Peer-to-peer alapú elosztott fájlrendszerek

VINCZE GÁBOR, PAP ZOLTÁN, HORVÁTH RÓBERT

Budapesti Műszaki és Gazdaságtudományi Egyetem, Távközlési és Médiainformatikai Tanszék
{vincze,pap,horvath.r}@tmit.bme.hu

Kulcsszavak: végpontok közti összeköttetések, fájlcserélő hálózatok, hierarchikus hálózat, tartalom-alapú címzés

A peer-to-peer hálózatok egyre több és fejlettebb szolgáltatást nyújtanak, ám a hagyományos hálózati fájlrendszerek szolgáltatásainak mennyiségét, minőségét és megbízhatóságát a mai napig nem sikerült elérni. Bemutatjuk a peer-to-peer hálózatok fejlődését, valamint azokat a nehézségeket, amelyek eddig meggátolták a fájljellegű szolgáltatások nyújtását. Ismertetjük a CFS-t és az Ivy-t, a két DHT alapú elosztott fájlrendszert, amelyek azonban nem adnak választ a kulcsproblémákra, és végül felvázoljuk, milyen megoldások várhatóak a jövőben.

1. Bevezetés

A peer-to-peer (továbbiakban: p2p) számítástechnika célja egy autonóm, önkonfiguráló és hibatűrő hálózat létrehozása, amely úgy viselkedik, mint egyetlen hatalmas számítógép. Egy p2p hálózatban, a klasszikus számítógépes és távközlési hálózatokkal szemben, nincsenek központi szerverek vagy csomópontok, amelyek a hálózati funkciókat koordinálnák – ehelyett a hálózati csomópontok kooperálnak, és mindegyikük a képességeinek (erőforrások, hálózati csatlakozás minősége) megfelelő mértékben igyekszik a hálózatmenedzsmentben részt venni. A gyakorlatban a p2p hálózatokat arra használják, hogy az Internet peremein található erőforrásokat összefogják egyetlen feladatra, amely éppúgy lehet adattárolás, mint egy rendkívül számításigényes feladat. Ugyanis az egyenként kis teljesítményű otthoni számítógépek kombinált tároló- és számítási kapacitása együttesen messze meghaladja bármely szuperszámítógépét.

Fejlődésük során a p2p hálózatok egyre ambiciózusabbakká váltak az általuk nyújtott szolgáltatások terén, és egyre több és jobb minőségű alkalmazást tettek lehetővé. Ebben a cikkben ezeknek a szolgáltatásoknak egy típusát, az adattárolást vizsgáljuk meg. Először bemutatjuk a fájlcserélő-p2p hálózatok fejlődését, majd válasz keresünk arra, miképpen lehetne elérni egy teljesen elosztott, p2p alapú fájlrendszer megvalósítását, valamint áttekintjük azokat a problémákat, melyek eddig meggátolták ezt.

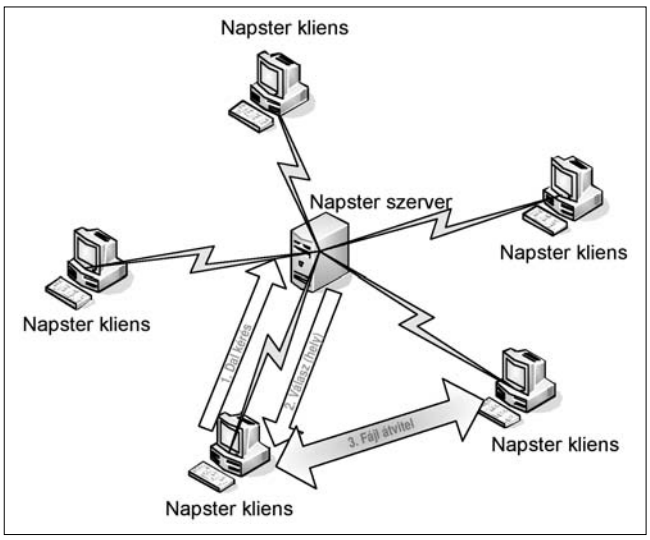
2. Első és második generációs peer-to-peer hálózatok

Bár a p2p számítástechnika koncepciója már az ARPANET hálózattal feltűnt (sőt, fejlesztésének egyik fő motivációja volt), a fogalmat ma inkább az újabb, fájlcserélő hálózatokra alkalmazzák. Fejlődésük során a p2p hálózatok felépítése lényegi változásokon ment keresztül, de többé-kevésbé valamennyi ugyanazokkal a motiváló erőkkel és tulajdonságokkal rendelkezik. A hálózatok szolgáltatásainak természete miatt – zenék vagy filmek cseréje, amelyek gyakran illegálisak – fontos volt, hogy a felhasználók megőrizhessék névtelenségüket. Ugyanezen okok miatt a legtöbb p2p hálózat nem koordinált fejlesztési projektként, hanem széles rétegek által használt alkalmazásként indult. A fél- vagy teljesen illegális jelleg miatt egy központi szerver nem csupán megbízhatósági szempontból kritikus elem, hanem tökéletes célpontot nyújt a (jogi) támadások számára is. A felhasználók száma rendszerint igen nagy, ám ezek a felhasználók csak ritkán és rövid időre csatlakoznak, és amint megkapták, ami számukra szükséges volt, kilépnek a hálózatból. Így bármely p2p hálózatnak, amely működőképes szolgáltatásokat akart nyújtani, hamar megoldást kellett találnia a hibatűrés és az adatok redundáns tárolásának kérdésére.

2.1. Napster

Az első masszívan népszerű p2p rendszer, amely a p2p-számítástechnikát behozta a köztudatba, az 1999. őszén megjelent Napster volt. Bár már ezelőtt is léteztek módszerek a zenefájlok cseréjére, a Napster volt az első, kifejezetten mp3-cserélésre specializálódott hálózat. A Napster megjelenése szinte azonnal magára vonta a nagy kiadóvállalatok figyelmét, és 1999. decemberében keresetet nyújtottak be ellene. A per hatalmas reklámot jelentett a Napster számára és rendkívül népszerűvé tette. A felhasználók száma 2001 februárjában érte el csúcsát, 13,6 millió taggal.

A Napster nem volt „valódi” p2p hálózat: a felhasználók által megosztott összes fájl listáját egy központi adatbázisban tárolta. Így a hálózati útvonalválasztás központosított módon történt: amikor egy felhasználó elindított egy keresést, elküldte a kérését a központi szervereknek, ahonnan megkapta a keresett fájl helyét. Miután megtalálta a fájlt, az átvitel már közvetlenül a felhasználók között – az *1. ábrán* láthatóan,– p2p-módon történt.



1. ábra Napster routing és adatátvitel

Ez a tervezési hiányosság jelentette végül a Napster végzetét: 2001. júliusában egy bíró elrendelte a központi szerverek leállítását. Ám a Napster megnyitotta az utat az őt követő számos p2p hálózat számára.

2.2. SETI@home

A SETI@home nem fájlcserélő hálózat, és itt csak azért említjük meg, hogy megmutassuk, a p2p architektúra könnyen alkalmazható más feladatokra is. Röviddel a Napster előtt indult 1999. májusában, és a mai napig ez a legsikeresebb elosztott számítási hálózati megoldás, több mint 5,3 millió felhasználóval és 2,2 millió évnyi aggregált CPU idővel (2005. januári adatok).

A projekt célja a Puerto Ricói Arecibo rádióteleszkóp által gyűjtött adatok elemzése, földön kívüli intelligencia jelei után kutatva. Az adatok elemzése a projekt anyagi lehetőségeit messze meghaladó számítási teljesítményt igényelne. Két év alatt a rádióteleszkóp háromszor pásztázta végig az ég általa látható részét, naponta kb. 35 gigabájtnyi adatot termelve. Az adatokat hagyományos postai úton, egy DLT kazettán küldik el a Berkeleyben telepített központba. Itt az adatokat 0,25 megabájtos darabokra szabdalják, amelyeket aztán elküldenek a klienseknek elemzésre. A kliensprogram képernyővédőként, vagy alacsony prioritású processzként működik, és szinte az összes népszerű operációs rendszerre elérhető. Az elemzés során három fő vizsgálatot végez el:

– Gaussi-emelkedéseket és eséseket keres az adási teljesítményben, amely azt jelezné, hogy az antenna egy rádióforrás felett haladt el,
– impulzusokat keres, amelyek keskenysávú, digitális jellegű átvitelt jelezhetnek,
– impulzus-hármasokat keres.

Az adatcsomagokat egyszerre több kliensnek is elküldik, és hibadetektáló algoritmusokat építettek a számítási algoritmusba a számítási hibák, vagy a szándékosan elküldött hibás eredmények kiszűrésére. A központi szerverekkel rendelkező, félig p2p rendszerek architekturális gyengesége itt is megmutatkozott, amikor ez év januárjában egy építkezés miatti hosszabb áramkimaradás miatt az összes szervert le kellett állítani.

2.3. Gnutella

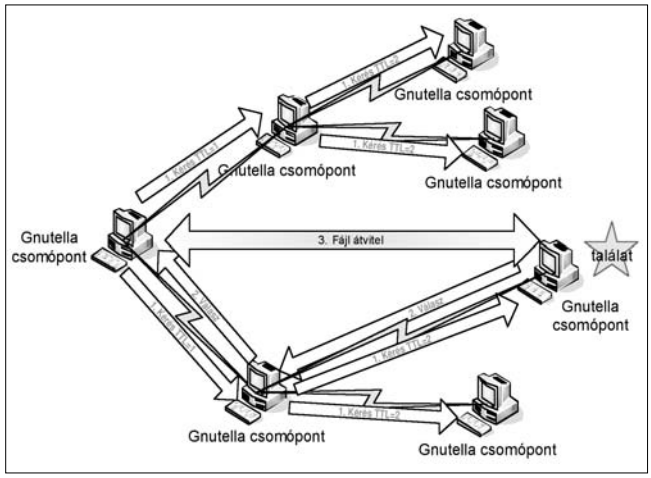
A Napster első utódja, a Gnutella az AOL-hálózaton belül működő Nullsoft által fejlesztett projektként indult. 2000. március 14-én a cég honlapjára helyezték a szoftvert letöltésre, azonban az AOL, jogi bonyodalmaktól tartva a következő napon levetette onnan, és a projektet leállította. Ám az esemény megjelent a Slashdot hírportálon, és ez az egyetlen nap elég volt hozzá, hogy több ezren letöltsék a programot. A protokollt ezek után visszafejtették, és több nyílt forráskódú kliens is megjelent (mint a LimeWire, a BearShare, vagy a Gnucleus).

A Napsterrel ellentétben egy Gnutella hálózat tökéletesen decentralizált, nem támaszkodik központi szerverekre, amelyek kritikus meghibásodási pontok, vagy támadási célpontok lehetnének.

Amikor egy A csomópont inicializálja magát, megtalál legalább egy másik B csomópontot valamilyen sávon kívüli módszerrel (például a szoftverrel előre csomagolt csomópontlista alapján, egy website segítségével, vagy IRC-n keresztül). Ezek után az A csomópont megkapja a B csomóponttól az összes általa ismert működő csomópont listáját, és megpróbál azokhoz is csatlakozni. Ez az iteratív folyamat addig folytatódik, amíg az A csomópont nem csatlakozott egy meghatározott számú (általában a felhasználó által megadható, tipikusan öt körüli) másik csomóponthoz. Az A csomópont megtartja az összes még ki nem próbált csomópont listáját, viszont törli azokat a csomópontokat a listából, melyek nem működtek. Csatlakozás után rendszeres időközönként ellenőrzi egy ping üzenettel, hogy a szomszédjai még mindig csatlakoznak-e hozzá.

Amikor a felhasználó le szeretne tölteni egy fájlt, az A csomópont elküldi a kérést az összes szomszédjának. Ha ezek közül egynek sincsen meg a keresett fájl, továbbküldik szomszédjaiknak, és így tovább. Ilyen módon a kérés elméletileg előbb-utóbb eljut az összes csomóponthoz a hálózatban, amint az a *2. ábrán* látható. A hálózat elhagyásakor az A csomópont elmenti a csomópontlistáját a következő csatlakozáshoz.

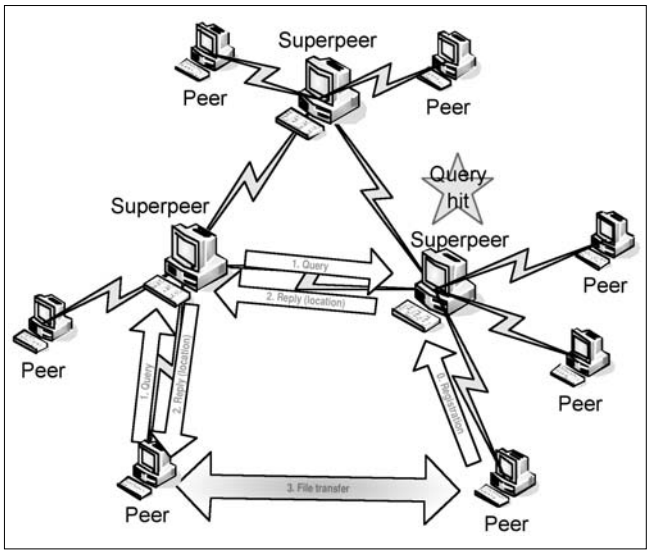
2. ábra Gnutella routing és adatátvitel



A gyakorlatban a Gnutella hálózatok nem voltak képesek megbirkózni a Napster leállítása után hozzájuk özönlő felhasználókkal. A legfőbb problémát az állandóan csatlakozó és kilépő felhasználók által okozott hálózati instabilitás, valamint az elárasztásos kérések jelentik, amelyek nem működnek sok csomópontot tartalmazó hálózatok esetén. Ezen felül ez a keresés nem garantálja, hogy valóban megtalálunk egy olyan fájlt, amely a hálózatban van.

A FastTrack a Gnutella kiterjesztése, amely hierarchikus hálózatok kiépítését teszi lehetővé a skálázhatóság és a stabilitás növelésének érdekében. Egy gyors számítógép jó hálózati kapcsolattal automatikusan szuper-csomóponttá válik. A közönséges kliensek a megosztott fájlok listáit, valamint a kéréseiket a szuper-csomópontoknak küldik, és azok utána egymás közt továbbítják a kéréseket. A nagyméretű fájlokat egyszerre több helyről is le lehet tölteni az UUHash algoritmus segítségével. A FastTrack hálózatok működését a *3. ábra* mutatja.

3. ábra FastTrack routing és adatátvitel



2.4. BitTorrent és eXeem

A BitTorrent egy decentralizált letöltést lehetővé tevő eszköz, amely áthidalja a letöltések forrásánál keletkező szűk keresztmetszetből adódó nehézségeket. Eredetileg linux-disztribúciók letöltéséhez használták, de mára a filmletöltés is jelentős alkalmazássá vált.

Egy fájl letöltéséhez a felhasználónak először is szüksége van egy .torrent fájlra, ami egyrészt tartalmazza egy nyomkövető szerver címét, valamint a letöltendő fájl minden adatblokkjából képzet hash-értéket. Amikor a felhasználó megkezdi a letöltést, kapcsolatba lép a nyomkövető szerverrel, amely átirányítja kérését olyan felhasználókhoz, akik már rendelkeznek a letöltendő fájl egyes adatblokkjaival. Amint rendelkeznek néhány adatblokkal, a többi letöltő elkezdi ezeket letölteni tőlük.

A BitTorrent nem nyújt keresési/útvonal-választási képességeket, és a .torrent fájlokat felkínáló websiteok, valamint a nyomkövető szerverek a hálózat könnyen támadható pontjai. Az eXeem egy BitTorrentre épülő p2p alkalmazás, amelyben minden letöltő egyben nyomkövető szerverré is válik.

Az első és második generációs p2p hálózatoknak, bár többé-kevésbé működő rendszerek megvalósítását tették lehetővé, nem sikerült megoldaniuk az alapvető p2p-célkitűzéseket. A Napster és SETI@home központi szerverekre támaszkodnak, így nem is tekinthetők valódi p2p hálózatoknak, és a központi szerverek leállásai, az ellenük intézett támadások megmutatták, hogy ez nem pusztán elméleti jelentőségű probléma. Az őket követő második generációs hálózatok már központi szerverek nélkül működnek, ám a keresés/útvonalválasztás megvalósítása nem kellőképpen átgondolt, így a hálózatok nem skálázhatóak, és egy bizonyos méret felett működésképtelenek. Ezen felül a központi szerver megszűnése még a keresési funkció elveszését is jelentette. Ezekre a problémákra a p2p hálózatok harmadik generációja próbál megoldásokat találni.

3. DHT-alapú peer-to-peer hálózatok

A p2p hálózatok harmadik generációja, amelynek tagjai implicit vagy explicit módon az elosztott hash-táblákon (Distributed Hash Tables, DHT) alapulnak, az előző rendszerek gyenge pontjait igyekeznek kiküszöbölni, és olyan skálázható, megbízható hálózatokat kialakítani, ahol egy a hálózatban lévő fájl, központi szerverek alkalmazása nélkül is garantáltan megtalálható.

Ezt úgy érik el, hogy értékeket (a hálózatban elhelyezett hasznos tartalmat) kulcsokra képeznek le valamilyen hash-algoritmus segítségével (a hash algoritmus megválasztása tulajdonképpen nem lényeges kérdés a hálózati architektúra szempontjából). Egy jó hash-algoritmus olyan kulcsokat ad a bemenetként kapott értékekre, amelyek a bemenetek eloszlásától függetlenül a kulcstérben egyenletesen oszlanak el.

Az érték→kulcs leképzés után minden csomópont a kulcstér egy bizonyos tartományáért felelős, és tartalmazzák vagy a pointereket, vagy magát a hasznos tartalmat (általában az adattárolás és -átvitel kérdésével a DHT algoritmusok nem foglalkoznak, csupán az elosztott útvonalválasztás problémájára próbálnak megoldást adni). A különböző DHT-alapú p2p hálózatok a kulcstér leképzésében valamilyen absztrakt topológiára (általában gyűrű vagy hiperkocka), valamint az útvonalválasztási algoritmusában térnek el egymástól.

A négy legismertebb DHT-implementációból (Chord, CAN, Pastry, Tapestry) itt kettőt mutatunk be.

3.1. Chord

A Chord hálózatban [1] minden csomóponthoz egy *m* bites csomópontazonosítót rendelünk, általában a csomópont IP címének hash-leképzésével, az SHA-1 algoritmus segítségével (ami egy 160 bites azonosítót adna). A csomópontokat egy gyűrűbe rendezzük, mely lefedi a teljes azonosító-teret. Az értékeket szintén

hasheljük, hogy *m* bites kulcsokat kapjunk. Ezek után minden kulcsot azon a csomóponton tárolunk, amelynek azonosítója megegyezik vagy követi azt a kulcstérben. Ezt a csomópontot a kulcs *utódjának* (successor) nevezzük.

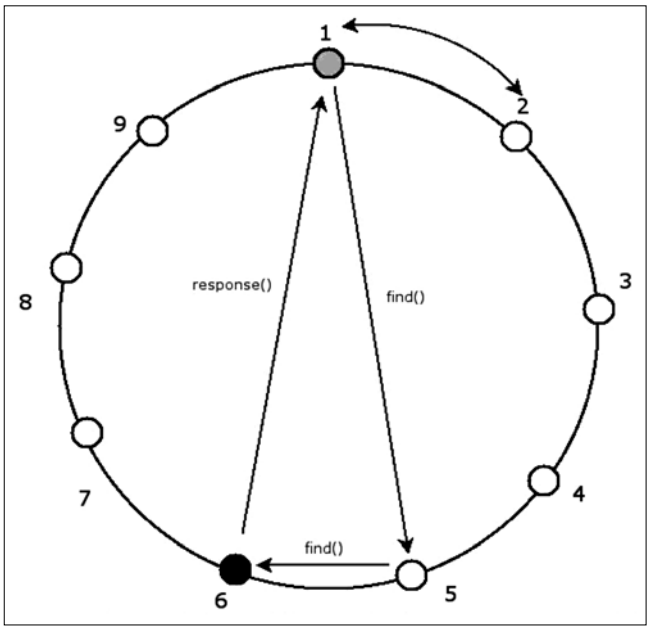
Az útvonal-választási információk egy mutatótábla (finger table) formájában találhatók meg minden csomóponton, amelynek *m* eleme van, és az *i.* elem *utód(n +2^(i-1))* (ahol *n* a csomópont azonosítója). Hasonló módon a csomópont *elődje* az azonosító-gyűrűn az őt közvetlenül megelőző csomópontot jelöli. Így minden csomópont csak néhány másik csomópontról tárol információt, és az információ részletessége csökken az azonosító-térbeli távolsággal.

Amikor egy csomópont megkap egy kérést *k* kulcs megtalálására, vagy ismeri *utód(k)*-t (a mutatótáblájában van), és ebben az esetben továbbítja a kérést annak a csomópontnak, vagy továbbítja a mutatótáblájában a *k*- legközelebbről megelőző csomóponthoz, amelynek több lokális információja van az azonosító-tér *k*-t tartalmazó tartományáról. Íly módon egy kérés előbb-utóbb *(O(log(n))*-lépésben) eléri a kucsért felelős csomópontot.

Amikor egy új csomópont lép be a hálózatba, inicializálja az elődjét és a mutatótábláját. A meglévő csomópontok mutatótábláit és elődjeit szintén frissítjük. Azokat a kulcsokat, amelyekért az új csomópont felelőssé vált, ezek után átmásoljuk rá. Minden új csomópont belépésekor mindössze *O(1/N)* kulcs/érték párt kell új helyre másolni.

A csomópontok meghibásodását a minden csomóponton futó stabilizáló rutin kezeli, amely egyrészt időnként ellenőrzi, hogy beléptek-e új csomópontok a csomópont és közvetlen szomszédjai közé, valamint detektálja a meghibásodott csomópontokat. Az adatok redundáns tárolását úgy oldhatjuk meg, hogy minden kulcsról tárolunk egy másolatot az érte felelős csomópont *r* utódján is.

4. ábra Chord gyűrű és keresés



3.2. CAN

Egy CAN-ben [2] (Content Addressable Network – tartalom-címezhető hálózat) az azonosítóteret egy *d*-dimenziós koordinátatérre képezzük le, amelyet dinamikusan particionálunk a csomópontok között. Minden kulcsot egy *P* pontra képezünk le ebben a koordinátatérben. Ezek után minden kulcsot a koordinátatérnek *P*-t tartalmazó részéért felelős csomóponton tárolunk.

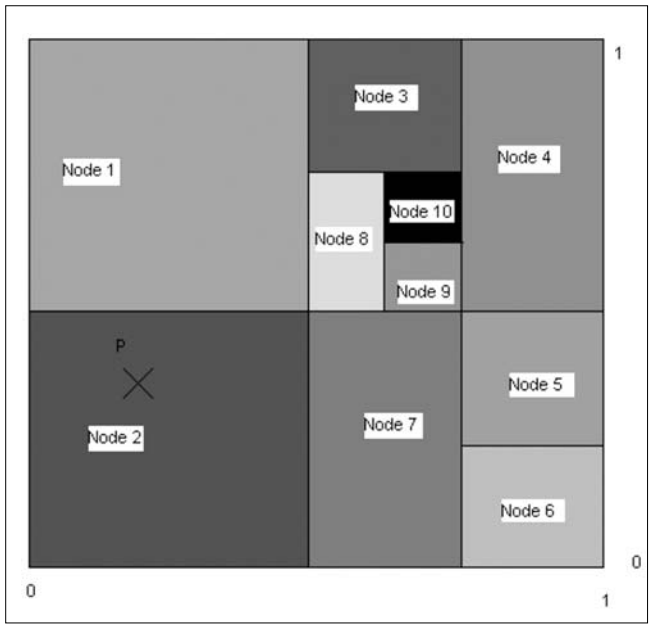
Minden csomópont számon tartja a szomszédjait, azaz azokat a csomópontokat, amelyek osztoznak vele a koordinátatérbeli zónájának valamely határán. A kéréseket a csomópontok mohón a *P*-hez legközelebbi szomszédjuk fele továbbítják. Amennyiben valamilyen oknál fogva ez a szomszéd meghibásodott volna, egy növekvő gyűrűs keresést folytatunk, amíg nem találunk egy megfelelő csomópontot, és a normális útvonalválasztás ettől fogva helyreáll.

Egy *d*-dimenziós koordinátatérben, amelyet *N* csomópont között osztunk fel, a kérések célba éréséig a teljes úthossz *d/4*N^(1/d)*. Így *d* növelésével csökkentjük az útvonalhosszt, de növeljük az útvonal-választási tábla méretét.

Amikor egy új csomópont belép a hálózatba, meg kell találnia valamilyen sávon kívüli módszerrel legalább egy csomópontot, amely már a CAN része. Az új csomóponthoz egy véletlen *P* pontot rendelünk a koordinátatérben. A CAN útvonal-választás segítségével a csomópont megkeresi a pontot tartalmazó zónáért felelős csomópontot, és ezek után a zónát kettéosztjuk a már meglévő és az új csomópont között. A zónával szomszédos csomópontokat ezek után értesítjük a kettéosztásról. A Chord-hoz hasonló módon itt is minden csomópont rendszeresen ellenőrzi szomszédjait, hogy még mindig a hálózatban vannak-e.

Az adatok redundáns tárolását úgy érjük el, hogy több párhuzamos valóságot (koordinátateret) hozunk létre. Minden csomópont más-más zónákért felelős min-

5. ábra Kétdimenziós CAN topológia



den valóságban, és így minden kulcs más-más csomópontra kerül minden egyes valóságban. Ezen felül az útvonalkeresés is indítható minden valóságban, hogy végül a legrövidebb útvonalat választhassuk ki.

3.3. Freenet

A Freenet célja [5] egy olyan hálózat létrehozása az információk megosztására, amely ellenáll mindenfajta cenzúrálási kísérletnek, és abszolút anonimitást biztosít felhasználóinak. Minden felhasználó hozzájárul valamennyi tárhellyel a hálózathoz, ám a fájlok szegmentáltan kerülnek tárolásra, és a különböző szegmenseket szétszórva, redundánsan és titkosítva helyezi el, így egyetlen felhasználónak sincsen tudomása arról, mit tárol a számítógépe (ami nagyon nehézzé teszi a felhasználók elleni jogi támadásokat). A Freenet nem ad keresési funkciókat, és egy heurisztikus kulcs-alapú útvonalválasztást használ (és így, bár nem olyan strukturált, mint a többi DHT implementáció, ő maga is implicit módon egy DHT-hálózat), és nem ad garanciát arra, hogy meg is találjuk azt, amit keresünk.

Ahogyan a többi DHT-hálózatban is, minden csomópontnak csak néhány másik csomópontról van információja. Amikor megkap egy kérést, egy csomópont megkeresi az adatot a saját adattárában. Amennyiben az adat nem áll rendelkezésre, továbbítja a kérést annak a csomópontnak, amelyikről úgy gondolja, hogy a leggyorsabban fogja megtalálni az adatot (ez a „következő generációs útvonal-választási” – Next Generation Routing – protokoll.

A korábbi verziókban a kéréseket azoknak a csomópontoknak továbbítottuk, amelyik előzőleg a mostani kéréshez leginkább hasonló kérésre már adott választ). Amennyiben egy csomópont hurkot észlel az útvonalválasztási táblában (megkap egy kérést, amelyet egyszer már továbbított), levágja azt. A kérés továbbítása addig folytatódik, amíg a keresett adatot megtaláljuk, vagy amíg a kérés elér egy meghatározott továbbítási számot (hops to live).

Amikor a kérés eléri az adatot tartalmazó csomópontot, az adat visszafele halad a kérés útvonalán, csomópontról csomópontra; az útvonal mentén egyetlen csomópont sem tudja, hogy az, ahonnan az adatot kapta, az adat forrása, vagy csak egy egyszerű csomópont a kérés továbbítási útvonalán. Ezen felül minden csomópont cacheli az adatot, így a népszerű tartalom többszörösen tárolódik a hálózatban.

Minden csomópont egy bizonyos típusú kulcsra specializálódik, amelyeket véletlenszerűen osztunk szét a hálózat felállításakor. Ahogy egyre több adat kerül tárolásra, a hasonló kulcsokat tároló csomópontokon csomósodik, és a hálózat egyre strukturáltabbá válik.

4. Peer-to-peer alapú elosztott fájlrendszerek

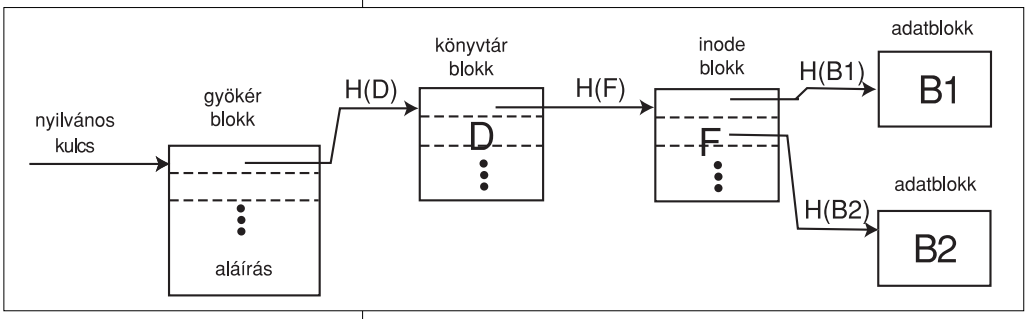
A fő problémát egy p2p alapú elosztott fájlrendszer létrehozásánál az jelenti, hogy a decentralizált hálózatok, és ezek közt a DHT-alapú hálózatok (amelyek a skálázhatóságuk és strukturáltságuk miatt a legjobb alapnak tűnnek) nem rendelkeznek indexelési/listázási/keresési képességgel. Ez a képesség pedig bármely fájlrendszerhez alapvető fontosságú (elég csak arra gondolni, hogyan használjuk a leggyakrabban a fájlrendszereket: belépés egy könyvtárba, a tartalom listázása, és így tovább). Ezért első lépésben egy indexelési réteget kell felépíteni a DHT réteg fölé.

Itt két megoldást mutatunk be, amelyek két különböző fájlrendszer-paradigmának felelnek meg, és ennek megfelelően különböző módon oldják meg ezt a problémát.

4.1. CFS

A CFS (Cooperative File System – kooperatív fájlrendszer) egy Chord hálózat felett nyújt fájlrendszer szolgáltatásokat [6]. Három rétegből áll: egy Chord rétegből, amely az adatblokkok megtalálásához szükséges útvonalválasztásért felelős; egy DHash rétegből, amely a strukturálatlan adatblokkok megbízható tárolását biztosítja; és egy fájlrendszer rétegből, amely a blokkokat fájlonként értelmezi, és fájlrendszer interfészt biztosít az alkalmazásoknak. A DHash réteg a népszerű fájlok blokkjait egyszerre több szerveren tárolja a terhelés megosztására. Az adatblokkokat cache-eli azokon a szervereken, amelyeknél valószínűsíthető, hogy keresik őket, és támogatja az előre lekérést (pre-fetching) a letöltési késleltetés csökkentésének érdekében. A blokkokat redundánsan tárolja a rendszer hibatűrő-képességének növelése érdekében.

6. ábra Egyszerű CFS könyvtárstruktúra



A struktúra nagyon hasonlít a UNIX V7 fájlrendszerére, de DHash blokkokat és blokk-azonosítókat alkalmaz a merevlemez-blokkok és -címek helyett. Egy CFS-ben egyszerre több könyvtárstruktúra is található, amelyek csak olvashatóak, kivéve a könyvtárstruktúra létrehozóját – így gyakorlatilag minden tartalom-kiadó saját fájlrendszerrel rendelkezik, melynek része egy gyökérblokk, amelyet a kiadó a saját nyilvános kulcsával ír alá, amely ezek után a fájlrendszer azonosítására szolgál a CFS-ben. A gyökér-blokk bejegyzései könyvtárblokkokra mutatnak, amelyek további könyvtárblok-

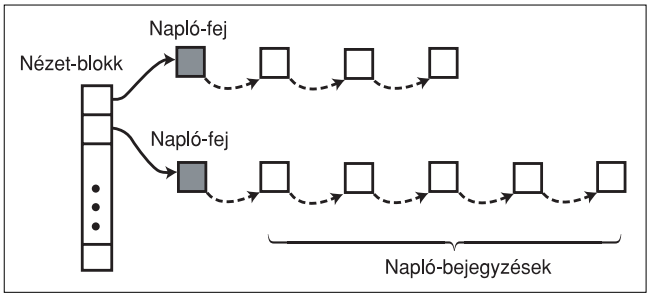
kokra, vagy inode blokkokra mutató bejegyzéseket tartalmaznak. Az inode blokkok már magukra az adatblokkokra mutató bejegyzéseket tartalmaznak.

4.2. Ivy

Az Ivy [7] nagyon hasonló a CFS-hez, és ugyanazt a DHash réteget használja, azonban írható/olvasható fájlrendszert biztosít egy naplózott adatstruktúra alkalmazásával. Egy Ivy-napló egy megváltoztathatatlan naplóbejegyzésekből áló láncolt lista. A legújabb naplóbejegyzést a résztvevők a napló elején tárolják, és a nyilvános kulcsának segítségével található meg a DHash rétegben. Egy fájl konkurens módosításait verzió-vektorok rendezésével oldja meg az Ivy, és minden potenciális módosító saját naplóval rendelkezik.

A fájlok írási jogosultságait titkosítással oldják meg: a fájlt megtekintő felhasználók egyszerűen figyelmen kívül hagyják az adott fájl módosítására jogosulatlan felhasználók által aláírt naplókat.

7. ábra Ivy nézet és naplóbejegyzések



5. Konklúzió

A fájlrendszerek megvalósítása p2p hálózatokon jelenleg még megvalósításra váró feladat. A félig centralizált, hibrid p2p rendszerek architekturális gyengeségeit a gyakorlati tapasztalatok is megmutatták, ráadásul ezek a rendszerek nem nyújtanak lényeges előrelépést a hagyományos, stabil, és rendkívül alaposan ellenőrzött hálózati fájlrendszerekhez képest.

A p2p hálózatok második generációja már megszabadult a központi szerverektől, ám ennek a skálázhatóság, valamint a listázási/keresési funkciók feláldozása volt az ára. A DHT alapú p2p hálózatok skálázható, strukturált és megbízható rendszerek megvalósítását teszik lehetővé, azonban még mindig hiányzik belőlük a tartalom listázásának/keresésének lehetősége, ami a fájlrendszerek alapvető összetevője. A DHT hálózatoknál elég magától értetődő ötlet a fájlrendszer metaadatait tartalmazó blokkok hagyományos adatblokkokként történő tárolása, és ezen a téren mind a CFS, mind az Ivy ígéretes perspektívákat nyújt. Ám a CFS csak olvasható fájlrendszert valósít meg, és egyik megoldás sem oldja meg a fájlrendszer gyökérkönyvtárának/gyökérblokkjának tárolását, mely nélkül nem lehet a fájlrendszer tartalmát elérni anélkül, hogy legalább valamilyen a priori tudással rendelkeznénk a fájlrendszerről. Következésképpen a p2p alapú fájlrendszerek fejlődési kulcskérdésének jelen pillanatban a fájlrendszer gyökerének elosztott tárolása tűnik.

Irodalom

[1] Ion Stoica, Robert Morris, David Karger, M. Frans Kaashoek, Hari Balakrishnan:
Chord: A Scalable Peer-to-peer Lookup Service for Internet Applications.

[2] Sylvia Ratnasamy, Paul Francis, Mark Handley, Richard Karp, Scott Shenker:
A Scalable Content-Addressable Network.
SIGCOM'01, August 2001.

[3] Antony Rowstron, Peter Drushel:
Pastry: Scalable, decentralized object location and routing for large-scale peer-to-peer systems.
18th IFIP/ACM International Conference on Distributed Systems Platforms. Heidelberg, Germany, November 2001.

[4] Ben Y. Zhao, Ling Huang, Jeremy Stribling, Sean C. Rhea, Anthony D. Joseph, John D. Kubiatowicz:
Tapestry: A Resilient Global-scale Overlay for Service Deployment. IEEE Journal on Selected Areas in Communications, Vol. 22, No.1, January 2004.

[5] Ian Clarke, Oskar Sandberg, Brandon Wiley, Theodore W. Hong:
Freenet: A Distributed Anonymous Information Storage and Retrieval System.

[6] Frank Dabek, M. Frans Kaashoek, Robert Morris, Ion Stoica:
Wide-area Cooperative Storage with CFS.
SOSP '01, October 2001.

[7] Athicha Muthitacharoen, Robert Morris, Thomer M. Gil, Benjie Chen:
Ivy: A Read/Write Peer-to-Peer File System.