a konzisztencia fenntartásához szükséges műveleteket.
Itt nem szükséges az, hogy a módosításokat egyetlen replikán végezzük végig.
A másik két módszernél ez így van.
A lekapcsolódó és újrakapcsolódó csúcsoknak a lekapcsolt időben történt összes műveletet el kell juttatni, vagy más módon konzisztens állapotba kell hozni.

Elsődleges példány

Az elsődleges példány replikációval kapcsolatban már számos dolgot tanultunk adatbázisokból.

Multimaster

Itt a konzisztencia fenntartásához még több szinkronizációs üzenetre van szükség. Ez túl költséges lehet.

Konkrét implementációk

COPS

A COPS egy adattárház, amit arra terveztek, hogy a kauzális+ konzisztencia garanciát nyújta úgy, hogy jól skálázható legyen nagy területekre is - „WAN”. A korábban már említett konzisztencia kritériumokat úgy tudja biztosítani, hogy bevezeti a *függőségeket.* Az implementáció függőségeket definiál az egyes műveletek között. Ha egy adott műveletnek egy függősége nem teljesül, ami egy másik művelet, akkor azt addig nem hajtja végre, amíg a korábbi végre nem hajtódik. Ezek a függőségek a kliens által küldött írási, olvasási kérésekben szerepelnek.
A készítők azt vallják, hogy ez egy jól skálázható megoldás. Sajnos a gyakorlatban egy georeplikált környezetben nem vált be kellőképpen.
Egy másik újdonsága a COPS-nak a get tranzakciók használata.
Ezek a műveletek lehetővé teszi a kliens számára, hogy kulcsok egy halmazát kiolvashassák úgy, hogy előtte még a függőségeket is ellenőrzi a rendszer. Csak akkor adja vissza az eredményhalmazt ha valóban minden függőség teljesült.
De vegyünk egy példát, hogy lássuk miért is olyan hasznos ez a funkció.
- Tegyük fel, hogy két írásműveletet végzünk. Írjuk A-t és aztán írjuk B-t. Előfordulhat, hogy a „véletlenek” játékának köszönhetően a B-n végzett módosításunk gyorsabban terjed a replikák között mint az A-n végzett módosítás. Ennek következtében előfordulhat az, hogy ha A-t és B-t olvassuk valamely ilyen replikából, akkor B-ból az új és A-ból a régi változatot olvashatjuk ki. Ennek elkerülése érdekében használhatjuk a get tranzakciókat. Ennek a segítségével elérhető az, hogy ha két ilyen olvasást végzünk akkor ne kaphassuk vissza A-ról a régi képet ha már B-ről az új van, habár ez megfelelnek a Lamport féle események „korábban történt” definíciójának a részleges rendezésének, de ez bizonyos alkalmazások esetén nem kielégítő.
A konzisztencia garanciák és a skálázhatóság a függőségek alkalmazásának köszönhető. Viszont fontoljuk meg azt is, hogy mivel a függőségek a kliensnél tárolódnak és nem a rendszer kezeli őket, így ez problémákat jelenthet. Például vegyük azt az esetet, hogy egy kliensnek elképzelhető, hogy hatalmas mennyiségű függőséget kell tárolnia – amire esetleg nem lesz alkalmas. A tárolandó függőségek száma főleg a get tranzaciók során növekszik meg.

Walter

Ez egy egészen friss rendszer. Geo-replikált környezetre tervezett, amely lényegében egy key-value store, amely tranzakció kezeléssel is ellátott. A parallel snapshot isolation-t valósítja meg. Ez lehetővé teszi, hogy a tranzakciók hatásait aszinkron módon terjesszék a csúcspontok egymás között miután commitált egy központi szervernél a csúcspontban. A tranzakciók rendezését egy vektor időbélyeg oldja meg, amelyet a kezdetekor map meg, amely tartalmaz egy bejegyzést is az adott csúcspontra vonatkozólag. Minden vektor időbélyeg ezáltal meghatároz egy snapshotot a rendszerben. Emlékezzünk arra, hogy a PSI-nél fontos az is, hogy épp melyik csúcsról van szó, mivel a különböző csúcspontokban a snapshotok eltérhetnek, de ez a vektor tartalmazza az erre vonatkozó információt is.
Két további újjítást is tartalmaz a Walter. Ezek a PSI implementációval kapcsolatosak: a „preferred site” és a „counting set”.
Minden egyes objektumhoz tartozik egy preferred site, ami azt jelzi, hogy az objektum tulajdonosához melyik adattárház van a legközelebb. Ez lehetővé teszi azt, hogy a tranzakciók, amelyek lokálisak erre a csúcsra nézve, azokat hatékonyabb módon commmitálhassuk, egy úgynevezett Fast Commmit protocol segítségével.
Ez a protocol lehetővé teszi, hogy a preferred site-on a többi csúccsal való egyeztetés, koordináció nélkül is commmitálhassunk. Ez nyílván gyorsabb lesz mintha együttműködésben kellene ezt, koordinálva megtenni. A gyors commit során két dolgot ellenőriz a Walter:
- az érintett objektumokat nem módosították azóta amióta a tranzakció elkezdődött
- az érintett objektum nincs zárolva írásra.

Ha nem lokális a tranzakció ilyen értelemben, azaz olyan adatelemek módosítását is igényli a lezárása, amelyeknek a preferred site-ja nem a helyi csúcs, akkor a slow commit protocol végrehajtására lesz szükség. Ez a protokol egy két fázisú commit protokoll, amely a preferred site-okra terjed ki. Azaz zárolnia kell minden ilyen csúcson az adatelemet. (Ez a georeplikált környezet miatt költséges és lassú.)
A „counting set” egy új adattípus. Hasonló a kommutatív replikált adattípusokhoz. Ezeknek a használata lehetővé teszi a Fast Commit használatát, mivel kommutatívak a műveletei.
A Walter-t arra tervezték, hogy a PSI konzisztencia modell garanciát skálázható módon nyújtsa egy georeplikált elrendezésben is. De az a tény, hogy a tranzakciókat minden egyet csúcsban egy központi

szerver hajtja végre és commitálja az bizony egy szűk keresztmetszet lehet a teljesítmény számára. A másik hátulütője a slow commit protocol használata. Amennyiben nem helyi tranzakciókat is gyakran futtatunk, akkor a ez meghatározó lehet az áteresztőképesség és a skálázhatóság szempontjából

További források: [7][8]

Yahoo – PNUTS

A PNUTS egy nagy méretre szabott adat tároló rendszer, amit a Yahoo fejlesztett ki, hogy tárolási lehetőséget nyújtson a földrajzilag elosztott módon működő alkalmazások számára. Ez a rendszer az erős és az esetleges konzisztencia modellek „közé lő" azáltal, hogy a rekordonkénti idővonal konzisztencia modell megvalósítására vállalkozik. Ez a konzisztencia modell lehetővé teszi, hogy némi garanciát vállaljon a rendszer, de mégis jó teljesítményt biztosíthasson, és jól skálázható maradjon.
A rendszer rekoldokkal dolgozik, mint az alapvető adattárolási egység, amelyek egy közös adatbázis tábla soraiból kerülnek ki.
Ebben az implementációban minden egyes rekordhoz tartozik egy elsődleges példány, amelyhez a frissítési utasítások route-olódnak.
Az elsődleges példány a következőképpen kerül kiválasztásra:
- Ha egy rekord egy adott földrajzi területről kapja a legtöbb frissítést, akkor az ahhoz legközelebbi lesz az elsődleges példány.

Ez a megközelítés lehetővé teszi azt, hogy egy adaptív módon választhassuk ki az elsődleges példányt, alkalmazkodva az esetleges terhelésváltozásokhoz.
A PNUTS egy API-val szolgál számunkra, amely segítségével a konzisztencia különböző szintjeit állíthatjuk be.
A PNUTS jelenlegi változata nem használható olyan esetben, ha a hálózat esetleges szétesésével kell számolni. Ilyen esetben kettő lehetőség van annak a hálózatrésznek a számára, ahol egy adott rekordhoz tartozó elsődleges példány nem érhető el:
1; Nem hajthatunk végre írás műveleteket az adott rekordon vagy
2; Választunk egy új elsődleges példányt.
Az első lehetőségnek a nyílvánvaló hátrányait nem részletezném. A második lehetőségnek az a fő problémája, hogy ha a hálózat újra helyreáll és megszűnik a darabokra szakadás, akkor „hirtelen" több elsődleges példány fog rendelkezésre állni, amelyeken módosításokat hajthattak végre, így azok eltérnek, nem konzisztensek.

Amazon - Dynamo

A PNUTS egy nagy méretre szabott adat tároló rendszer, amit a Yahoo fejlesztett ki, hogy tárolási lehetőséget nyújtson a földrajzilag elosztott módon működő alkalmazások számára. Ez a rendszer az erős és az esetleges konzisztencia modellek „közé lő" azáltal, hogy a rekordonkénti idővonal konzisztencia modell megvalósítására vállalkozik. Ez a konzisztencia modell lehetővé teszi, hogy némi garanciát vállaljon a rendszer, de mégis jó teljesítményt biztosíthasson és jól skálázható maradjon.
A rendszer rekoldokkal dolgozik, mint az alapvető adattárolási egység, amelyek egy közös adatbázis tábla soraiból kerülnek ki.
Ebben az implementációban minden egyes rekordhoz tartozik egy elsődleges példány amelyhez a frissítési utasítások route-olódnak.
Az elsődleges példány a következőképpen kerül kiválasztásra:
- Ha egy rekord egy adott földrajzi területről kapja a legtöbb frissítést, akkor az ahhoz legközelebbi lesz az elsődleges példány.

Ez a megközelítés lehetővé teszi azt, hogy egy adaptív módon választhassuk ki az elsődleges példányt, alkalmazkodva az esetleges terhelésváltozásokhoz.
A PNUTS egy API-val szolgál számunkra, amely segítségével a konzisztencia különböző szintjeit állíthatjuk be.
A PNUTS jelenlegi változata nem használható olyan esetben, ha a hálózat esetleges szétesésével kell számolni. Ilyen esetben kettő lehetőség van annak a hálózatrésznek a számára, ahol egy adott rekordhoz tartozó elsődleges példány nem érhető el:
1; Nem hajthatunk végre írás műveleteket az adott rekordon vagy
2; Választunk egy új elsődleges példányt.
Az első lehetőségnek a nyílvánvaló hátrányait nem részletezném. A második lehetőségnek az a fő problémája, hogy ha a hálózat újra helyreáll és megszűnik a darabokra szakadás, akkor „hirtelen" több elsődleges példány fog rendelkezésre állni, amelyeken módosításokat hajthattak végre, így azok eltérnek, nem konzisztensek.

Forrás: [5]

Összefoglalás

A tanulmányban megismerhettük különféle adat konzisztencia modelleket és a meghatározásuk mögött rejlő megfontolásokat. Georeplikált környezetben szükséges „trade off”-ok okát és ezek következményeit. A konkrét implementációk megismerésével jól láthattuk, hogy milyen lehetőségeink is vannak az egyes modelleket választva. Ez a terület egy ma is nagyon aktív kutatási terület. Számos eredmény várható a közeljövőben is.
További implemetációkról még olvashatunk a sokat emlegetett tanulmányban. [5]

Irodalomjegyzék

[1] Gajdos Sándor: Adatbázisok, harmadik kiadás, 2006.

[2] http://www.db-book.com/ oldalon található idevágó, 17. Database System Architectures Feb 1, 2010, 18. Parallel Databases Feb 1, 2010, 19. Distributed Databases diasorok. 2012.10.07-én elérhető változatai szerzők: Avi Silberschatz, Henry F. Korth, S. Sudarshan

[3] Információs rendszerek üzemeltetése – 2011/2012 tavaszi félév jegyzetei

[4] IT labor 1. – MIT 1-es mérés segédanyagai. www.inf.mit.bme.hu

[5] Geo-Replication in Large-Scale Cloud Computing Applications, szerző: Sergio Garrau, Konzulens: Professor Luis Rodrigues URL: http://www.gsd.inesc-id.pt/~ler/reports/sergioalmeida-midterm.pdf, 2012.10.07

[6] Nicola Santoro: DESIGN AND ANALYSIS OF DISTRIBUTED ALGORITHMS, ISBN-13: 978-0-471-71997-7, 2007

[7] Marcos K. Aguilera: Transactional storage for geo-replicated systems (Prezentáció) http://cloud.berkeley.edu/data/walter.pptx (2012.10.10)

[8] Yair Sovran* Russell Power* Marcos K. Aguilera Jinyang Li: Transactional storage for geo-replicated systems http://research.microsoft.com/en-us/people/aguilera/walter-sosp2011.pdf (2012.10.10)