dc_565_12

Pletyka protokollok nagyméretű elosztott rendszerekben

MTA Doktora értekezés tézisei

Jelasity Márk

Szeged, 2013

Az értekezés teljes terjedelmében elérhető a következő címről:
http://www.inf.u-szeged.hu/dr/doktori-mu.pdf

Tartalomjegyzék

1. Az értekezés célkitűzései

Nagy méret és elosztottság mint fontos tendenciák. Az elmúlt évtizedben a számítástechnika egyik legfontosabb tendenciája a párhuzamosság és elosztottság növekedése volt szinte minden területen, kezdve a processzorok sokmagossá válásától egészen a sok százezer számítógépből álló óriási adatközpontokig.

Ezen a trenden belül az értekezés fő motivációját azok az elosztott számítógéprendszerek jelentik, amelyek az Internet gyors térnyerésének köszönhetően az ezredfordulótól kezdve alakultak ki. Ilyen rendszerekre jó példák azok az internetes szolgáltatások, amelyeket a meredeken növekvő igények miatt már csak rengeteg számítógépből álló komplex, többrétegű rendszerekkel lehet megvalósítani, mint amilyen az internetes keresés (Google, Microsoft) vagy az online áruházak (Amazon, Apple). Az ehhez kifejlesztett infrastruktúra optimális üzleti hasznosítására való törekvés eredményeképpen születtek meg a felhő szolgáltatások, és a háttérben az őket támogató szoftveres és mérnöki megoldások sora.

Az érdeklődésünkbe eső rendszerek egy másik csoportja az alulról fölfelé szerveződő óriási méretű elosztott rendszerek, amelyek szintén az Internetnek köszönhetően alakulhattak ki. Ezek központi irányítás nélküli egyenrangú résztvevőkből állnak, azaz un. peer-to-peer (P2P) hálózatok. A legfontosabb P2P alkalmazás kezdetben a fájlcserélés volt (Gnutella, Bittorrent), később pedig ma is népszerű szolgáltatások (Skype, Spotify), sőt az Interneten terjedő kártékony bot-hálózatok is is felhasználtak P2P algoritmusokat. Ide sorolhatók még az un. desktop grid rendszerek is, ahol a résztvevők kihasználatlan gépidejében különböző—általában tudományos—feladatokat oldhatunk meg (BOINC).

Megbízhatóság mint fontos követelmény nagy elosztott rendszerekben. Ha egy merevlemez várhatóan öt évente hibásodik meg, akkor egy 100 000 merevlemezt használó rendszerben nagyjából félóránként hibásodik meg egy merevlemez. A Google adatközpontjaiban, ahol ennek a kapacitásnak a többszöröse is megtalálható, külön célberendezések vannak telepítve a hibás lemezek folyamatos megsemmisítésére. Egy P2P rendszerben pedig további hibaforrás, hogy a résztvevők bármikor kiléphetnek vagy beléphetnek figyelmeztetés nélkül.

A pletyka mint hasznos eszköz. Ilyen feltételek mellett az egyik legfontosabb tervezési szempont a rendszerek megbízhatósága. Ennek az elérésére számos módszer létezik, amelyeket az alapján csoportosíthatunk, hogy milyen garanciát nyújtanak a felhasználó alkalmazás felé. Ezeknek a garanciáknak a leírása különböző konzisztenciamodellek segítségével történik. A konzisztencia egyik fajtája az un. végső konzisztencia (*eventual consistency*), ami azt jelenti, hogy ha egy bizonyos pont után nem történik változás (hiba, vagy bármilyen módosítás) akkor a rendszer véges időn

1. algoritmus. A pletyka algoritmus váza.

```
 1: loop
 2:     wait(Δ)
 3:     p ← selectPeer()
 4:     if push then
 5:         sendPush(p,state)
 6:     else if pull then
 7:         sendPullRequest(p)
 8:
 9: procedure onPullRequest(m)
10:     sendPull(m.sender,state)
11: procedure onPush(m)
12:     if pull then
13:         sendPull(m.sender,state)
14:     state ← update(state,m.state)
15:
16: procedure onPull(m)
17:     state ← update(state,m.state)
```

belül konzisztenssé válik. A fogalom többé kevésbé az irányításelmélet stabilitásfogalmához hasonlítható.

Ennek elérésére egy lehetséges általános megközelítést ajánlanak a pletyka (vagy járvány) algoritmusok. Kezdetben ezeket az algoritmusokat replikált adatbázisok megvalósítására javasolták, ahol a feladat az volt, hogy a módosításokat minden másolat megkapja [1]. Az algoritmusnak a vonzereje abban rejlik, hogy a hálózat méretének (N) függvényében $O(\log N)$ idő alatt, akár $O(\log \log N)$ üzenet felhasználásával nagy valószínűséggel minden másolathoz eljut minden frissítés.

A pletyka paradigma általánosításai. A disszertáció pletyka algoritmusokat tárgyal számos olyan probléma megoldására, amelyek a hagyományos adatszóráson (broadcast) messze túlmutatnak. A mellékelt algoritmusséma olyan absztrakciós szintet képvisel, amelyben megfogalmazhatók globális számítási feladatok, mintavételezési feladatok, vagy átfedőhálózatok (más szóval elosztott adatstruktúrák) építése. Mindezen feladatok közös jellemzője, hogy a résztvevők periodikus jelleggel információt cserélnek más résztvevőkkel, és ennek eredményeképpen módosítják a saját állapotukat. Annak függvényében, hogy a kommunikációs partnereket hogyan választjuk ki, milyen információt cserélünk, és ezzel az információval mihez kezdünk, számtalan funkció valósítható meg, ahogy később látni fogjuk.

Példákon keresztül bemutatjuk azt is, hogy a különböző feladatokat ellátó pletyka algoritmusok építőkövekként használhatók és egymással hatékonyan kombinálhatók komplex pletyka algoritmusokat létrehozva, amelyek akár komplett, több rétegből álló teljes rendszereket is megvalósíthatnak.

2. Kutatási módszertan

Sajátos módszertan. A disszertáció módszertana meglehetősen sajátos, és rokonságot mutat a komplex rendszerek tanulmányozására használatos módszertanokkal. A kutatásunk tárgyát olyan rendkívül nagyméretű, sok komponensből álló rendszerek képezik, amelyekben a komponensek egymással is kölcsönhatásban állnak, és folyamatosan változó, összetett, külső környezeti feltételek mellett kell egy meghatározott funkciót ellátniuk hatékonyan. Sokszor nemcsak az algoritmus belső kérdéseit kell megérteni, hanem maga a környezet (pl. az Internet) releváns és gyakran emergens tulajdonságainak a megértése, modellezése sem megoldott. A terület tudományos közössége tehát joggal várja el a rendszerek minél élethűbb vizsgálatát, ami pontos szimulációt, és valós környezetben való tesztelést is megkíván. Ennek az eléréséhez pedig ténylegesen implementálnunk szükséges ezeket a rendszereket, és a teszteléshez szükséges infrastruktúrát is.

Valójában maga a metodológia is kutatás tárgyát képezi, hiszen nincs teljes egyetértés abban a tekintetben, hogy mi a célravezetőbb: a minél valósághűbb környezetben való tesztelés (aminek a reprodukálhatósága sokszor problémát okoz), vagy a modellezés és szimuláció valósághűségére, pontosságára való törekvés (amit újabban sok kritika ér arra hivatkozva, hogy ez oda vezet, hogy „maga a táj lesz a térkép” [2]), vagy inkább a legfontosabb tulajdonságok absztrahálása és közelítő modellezése (ami esetleg nem jelenet kielégítő biztosítékot valós informatikai alkalmazások számára).

Modellezés, elméleti eredmények. Az elmélet szerepe egyfelől az algoritmusaink viselkedésének a pontos leírása: az elosztott átlagszámítás konvergenciasebességére pl. praktikusan is jól használható, pontos közelítést sikerült adni. Az elméleti megfontolások másik, gyakoribb funkciója közelítő modell alkotása, amellyel tudatos egyszerűsítéseken keresztül egy folyamat lényegi tulajdonságait és dinamikáját szemléltetjük. A modellek pontosságát (azaz az egyszerűsítések létjogosultságát) kísérleti módszerekkel ellenőrizzük.

Szimulációk: PeerSim. A fő eszközünk a szimuláció. A disszertációban szereplő összes szimulációt a PeerSim nevű szimulátorral készítettük [3], amit nyílt forrású szoftverként elérhetővé is tettünk. Olyan szimulátort szerettünk volna létrehozni, ami a P2P alkalmazások teljes körét képes kezelni, de ami ugyanakkor az ismert hálózatszimulátoroknál magasabb absztrakciós szintet is támogat (pl. az alsóbb hálózati rétegeket nem modellezzük részletesen) ezáltal elérve a kulcsfontosságú skálázhatóságot, hiszen sokszor milliós hálózatokat kell vizsgálnunk. A stratégia jónak bizonyult, amit az is mutat, hogy az elmúlt évtizedben a PeerSim cikkek százaiban került felhasználásra, és több egyetemen a tanításban is felhasználják jelenleg is. A PeerSim szimulátor lehetővé teszi a P2P algoritmusok hatékony vizsgálatát több

szimulációs modellben, a sejtautomata-szerű, multi-ágens rendszerekben népszerű ciklikus modellezésen keresztül az eseményvezérelt, realisztikus szimulációkig. A szimulátor jól skálázódik, és programkönyvtárakat ajánl a hálózati környezet magas szintű dinamikájának a modellezésére, illetve az átfedőhálózatok vizsgálatára.

Tesztelés valós környezetben. Sok esetben valós implementációkat is készítettünk, amelyeket különböző valós teszthálózatokon vizsgáltunk kísérletileg, mint amilyen a PlanetLab hálózat [4] vagy a holland DAS országos számítási klaszter.

Rendszermodell: átfedőhálózatok, megbízhatatlan kommunikáció A módszertan részét képezik az általunk vizsgált hálózati környezetről tett feltevéseink. Ezek az egyes fejezetekben kismértékben változnak, de a közös elemek a következők. Feltesszük, hogy a rendszer számítási eszközök hálózata. Az eszközöket csúcsoknak hívjuk. Tipikusan nagyon sok csúcsot tételezünk fel, akár milliós vagy milliárdos nagyságrendben. A csúcsok üzenetküldés útján képesek egymással kommunikálni egy csomagkapcsolt hálózat segítségével. Bármelyik csúcs küldhet üzenetet bármikor bármelyik másik csúcsnak, az üzenetküldéshez elegendő a címzett címét tudni. A csúcsokból *átfedőhálózatok* (overlay networks) építhetők, amelyekben az i csúcsból akkor vezet a j csúcsba él, ha i ismeri j-t. Az átfedőhálózat elnevezése arra utal, hogy a fizikai kommunikációs hálózattól teljesen független logikai hálózatról van szó. Az átfedőhálózatok gyakran elosztott adatstruktúrákat valósítanak meg (pl. elosztott hash táblákat). A gyakorlatban az átfedőhálózatok természetesen nem függetleníthetik magukat a fizikai hálózattól teljes mértékben, költségszempontok miatt. Az üzenetek elveszhetnek, vagy késhetnek. Bármelyik csúcs bármikor meghibásodhat vagy kiléphet a hálózatból.

3. Társ mintavételezés

A probléma. A pletyka algoritmus egyik fő komponense a helyben véletlen szomszédokat szolgáltató SELECTPEER metódus. Ennek a megvalósítása a korábban vázolt rendszermodellben komoly kihívás, mert a csúcsok teljes listáját tárolni és folyamatosan frissíteni nehéz, ezért elosztott mintavételt kell megvalósítani.

Azonosítottuk a társ mintavételezést, mint önálló középréteg szolgáltatást. Javasoltunk egy pletyka alapú megvalósítást, amely egy véletlen fedőhálózat folyamatos keverésén alapul, minden központi segítség nélkül.

A megoldás. A publikációk amikre építettünk a következők: [5–7]. A probléma megoldásához először azonosítottuk és motiváltuk a társ mintavételezést, mint önálló középréteg szolgáltatást, és megterveztük az interfészét az alkalmazások felé. A társ mintavételezés megvalósítására egy pletyka alapú protokollt javasoltunk, amelynek a lényege, hogy minden csúcs tárol egy kisszámú véletlen mintát a hálózat csúcsaiból. Ezek a minták egy véletlen fedőhálózatot definiálnak. A csúcsok úgy jutnak új mintákhoz, hogy az aktuálisan ismert szomszédokkal keverési lépéseket hajtanak végre folyamatosan, amelynek során egymás szomszédainak a segítségével frissítik a saját szomszédlistájukat.

Az algoritmust általános keretként fogalmaztuk meg, amely paraméterezhető, és amely a szakirodalomban ismert hasonló javaslatokat (amelyek egy része tőlünk származott) általánosítja és egységes sémába illeszti. A paraméterekkel folytonosan

2. algoritmus. Newscast

```
1:  loop
2:      wait(Δ)
3:      p ← selectGPSPeer()
4:      sendPush( p, toSend() )
5:      view.increaseAge()
6:
7:  procedure ONPUSH(m)
8:      sendPull( m.sender, toSend() )
9:      onPull(m)
10:
11: procedure ONPULL(m)
12:     update(m.buffer,c)
13:     view.increaseAge()
14: procedure UPDATE(buffer,c)
15:     view.append(buffer)
16:     view.removeDuplicates()
17:     view.removeOldItems(view.size-c)
18:
19: procedure TOSEND
20:     buffer ← ((MyAddress,0))
21:     buffer.append(view)
22:     return buffer
```

lehet állítani, hogy egyes tulajdonságok mekkora hangsúlyt kapjanak: a csúcsokon gyűjtött minták korrelálatlansága más csúcsok mintáival vagy a hibatűrő és önjavító képesség domináljon, vagy a kettő kombinációja jelenjen meg. Itt a NEWSCAST algoritmust mellékeljük (c a helyben tárolt minta (view) nagysága), ami a keret egy lehetséges kitöltését jelenti, a spektrum leginkább önjavító szélét megvalósítva.

Kiértékelés: helyi véletlenség. Egy rögzített csúcsra érkező véletlen minták véletlenségét vizsgáltuk a véletlenszám generátorok vizsgálata során bevett módszertan segítségével, azaz speciálisan kifejlesztett statisztikai tesztek egy halmazát alkalmazva. Megállapítottuk, hogy a minták megfelelnek a véletlenszám generátorokkal szemben támasztott követelményeknek.

Statisztikailag igazoltuk a helyi minták véletlenségét. Kísérletileg igazoltuk, hogy a keveredő véletlen átfedőhálózat rendkívül adaptív és robosztus, akár a korrelálatlan véletlen gráfok. Az öngyógyító verzió klaszterezettsége magas, de időben nem tartósak a korrelációk.

Kiértékelés: globális véletlenség. Világos, hogy az egyes csúcsokon kapott minták nem függetlenek más csúcsok mintáitól. Módszertani újításunk lényege, hogy a dinamikus véletlen átfedőhálózatra koncentráltunk és összehasonlítottuk azokkal a véletlen gráfokkal, amiket a függetlenség feltevésével kapnánk. A fokszámeloszlásra és a legrövidebb utak átlagos hosszára koncentráltunk, hiszen előbbi a terheléselosztást, utóbbi a pletyka globális tulajdonságait (terjedési sebesség) határozza meg. Vizsgáltuk még a klaszterezettségi együtthatót is is, ami direkt módon a szomszédok korreláltságát jellemzi.

Módszertant tekintve szimulációkat használtunk, valamint valós implementációt is kisebb skálán. Igazoltuk, hogy a rendszer drasztikusan különböző kezdőállapotokból ugyanabba a stabil konfigurációba konvergál, amelyben a paraméterek függvényében a vizsgált mutatók stabil értékeket vesznek fel. Ha az öngyógyítás mértékét növeljük, a klaszterezettség drasztikusan megnő, de a klaszterek gyorsan változnak (keletkeznek és feloszlanak). A fokszámeloszlás viszonylag kis szórású, és mind a gráf csúcsai felett egy adott időpontban, mind pedig egy adott csúcson különböző időpontokban értelmezve megegyezik.

Kiértékelés: hibatűrés. A rendszer hibatűrését is vizsgáltuk változatos forgatókönyvek mellett. Többek között a csúcsok folyamatos cserélődését (churn) illetve a hálózat jelentős részének a hirtelen kiesését vizsgáltuk. Ebben a tekintetben azt találtuk, hogy a dinamikus hálózataink öröklik a viszonylag kis szórású (nem nehézfarkú) fokszámeloszlással rendelkező véletlen gráfok kedvező tulajdonságait, és a hibák megszűntével gyorsan visszaállnak stabil állapotukba.

4. Átlagszámítás

A probléma. A teljesen elosztott, dinamikus rendszermodellünkben sok alkalmazás számára szükséges olyan számításokat végezni az elosztott csúcsok felett, amelyek valamilyen módon aggregálják a helyi adatokat, pl. átlagot számolnak. Ilyenek pl. az elosztott rendszerek megfigyelésére, irányítására szolgáló alkalmazások, vagy maga az aggregáció is lehet elsődleges cél, pl. szenzorhálózatokban.

Teljesen elosztott algoritmust javasoltunk adat aggregációra, ami többek között az átlag, egyéb középértékek, minimum, maximum, és magasabb momentumok kiszámítására képes.

A megoldás. A publikációk amikre építettünk a következők: [8–10]. Az absztrakt pletyka keretbe illeszkedő aggregációs algoritmust javasoltunk, amelynek alapötlete, hogy olyan periodikus, lokális információcserére épülő dinamikus rendszert definiál, amelyben a kezdeti adatokból kiindulva minden csúcs a keresett aggregált értékhez konvergál a mellékelt algoritmus szerint.

A mellékelt algoritmus UPDATE(x, y) metódusának implementálásától függően számolhatunk átlagot (UPDATE$(x, y) = (x + y)/2$), maximumot vagy minimumot (UPDATE$(x, y) = \max(x, y)$), általános közepeket (UPDATE$(x, y) = f^{-1}((f(x) + f(y))/2)$), vagy momentumokat (UPDATE$(x, y) = (x^k + y^k)/k$). Ezekből építkezve komplex számítások is végezhetők mint amilyen az EM-algoritmus vagy a naiv Bayes algoritmus. Speciális kiindulási adatokból kiindulva a hálózat mérete is meghatározható (ha egy csúcs értéke 1, a többié 0, akkor az átlag $1/N$).

Elméletileg beláttuk, hogy a közelítések varianciája a hálózatban exponenciálisan csökken, és megadtuk a pontos sebességet is.

Elméleti eredmények. Az átlagszámítás esetében beláttuk, hogy a csúcsokon található közelítő értékek varianciája exponenciálisan csökken. Jellemeztük a csökkenés multiplikatív faktorát is egy iteráció végrehajtása során, aminek az értéke $\exp(-1/2)/2$. Beláttuk, hogy az optimális faktor $1/4$, ami akkor áll elő, ha minden iterációban kettő, egymástól független teljes párosítás párjai szerint végezzük el az UPDATE metódust (ennek a megvalósítása viszont elosztottan nem kifizetődő).

Ezen felül elméletileg jellemeztük az algoritmus viselkedését abban az esetben, ha üzenetvesztés is lehetséges, vagy csúcsok kieshetnek a számítás végzése közben. Azt találtuk, hogy az előbbi esetben a számítás egyfelől arányosan lelassul, másfelől

3. algoritmus. Elosztott aggregáció

```
1: loop
2:     wait(Δ)
3:     p ← selectPeer()
4:     sendPush(p,x)
5: procedure ONPUSH(m)
6:     sendPull(m.sender,x)
7:     x ← update(m.x, x)
8:
9: procedure ONPULL(m)
10:    x ← update(m.x, x)
```

(a lassulást kompenzáló skálázás után is) a konvergencia faktor az $\exp(-1)$ értékhez tart az üzenetvesztés valószínűségének a növekedésével.

Praktikus részletek. Az algoritmust kiegészítettük egy újraindító mechanizmussal, amely adott számú iteráció után egy újabb „korszakot” (epoch) indít. Mivel a rendszer gyorsan konvergál, egy korszak rövid lehet, pl. 30 iteráció. A megoldás aszinkron, és biztosítja a rendszer robosztusságát a churn (változó csúcshalmaz) és más hibák esetében is. A robosztusság további növelése érdekében párhuzamos, független példányokat is futtattunk az algoritmusból, és a medián értéket vettük a közelítés értékének.

Alapos kísérleti elemzéssel demonstráltuk az újraindító mechanizmussal kiegészített algoritmus gyakorlati megbízhatóságát, és megerősítettük az elméleti predikcióink pontosságát.

Kiértékelés. Kísérletileg vizsgáltuk az algoritmus tulajdonságait számos forgatókönyv mellett, amik valós környezetben előforduló problémákat modelleztek, mint pl. a közelítendő átlag folytonos változása, üzenetvesztés, és tömeges csúcs-kilépés. Az algoritmust implementáltuk és teszteltük a PlanetLab teszthálózatban is.

A kísérletek eredménye megerősítette az elméleti eredményeink pontosságát mind a konvergenciasebesség, mind pedig a hibatűrés terén. A fent említett praktikus kiegészítésekkel valós környezetben is robosztus közelítéseket kaptunk.

5. Elosztott hatványiteráció

A probléma. A korábban tárgyalt adat aggregáció mellet másfajta globális jellegű számítások is szerepet kapnak számos alkalmazásban. Ha adott egy átfedőhálózat, sokszor érdekes ennek a hálózatnak a spektrális tulajdonságait vizsgálni. Pl. a PageRank algoritmus [11], amelyet csúcsok fontossági rangsorolására használhatunk, egy átfedőhálózat domináns sajátvektoraként áll elő. A domináns sajátvektor további alkalmazásaira példa még a szociális hálózatokban helyi bizalmi viszonyokból globális megbízhatósági érték számolása, vagy a spektrális klaszterezés.

Elosztott aszinkron algoritmust javasoltunk nemnegatív élsúlyú, erősen összefüggő átfedőhálózatok domináns sajátvektorának a meghatározására tetszőleges pozitív domináns sajátérték esetére.

A megoldás. A publikáció amire építettünk a következő: [12]. A megoldásunk illusztrálja a korábban tárgyalt pletyka aggregáció alkalmazhatóságát komplexebb algoritmusok részeként. A szakirodalomból ismert kaotikus iteráció [13] algoritmusát (amely kizárólag akkor használható, ha a domináns sajátérték 1, és a mátrix nemnegatív és irreducibilis) kiegészítettük a pletyka alapú aggregációval abból a célból, bármilyen pozitív domináns sajátérték esetén használható legyen.

Az algoritmus az ismert kaotikus iteráció [13] pletyka alapú elosztott normalizálására épül, és a PageRank értékek számolására is alkalmas.

Az aggregáció két alkalmazása. Az aggregáció két célból is bevethető. Egyfelől a konvergencia biztosítása céljából. Ha $\mathbf{x}^*$ a domináns sajátvektor, és $\lambda > 0$ a domináns sajátérték, akkor a mellékelt algoritmusban $b_i/x_i^* = \lambda$ minden i csúcson. A konvergenciát úgy érjük el, hogy a helyi $b_i^{(t+1)}/x_i^{(t)}$ kiindulási értékek fölött geometriai közepet számolunk folyamatosan, és ennek a középnek az aktuális közelítésével korrigáljuk az x_i értékét. Ennek a rendszernek a fixpontja az az állapot, amikor x_i nem változik. Viszont a konvergált $\mathbf{x}$ vektor hossza bármekkora lehet, nem ismert.

Az aggregáció második alkalmazása a vektor hosszának a normalizálása. Ez minden esetben fontos amikor a számolt értékek alapján tisztán helyben kell döntéseket hozni, tehát nem csak a relatív különbség érdekes az értékek között. Ez az aggregáció maguk az x_i értékek fölött működik. Ezzel az értékkel nem feltétlenül kell korrigálni a hatványiteráció alatt, hiszen elég csak ismerni az vektorhosszt, amivel aztán helyben is lehet normalizálni. Mindazonáltal javasoltunk korrigáló mechaniz-

4. algoritmus. Aszinkron hatányiteráció az i csúcson [13].

```
1: loop
2:    wait(Δ)
3:    for each j ∈ out-neighbors_i do
4:       send weight A_ji x_i to j
5:    b_i ← Σ_{k∈in-neighbors_i} b_ki
6:    x_i ← b_i
7: procedure ONWEIGHT(m)
8:    k ← m.sender
9:    b_ki ← m.x
```

must, pl. olyan extrém esetekre, amikor numerikus túlcsordulásra vagy alulcsordulásra lehet számítani.

A PageRank algoritmus a fenti normalizálások mellet egy un. véletlen szörfös operátort is tartalmaz, amely egy kis súlyú élt ad a gráfhoz minden csúcs felé minden csúcsból. Ennek a hatását szintén lehet helyben, újabb élek nélkül modellezni ha a vektorelemek összege ismert (ami a vektorhossz egy lehetséges definíciója, hiszen esetünkben minden vektorelem nemnegatív). Ez lehetővé teszi a PageRank megvalósítását, ha a vektorhossz komponens a vektorelemek összegét közelíti.

Kiértékelés. Az algoritmust mesterségesen generált és valós hálózatokon is kiértékeltük. Spektrális tulajdonságaikat tekintve a teszthálózatok között voltak olyanok, amelyek gyors, és voltak amelyek lassú konvergenciát eredményeznek a hatványiterációs módszerek esetén. A valós hálózat egy a webről gyűjtött, nyilvánosan elérhető hiperlinkhálózat volt, amelyen a PageRank algoritmust teszteltük. A modellezett hibák az üzenetvesztés és üzenet késleltetés voltak. A kísérletek azt mutatták, hogy ha a vektorhossz segítségével korrigáljuk a frissítési szabályt (tehát a rendszer adott normájú domináns sajátvektorhoz fog konvergálni) akkor szélsőségesen megbízhatatlan környezeteket modellező forgatókönyvek esetében előfordulhat, hogy nem konvergál az algoritmus. Mint említettük, ez a korrekció nem feltétlenül szükséges. Enélkül azonban rendkívül robosztus viselkedést tapasztaltunk, minden vizsgált forgatókönyv mellett konvergált az algoritmus.

6. Átfedőhálózatok szeletelése

A probléma. Számos elosztott rendszerben merül fel az erőforrások alkalmazásokhoz rendelésének a kérdése. Ilyenek pl. a felhő vagy az un. desktop grid rendszerek. Az általunk vizsgált rendszermodellben nincs központi irányítás, tehát a rendelkezésre álló erőforrások felosztását és menedzselését elosztottan kell végezni. Ennek egyik részfeladata a hálózat szeletelése, amin azt értjük, hogy adott képességek mentén (pl. rendelkezésre álló memória) a hálózat csúcsait osztályozzuk (pl. két egyenlő részre: kevés és sok memóriával rendelkező csúcsokra). A nehézséget az adja, hogy a rendelkezésre álló erőforrások eloszlása nem ismert, így lokálisan nem dönthető el, hogy egy adott csúcs melyik osztályba tartozik.

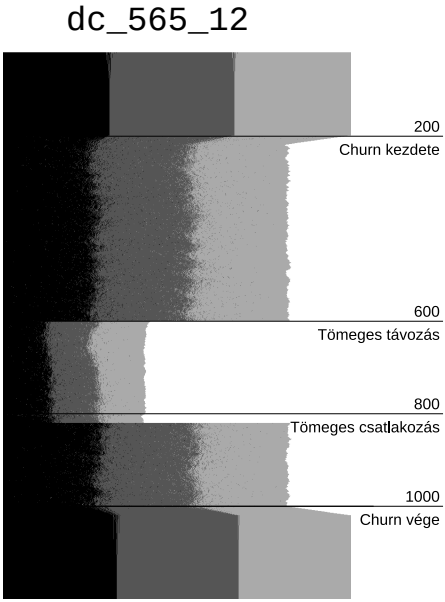
Azonosítottuk a szeletelés problémáját, amelyben ismeretlen eloszlású lokális erőforrások mentén a csúcsok osztályokba sorolását keressük.

A megoldás. A publikáció amire építettünk a következő: [14]. Tegyük fel hogy egy i csúcs x_i erőforrással rendelkezik. Minden csúcs egyenletes eloszlásból vesz egy r_i mintát. Az algoritmus ezeket a mintákat fogja rendezni az x_i értékek mentén. A rendezés után, mivel az r_i értékek eloszlása ismert, ezek segítségével már lehet helyben döntéseket hozni a különböző osztályokhoz való tartozásról. A rendezés úgy zajlik, hogy minden csúcs cserepartnereket keres, akikkel az r értékeket kicserélve a rendezettséget növelni tudja (l. a mellékelt algoritmust).

Elméleti eredmények. Szoros kapcsolatot mutattunk ki az átlagolás feladatával. Bemutattuk, hogy a rendezés során a helyes indextől vett abszolút távolság várható értékben átlagolódik egy sikeres cserét követően, ami azt jelenti, hogy a sikeres cserék sorozatát tekintve az algoritmus erre a rendezetlenségi mértékre nézve átlagolásként viselkedik.

5. algoritmus. Szeletelés

```
1:  loop
2:      wait(Δ)
3:      p ← selectSlicePeer()
4:      if p ≠ null then
5:          sendPush(p,(x, r))
6:  procedure ONPUSH(m)
7:      sendPull(m.sender,(x, r))
8:      onPull(m)
9:
10: procedure ONPULL(m)
11:     if (x − m.x)(r − m.r) < 0 then
12:         r ← m.r
```



1. ábra. Az algoritmus illusztrációja különböző hiba forgatókönyvek esetében.

A problémára teljesen elosztott rendezésen alapuló megoldást adtunk, ami az átlagszámításhoz hasonló tulajdonságokkal rendelkezik.

Kiértékelés. Szimulációs kísérletekben alátámasztottuk az elméleti eredmények jóslatait, valamint számos típusú hibát megvalósító forgatókönyvben vizsgáltuk az algoritmus hibatűrését. Többek között a churn (csúcsok folyamatosan távoznak és érkeznek) és a tömeges távozás és érkezés hatását vizsgáltuk. Javasoltunk egy kiegészítést az algoritmushoz, amely szerint az újonnan csatlakozott csúcsokat csak egy kis késleltetés után vesszük figyelembe az osztályozásban (ti. amikor már valamennyire konvergáltak).

7. Topológia konstrukció a T-Man algoritmussal

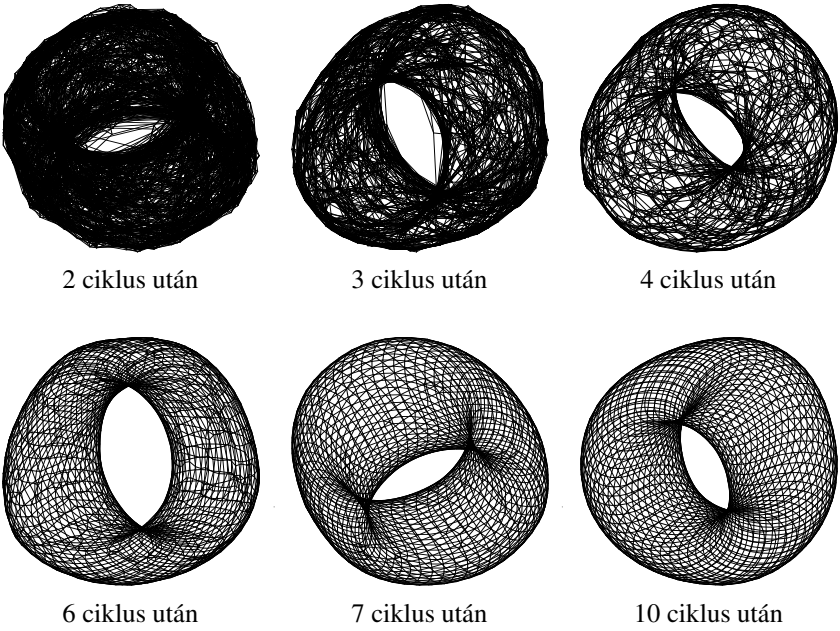
A probléma. Az átfedőhálózatok központi szerepet játszanak az elosztott rendszerek működésében mint elosztott adatstruktúrák, vagy kommunikációs hálózatok. Számos típusú átfedőhálózat létezik, és mindegyik hálózat létrehozására és fenntartására speciális algoritmusok szolgálnak. Két problémát azonosíthatunk. Egyfelől, az ismert átfedőhálózatok algoritmusai a topológia javítására, hosszú távú fenntartására fókuszálnak, viszont nem megoldott a hálózatok hatékony és gyors létrehozása. Másfelől, egy általános célú algoritmus, amely topológiák széles skáláját tudná létrehozni, számos előnnyel rendelkezne, pl. magát a topológiát is lehetne dinamikusan és adaptívan definiálni.

Általános célú átfedőhálózat-konstruáló algoritmust javasoltunk (T-MAN), ami csak a társ mintavételezésre (pl. NEWSCAST) támaszkodik.

A megoldás. A publikációk amikre építettünk a következők: [15, 16]. A megoldásunk lényege, hogy a társ mintavételezésnél látott módszerhez hasonlóan a csúcsok rendszeresen kicserélik egymással a szomszédlistájukat, így gyűjtve össze azokat a szomszédokat, amelyeket az adott topológia megkövetel. A topológia egy rangsoroló függvénnyel van meghatározva, amely minden i csúcs szempontjából bármely csúcshalmazt rangsorol abból a szempontból, hogy *az adott halmazon belül* i számára mennyire kívánatos mint szomszéd. Fontos, hogy általánosabb fogalomról van szó, mint a céltopológián belüli távolság. A társ mintavételezéssel ellentétben itt nem véletlen szomszédokat választunk, hanem olyanokat, amelyek kívánatosak mint szomszéd, abból a heurisztikus feltevésből kiindulva, hogy azok több kívánatos szomszédot tudnak biztosítani a konvergencia minden fázisában. A rendszer inicializálását véletlen szomszédokkal végezzük el, amihez szükség van a társ mintavételező szolgáltatásra is.

6. algoritmus. T-MAN

```
1:  loop
2:      wait(Δ)
3:      p ← selectPeer(ψ, rank(myDescriptor,view))
4:      sendPush(p, toSend(p, m))
5:
6:  procedure onPush(msg)
7:      sendPull(msg.sender, toSend(msg.sender,m))
8:      onPull(msg)
9:  procedure onPull(msg)
10:     view.merge(msg.buffer)
11:
12: procedure toSend(p, m)
13:     buffer ← (MyDescriptor)
14:     buffer.append(view)
15:     buffer ← rank(p,buffer)
16:     return buffer.subList(1, m)
```



2. ábra. Egy tórusz fejlődése a T-MAN algoritmus futása során.

Paraméterek beállítása. Az algoritmus elemzése során arra a következtetésre jutottunk, hogy a legtanácsosabb a legközelebbi ismert szomszédot választani a csere céljára, kombinálva egy tabulistával, amelyen az elmúlt néhány iteráció szomszédai szerepelnek. Emellett egy elméleti modell segítségével amellett érveltünk, hogy a helyben tárolt, összegyűjtött szomszédok számát nem kell korlátozni, mert a hálózat méretében logaritmikus tárhelyigény lép csak fel.

A kísérleti kiértékelés azt támasztja alá, hogy a kívánt topológiát a hálózat méretének logaritmusával arányos időn belül előállítja az algoritmus. Ezen felül az algoritmus hibatűrése is kiváló.

Kiértékelés. Az algoritmust kiegészítettük néhány gyakorlati szempontból fontos részlettel, pl. indító és leállító mechanizmussal. Az algoritmust két topológián vizsgáltuk: a rendezett gyűrű, és a bináris fa topológiákon. A szimulációs kísérleteink egyértelműen azt támasztják alá, hogy a konvergenciához szükséges idő a hálózat méretének logaritmusával nő. A T-MAN algoritmus robosztus az üzenetvesztésre és késleltetésre, ill. csúcsok távozása sem jelent problémát.

dc_565_12

8. T-CHORD: a Chord átfedőhálózat hidegindítása

A probléma. Az átfedőhálózatok egyik gyakori alkalmazása az elosztott hash táblák implementációja (distributed hash table (DHT)), ahol egy adott kulcsért felelős csúcs hatékony elérését teszik lehetővé. A CHORD elosztott hash tábla [17] átfedőhálózata egy gyűrűből, és húrokból áll. Leegyszerűsítve: minden csúcsból $O(\log N)$ számú húr indul ki (N a hálózat mérete) exponenciálisan növekvő távolságokra mutatva a csúcstól. Ez a struktúra $O(\log N)$ lépésű keresést tesz lehetővé. A probléma amit vizsgálunk az ennek a hálózatnak a hidegindítása véletlen hálózatból kiindulva.

A T-MAN algoritmust adaptáltuk a klasszikus CHORD átfedőhálózat [17] gyors hidegindítására. A kiértékelés alátámasztotta hogy a javasolt algoritmus a gyakorlatban hatékony és robosztus.

A megoldás. A publikációk amikre építettünk a következők: [15, 18]. Az ötlet egyszerűsített lényege, hogy a T-MAN algoritmussal rendezett gyűrűt hozunk létre, de közben a húrokat is létrehozzuk, mégpedig azt felhasználva, hogy a konvergencia során meglátogatott szomszédok éppen olyan eloszlással rendelkeznek a csúcstól való távolságot tekintve, mint a keresett húrok, így nagy valószínűséggel további költségek nélkül majdnem minden húr-hely betölthető. Az alapötlet egy olyan variánsát is javasoltuk, ahol a húrokat úgy választjuk ki, hogy egy adott húr-hely betöltésére a több lehetséges jelöltből a legkisebb késleltetésű kapcsolattal rendelkező jelöltet használjuk fel.

Kiértékelés. A kiértékelést szimulációban végeztük el, ahol igazoltuk, hogy az általunk létrehozott elosztott hash tábla minősége mindenben megfelel a követelményeknek, és a rendezett gyűrű létrehozásával megegyező költséggel létrehozható.

9. Általános átfedőhálózat hidegindítás

A probléma. A korábban tárgyalt pletyka komponensek (társ mintavételezés, aggregáció, szeletelés, stb.) mindegyike felfogható egy egyszerű középréteg szolgáltatásnak, amelyek egymásra épülnek. A fedőhálózat konstruálásnak ebbe a keretbe illesztése az ilyen moduláris rendszerekben újabb funkciókat és alkalmazásokat tenne lehetővé. Ehhez egyfelől általánosítani érdemes a korábban tárgyalt hidegindító algoritmusokat, másfelől hidegindító szolgáltatásként kell definiálni azokat a moduláris pletyka architektúra keretei között.

A T-MAN algoritmust adaptáltuk bármely prefix alapú elosztott hash tábla gyors hidegindítására. Emellett demonstráltuk, hogyan lehet pletyka komponensekből komplex rendszereket építeni.

A megoldás. A publikáció amire építettünk a következő: [19]. A korábbi T-CHORD algoritmust általánosítottuk bármilyen prefix alapú átfedőhálózat előállítására. A prefix alapú hálózatokban az azonosítók egy véges ábécé feletti sorozatok, közelebbről valamely számrendszerben ábrázolt számok. A CHORD hálózatban található húrok helyett egy prefix-táblát építünk, aminek a segítségével olyan útvonalválasztó algoritmus tervezhető, amely garantálja, hogy minden lépésben az aktuális csúcs azonosítója legalább eggyel hosszabb közös prefixszel rendelkezik a célcsúccsal (pl. [20]).

Architektúra. A mellékelt ábrán illusztráljuk az eddig tárgyalt pletyka komponensek egymásra épülését. Fontos megjegyezni, hogy ezek az algoritmusok azonos kommunikációs sémát és feltevéseket használnak, így minden moduláris rendszer örökli az egyes komponensek hibatűrését, ill. a pletyka üzenetek költséghatékonyan kombinálhatók.

3. ábra. A javasolt architektúra rétegei.

Hivatkozások

[1] Demers, A., Greene, D., Hauser, C., Irish, W., Larson, J., Shenker, S., Sturgis, H., Swinehart, D., Terry, D.: Epidemic algorithms for replicated database maintenance. In: Proceedings of the 6th Annual ACM Symposium on Principles of Distributed Computing (PODC'87), Vancouver, British Columbia, Canada, ACM Press (August 1987) 1–12

[2] San Miguel, M., Johnson, J., Kertesz, J., Kaski, K., Díaz-Guilera, A., MacKay, R., Loreto, V., Érdi, P., Helbing, D.: Challenges in complex systems science. The European Physical Journal Special Topics **214**(1) (2012) 245–271

[3] Montresor, A., Jelasity, M.: Peersim: A scalable P2P simulator. In: Proceedings of the 9th IEEE International Conference on Peer-to-Peer Computing (P2P 2009), Seattle, Washington, USA, IEEE (September 2009) 99–100 extended abstract.

[4] Bavier, A., Bowman, M., Chun, B., Culler, D., Karlin, S., Muir, S., Peterson, L., Roscoe, T., Spalink, T., Wawrzoniak, M.: Operating system support for planetary-scale services. In: Proceedings of the First Symposium on Network Systems Design and Implementation (NSDI'04), USENIX (2004) 253–266

[5] Jelasity, M., Voulgaris, S., Guerraoui, R., Kermarrec, A.M., van Steen, M.: Gossip-based peer sampling. ACM Transactions on Computer Systems **25**(3) (August 2007) 8

[6] Jelasity, M., Kowalczyk, W., van Steen, M.: Newscast computing. Technical Report IR-CS-006.03, Vrije Universiteit Amsterdam, Department of Computer Science, Amsterdam, The Netherlands (November 2003)

[7] Jelasity, M., Guerraoui, R., Kermarrec, A.M., van Steen, M.: The peer sampling service: Experimental evaluation of unstructured gossip-based implementations. In Jacobsen, H.A., ed.: Middleware 2004. Volume 3231 of Lecture Notes in Computer Science., Springer-Verlag (2004) 79–98

[8] Jelasity, M., Montresor, A., Babaoglu, O.: Gossip-based aggregation in large dynamic networks. ACM Transactions on Computer Systems **23**(3) (August 2005) 219–252

[9] Jelasity, M., Montresor, A.: Epidemic-style proactive aggregation in large overlay networks. In: Proceedings of The 24th International Conference on Distributed Computing Systems (ICDCS 2004), Tokyo, Japan, IEEE Computer Society (2004) 102–109

[10] Montresor, A., Jelasity, M., Babaoglu, O.: Robust aggregation protocols for large-scale overlay networks. In: Proceedings of The 2004 International Conference on Dependable Systems and Networks (DSN), Florence, Italy, IEEE Computer Society (2004) 19–28

[11] Bianchini, M., Gori, M., Scarselli, F.: Inside pagerank. ACM Transactions on Internet Technology **5**(1) (2005) 92–128

[12] Jelasity, M., Canright, G., Engø-Monsen, K.: Asynchronous distributed power iteration with gossip-based normalization. In Kermarrec, A.M., Bougé, L., Priol, T., eds.: Euro-Par 2007. Volume 4641 of Lecture Notes in Computer Science., Springer-Verlag (2007) 514–525

[13] Lubachevsky, B., Mitra, D.: A chaotic asynchronous algorithm for computing the fixed point of a nonnegative matrix of unit radius. Journal of the ACM **33**(1) (January 1986) 130–150

[14] Jelasity, M., Kermarrec, A.M.: Ordered slicing of very large-scale overlay networks. In: Proceedings of the 6th IEEE International Conference on Peer-to-Peer Computing (P2P 2006), Cambridge, UK, IEEE Computer Society (September 2006) 117–124

[15] Jelasity, M., Montresor, A., Babaoglu, O.: T-Man: Gossip-based fast overlay topology construction. Computer Networks **53**(13) (2009) 2321–2339

[16] Jelasity, M., Babaoglu, O.: T-Man: Gossip-based overlay topology management. In Brueckner, S.A., Di Marzo Serugendo, G., Hales, D., Zambonelli, F., eds.: Engineering Self-Organising Systems: Third International Workshop (ESOA 2005), Revised Selected Papers. Volume 3910 of Lecture Notes in Computer Science., Springer-Verlag (2006) 1–15

[17] Stoica, I., Morris, R., Karger, D., Kaashoek, M.F., Balakrishnan, H.: Chord: A scalable peer-to-peer lookup service for internet applications. In: Proceedings of the 2001 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications (SIGCOMM), San Diego, CA, ACM, ACM Press (2001) 149–160

[18] Montresor, A., Jelasity, M., Babaoglu, O.: Chord on demand. In: Proceedings of the 5th IEEE International Conference on Peer-to-Peer Computing (P2P 2005), Konstanz, Germany, IEEE Computer Society (August 2005) 87–94

[19] Jelasity, M., Montresor, A., Babaoglu, O.: The bootstrapping service. In: Proceedings of the 26th International Conference on Distributed Computing

Systems Workshops (ICDCS WORKSHOPS), Lisboa, Portugal, IEEE Computer Society (2006) International Workshop on Dynamic Distributed Systems (IWDDS).

[20] Rowstron, A., Druschel, P.: Pastry: Scalable, decentralized object location and routing for large-scale peer-to-peer systems. In Guerraoui, R., ed.: Middleware 2001. Volume 2218 of Lecture Notes in Computer Science., Springer-Verlag (2001) 329–350