UNIVERZA V LJUBLJANI
FAKULTETA ZA RAČUNALNIŠTVO IN INFORMATIKO

Marko Novak

Odkrivanje virov v sistemih Grid na osnovi porazdeljene zgoščene tabele

MAGISTRSKO DELO

Mentor: prof. dr. Borut Robič

Ljubljana, 2009

Univerza
v Ljubljani

Fakulteta za računalništvo
in informatiko

Tržaška 25
1000 Ljubljana, Slovenija
telefon: 01 476 84 11
faks: 01 426 46 47
www.fri.uni-lj.si
e-mail: dekanat@fri.uni-lj.si

Št.: 129-MAG-RI/2009
Datum: 29. 6. 2009

Marko NOVAK, univ. dipl. inž. rač. in inf.

Ljubljana

Fakulteta za računalništvo in informatiko Univerze v Ljubljani izdaja naslednjo magistrsko nalogo

Naslov naloge: **Odkrivanje virov v sistemih Grid na osnovi porazdeljene zgoščene tabele**

Resource discovery in Grid systems, based on Distributed Hash Table

Tematika naloge:
Zasnujte porazdeljeni sistem, ki bo omogočal odkrivanje virov v sistemih Grid. Sistem naj temelji na strukturiranih tehnologijah "enak z enakim", natančneje, na principu porazdeljene zgoščene tabele. Sistem naj omogoča iskanje virov tako z ujemalnimi poizvedbami, pri katerih se uporablja samo operator enakosti, kot tudi z območnimi poizvedbami, pri katerih se uporabljajo vsi relacijski operatorji. Poleg tega naj podpira večparametrske poizvedbe, pri katerih lahko ena sama poizvedba vsebuje več iskalnih pogojev. Sistem naj generira čim manj mrežnega prometa in naj bo odporen na izpade vozlišč. Določite in opišite postopek za objavljanje virov v sistemu ter postopek za njihovo odkrivanje s pomočjo zgoraj navedenih poizvedb. Razvijte prototipno implementacijo sistema. Pri tem uporabite katero od znanih izvedb porazdeljene zgoščene tabele (Chord, Pastry, CAN, Tapestry), ki jo po potrebi tudi razširite. Sistem preizkusite ter določite njegove prednosti in slabosti.

Mentor:
prof. dr. Borut Robič

UNIVERZA V LJUBLJANI • Fakulteta za računalništvo in informatiko

Dekan:
prof. dr. Franc Solina

Besedilo je oblikovano z urejevalnikom besedil LaTeX.

IZJAVA O AVTORSTVU

magistrskega dela

Spodaj podpisani Marko Novak,

z vpisno številko 63000232,

sem avtor magistrskega dela z naslovom:

Odkrivanje virov v sistemih Grid na osnovi porazdeljene zgoščene tabele

S svojim podpisom zagotavljam, da:

- sem magistrsko delo izdelal samostojno pod mentorstvom
  prof. dr. Boruta Robiča
- so elektronska oblika diplomskega dela, naslov (slov., angl.), povzetek (slov., angl.) ter ključne besede (slov., angl.) identični s tiskano obliko diplomskega dela
- soglašam z javno objavo elektronske oblike diplomskega dela v zbirki "Dela FRI".

V Ljubljani, dne 17.5.2009

Zahvala

Prof. dr. Borutu Robiču in dr. Marjanu Šterku za pomoč pri izdelavi magistrske naloge in koristne nasvete pri raziskovalnem delu.

Posebna zahvala gre Jaki Močniku, ne samo za koristne nasvete pri realizaciji sistema, ki je ga opisujem v magistrskem delu, ampak ker ima prav on največ zaslug pri mojem izpopolnjevanju v veščinah programerske obrti. To je pomembno znanje, ki pa ga je zelo težko pridobiti brez pravega mentorja, zaradi česar je moja hvaležnost toliko večja.

Gregorju Pipanu za potrpljenje, ki ga je imel z menoj v teh treh letih magistrskega študija ter za trud, ki ga vlaga v to, da ohranja delo v podjetju XLAB pestro, vzdušje pa sproščeno.

Sodelavcem v podjetju XLAB za ustvarjanje raziskovalnemu delu prijaznega okolja. Predvsem Alešu, Borisu, Juretu, Maticu, Mihi in Primožu, brez katerih bi bila kosila v preteklem letu zelo dolgočasna.

Cimrom, zdajšnjim (Luka, Nejc, Miha, Edo) in preteklim (Jana, Sabina, Matjaž), ki so poskrbeli, da je bilo bivanje v Ljubljani precej bolj vzdržno.

Predvsem pa domačim (mama Nada, sestra Nika, brat Simon) ter sorodnikom, ki so mi nenehno v opomin, da v življenju obstajajo tudi pomembnejše reči od računalnikov. Zaradi njih se vedno rad vračam domov, pa čeprav samo za vikend.

Hvala vsem!

Domačim.

Kazalo

Slike

Tabele

Izpisi izvorne kode

Povzetek

Odkrivanje virov je ena izmed ključnih funkcionalnosti sistemov Grid, saj se uporablja pri množici drugih storitev. Cilj vsakega sistema za odkrivanje virov je zbiranje podatkov o vseh virih, ki se nahajajo v sistemu Grid in streženje teh podatkov zainteresiranim uporabnikom in aplikacijam. Uporaba je preprosta: storitvi za odkrivanje virov podamo poizvedbo, ta pa nam vrne seznam lokacij virov, ki ustrezajo pogojem v poizvedbi.

V pričujočem delu predstavimo porazdeljeni sistem za odkrivanje virov v sistemih Grid, ki temelji na strukturiranih tehnologijah "enak z enakim", natančneje, na porazdeljeni zgoščeni tabeli. Sistem omogoča iskanje virov tako z ujemalnimi kot tudi z območnimi poizvedbami. Podpora območnim izvedbam je ključnega pomena v sistemih Grid, saj je večina poizvedb, ki se uporabljajo v njih, tega tipa. Slabost večine obstoječih sistemov za odkrivanje virov na osnovi strukturiranih tehnologij "enak z enakim" je zelo potratna implementacija podpore za območne izvedbe. Pri našem sistemu ni tako, saj je količina mrežnega prometa, ki se generira pri razreševanju območnih poizvedb, enaka tisti pri razreševanju ujemalnih poizvedb.

Učinkovitejša je tudi podpora za večparametrske poizvedbe, pri katerih je ena sama poizvedba sestavljena iz več iskalnih pogojev. Tudi tu naš sistem generira manj mrežnega prometa od sorodnih sistemov za odkrivanje virov.

Zaradi omenjenih lastnosti ter zaradi boljše odpornosti na izpade vozlišč ima naš sistem precej boljšo razširljivost od ostalih sistemov na osnovi "enak z enakim" in je zato primernejši za uporabo v velikih sistemih Grid.

Ključne besede:

sistemi Grid, omrežno računalništvo, odkrivanje virov, porazdeljena zgoščena tabela, strukturirani sistemi "enak z enakim"

Abstract

Resource discovery is one of the key functionalities of the Grid systems, since it is used by a variety of different services. The main goal of every resource discovery system is to collect the data about all the Grid resources and to serve those data to all the users and applications that are interested in them. The usage of the resource discovery service is simple: we provide it a search query as an input and it returns us the list of locations of all the resources which satisfy all the conditions provided by the query.

This work presents a distributed system that is used for resource discovery in Grid systems. The system is based on structured peer-to-peer technologies; more precisely, on the Distributed Hash Table. It enables us to search for the Grid resources using exact-match queries as well as range queries. The support for range queries is very important in Grid systems since those queries are the most frequently used. The weakness of the majority of structured peer-to-peer resource discovery systems is the inefficient implementation of the support for range queries. In our system, on the other hand, the amount of network traffic that is generated during the resolution of range queries is the same as with exact-match queries.

Additionally, the support for multi-attribute queries (i.e., the queries that contain multiple search conditions) is also more efficient since the amount of network traffic when resolving multi-attribute queries is also lower than with the other resource-discovery systems.

Due to both the properties mentioned above and due to the better resilience to node failures, our system has better scalability than the rest of the resource-discovery systems that are based on a structured peer-to-peer system. As a consequence, its performance in large Grid systems is better than the other systems.

Key words:

Grid systems, Grid computing, resource discovery, distributed hash table (DHT), structured peer-to-peer (P2P) systems

1 Uvod

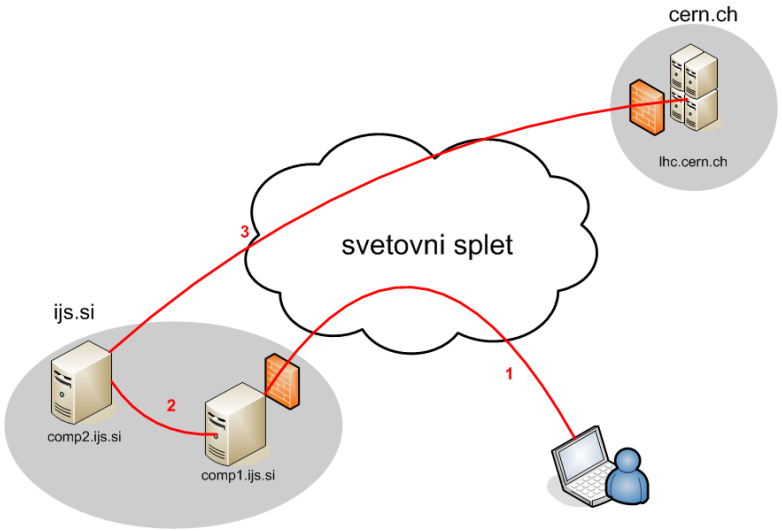
V uvodnem poglavju najprej predstavimo sisteme Grid in pojem navidezne organizacije. Sledi primerjava sistemov Grid s sorodnimi tehnologijami za porazdeljeno računanje ter opis primerov uporabe. Poglavje zaključimo z opisom in formalno definicijo problema odkrivanja virov.

1.1 Sistemi Grid

Omrežno računalništvo (angl. Grid computing) spada med mlajša področja računalniške znanosti. Pojavilo se je konec 1990-ih let z objavo dela [1]. V njem sta Ian Foster in Carl Kesselman predstavila pojem računalniškega omrežja, oziroma sistema Grid (angl. Grid system), s čimer sta hotela poudariti, da bi morala biti infrastruktura za zagotavljanje računske moči podobna omrežju za distribucijo električne energije. To namreč na konsistenten in transparenten način zagotavlja dostop do elektrike ne glede na njen vir. Žal trenutno stanje v porazdeljenih sistemih še zdaleč ni na nivoju električnih omrežij: dostop do računalnikov na geografsko ločenih lokacijah je navadno zelo zapleten, saj moramo prečkati več različnih administrativnih domen (administrativna domena je množica računalnikov, usmerjevalnikov in omrežij, ki jih upravlja en administrativni urad). Razlog za to je potreba po varnosti. Zaradi preprečevanja vdorov nepooblaščenih uporabnikov v administrativne domene, morajo biti te zaščitene z vrsto varnostnih mehanizmov (gesla, požarni zidovi, itd.). Prav ti varnostni mehanizmi najbolj zapletejo dostop do oddaljenih računalnikov.

Vzemimo za primer scenarij, ki je prikazan na sliki 1.1. Luka je zaposlen kot teoretični fizik na inštitutu Jožef Štefan. Njegov cilj je razviti teorijo o obnašanju osnovnih delcev pri trkih, pri čemer si pomaga s preučevanjem podatkov iz pospeševalnika CERN v Švici. Ti podatki so shranjeni na strežniku *lhc*, na žalost pa je dostop do njega vse prej kot preprost. Zaradi zaupnosti podatkov, ki so shranjeni na njem, želi CERN karseda zmanjšati možnost vdora nepooblaščenih uporabnikov. To stori s postavitvijo t.i. požarnega zidu (angl. firewall), ki omogoča vzpostavitev povezave do strežnika samo računalnikom z ustreznim naslovom IP. Eden izmed takih računalnikov je tudi *comp2*, Lukov delovni računalnik na inštitutu. Iz njega se lahko prijavi na *lhc* in izvaja simulacije nad podatki. Problem se pojavi, ko želi Luka

dostopati do *lhc* od doma. Z domačega računalnika se namreč ne more neposredno povezati v domeno *cern.ch*. Lahko se poveže le posredno, preko računalnika *comp2*. Tu pa se pojavi dodaten problem: *comp2* je dostopen samo znotraj domene *ijs.si*. Da lahko Luka dostopa do strežnika *lhc* od doma, se mora tako najprej prijaviti na enega izmed strežnikov znotraj domene *ijs.si*, ki so javno dostopni (tak je npr. *comp1*). S tega se nato poveže na delovni računalnik (*comp2*), preko njega pa na *lhc*. To prijavljanje je zelo zamudno, saj je treba za vsak računalnik posebej vnašati uporabniško ime in geslo. Za uporabnika bi bilo precej ugodneje, če bi obstajali mehanizmi, ki bi mu omogočali dostop do poljubnega strežnika (do katerega ima pooblastila) z eno samo prijavo.



Slika 1.1: Primer dostopa do oddaljenih računalnikov preko več vozlišč.

Navedeni problem je učinkovito rešen s pomočjo omrežnega računalništva. Sistemi Grid, kot jih navaja [2], so namreč definirani kot sistemi za nadzorovano deljenje virov in porazdeljeno reševanje problemov v dinamičnih navideznih organizacijah, ki lahko obsegajo več administrativnih domen. Deljenje virov, ki je omenjeno v zgornji definiciji, ne zajema samo izmenjave datotek, ampak tudi dostop do oddaljenih računalnikov, programske opreme, podatkov in ostalih virov, ki jih potrebujemo pri skupinskem reševanju problemov. Sem spada, med drugim, tudi koncept enkratne prijave (angl. single sign-on).

1.1.1 Navidezne organizacije

Izraz navidezna organizacija (angl. virtual organization), ki je uporabljen v definiciji sistemov Grid, označuje množico posameznikov in organizacij, ki jih zajemajo pravila za deljenje virov. Glavni namen navideznih organizacij je omogočiti geo-

grafsko ločenim skupinam, ki si prizadevajo doseči nek skupen cilj, sodelovanje in nadzorovano deljenje virov (ponudniki in odjemalci morajo nedvoumno in natančno določiti deljene vire in pogoje za delitev). Za lažjo ponazoritev pojma si oglejmo nekaj primerov navideznih organizacij:

- ponudniki računskih storitev, storitev za shranjevanje podatkov ter razni svetovalci, ki jih najame avtomobilska družba, da izberejo najboljšo lokacijo za postavitev nove tovarne,
- člani industrijskega konzorcija, ki načrtujejo izdelavo novega letala,
- ekipa za reševanje kriznih situacij, skupaj z ustreznimi podatkovnimi bazami in simulacijami,
- raziskovalne skupine, ki sodelujejo pri določenem znanem poskusu.

Navedeni primeri se razlikujejo med seboj po namenu, velikosti, trajanju, strukturi, številu udeležencev, vrsti uporabljenih virov in v marsičem drugem. Kljub temu pa imajo precej skupnih lastnosti:

- V vseh primerih imamo opraviti z večjim številom bolj ali manj zaupanja vrednih udeležencev, ki si med seboj delijo vire, da bi dosegli skupno zastavljeni cilj.
- Deljenje virov ni obvezno: vsak udeleženec se sam odloči, katere vire bo dal v souporabo in pod katerimi pogoji.
- Tudi uporabniki virov lahko določajo, kakšne morajo biti lastnosti virov, s katerimi so pripravljeni delati.
- Da bi zagotovili učinkovito upravljanje z viri, je treba vzpostaviti mehanizme za identifikacijo, avtentikacijo in pooblaščanje uporabnikov virov.
- Deljenje virov je dinamičen proces, ki se skozi čas spreminja. Spreminjajo se viri, ki so na voljo, uporabniki, ki smejo dostopati do njih, in način, na katerega smejo dostopati.
- Zaradi dinamične narave deljenja virov potrebujemo mehanizme za njihovo odkrivanje in karakterizacijo.
- Pojavi se tako potreba po raznovrstnih modelih komunikacije, od strežniškega do modela enak z enakim, kot tudi potreba po prefinjenih in natančnih metodah nadzora nad uporabo deljenih virov, ki vključujejo pooblaščanje ter vpeljavo lokalnih in globalnih pravil.
- Isti viri se uporabljajo na več različnih načinov, odvisno od pravic uporabnika in naloge, ki jo opravlja.

Trenutne tehnologije v porazdeljenem računalništvu rešujejo samo nekatere izmed pogojev za vzpostavitev navidezne organizacije. Internetne tehnologije rešujejo problem komunikacije in izmenjave podatkov med računalniki, ne pa tudi problema nadzorovane uporabe virov na različnih računalnikih. Porazdeljene objektno usmerjene tehnologije, kot sta Enterprise Java in CORBA, pa tipično omogočajo deljenje virov samo znotraj ene same administrativne domene. Trenutne tehnologije bodisi ne morejo gostiti več različnih tipov virov bodisi ne omogočajo prilagodljivega in nadzorovanega deljenja virov, potrebnega za vzpostavitev navidezne organizacije.

Ta problem v celoti rešujejo sistemi Grid. Raziskovalci na področju omrežnega računalništva so v preteklih letih razvili vrsto protokolov, storitev in orodij, ki rešujejo probleme, povezane z vzpostavitvijo navideznih organizacij. Mednje spadajo tudi:

- varnostni mehanizmi, ki podpirajo upravljanje s priporočili (angl. credentials), ko navidezne organizacije zajemajo več inštitucij,
- protokoli za zajemanje informacij in storitve, ki nudijo informacije o virih, organizacijah in storitvah,
- storitve za upravljanje s podatki, ki odkrivajo in prenašajo podatke med sistemi za shranjevanje in aplikacijami.

Pobudniki sistemov Grid verjamejo, da bo koncept navideznih organizacij močno spremenil način uporabe računalnikov za reševanje problemov, podobni kot je svetovni splet spremenil način izmenjave informacij.

1.1.2 Primerjava s sorodnimi tehnologijami za porazdeljeno računanje

Cilj omrežnega računalništva je povezovanje velikega števila računalnikov za potrebe reševanja tehničnih in znanstvenih problemov, ki zahtevajo veliko procesorsko moč oziroma dostop do velike količine podatkov. Pri tem začetni problem razdelimo na več podproblemov (teh podproblemov je lahko tudi več tisoč), programska oprema pa skrbi za dodelitev posameznega podproblema ustreznemu računalniku.

V tem pogledu so sistemi Grid podobni računalniškim skupkom (angl. clusters). Poleg tega je za oba pristopa značilno, da so vozlišča sestavljena iz splošno dostopnih računalniških komponent. Ker je cena takih komponent nekajkrat manjša od cene specializiranih komponent, je razmerje med ceno in zmogljivostjo pri skupkih in sistemih Grid precej ugodnejše kot pri specializiranih superračunalnikih.

Sistemi Grid pa se v marsičem razlikujejo od računalniških skupkov. Medtem ko so vozlišča (angl. nodes) pri skupkih navadno zbrana na eni sami lokaciji in so homogena (se ne razlikujejo bistveno v strojni opremi in operacijskem sistemu), so pri sistemih Grid porazdeljena širom po svetu in so heterogena. Poleg tega so vozlišča v sistemih Grid tudi šibko sklopljena (angl. loosely coupled), zaradi česar izpad posameznih vozlišč ne ogrozi delovanja celotnega sistema. Ta lastnost je v

sistemih Grid ključnega pomena, saj se njihova topologija neprestano menja. Zelo malo vozlišč je v sistemu neprekinjeno prisotnih dlje časa, večina se jih priključi za največ par ur.

Podobno kot pri skupkih, tudi v omrežnem računalništvu ni pravil glede velikosti. Sistemi Grid so lahko zelo majhni, obsegajo samo skupino računalnikov znotraj ene same administrativne domene, lahko pa se raztezajo čez različna omrežja in vključujejo veliko število administrativnih domen. V povprečju pa so sistemi Grid precej večji od skupkov. Razlogi za to so:

- skupki so sestavljeni iz vozlišč, ki se nahajajo na isti lokaciji, medtem ko se v sisteme Grid vključujejo računalniki s celega sveta,
- sistemi Grid delujejo na principu prostovoljstva: pri večini današnjih skupkov so vozlišča namenjena izključno za reševanje problemov, na njih se praviloma ne izvajajo uporabniški programi. Pri sistemih Grid temu ni tako: vanje se praviloma priključujejo uporabniki, ki želijo ponuditi neizkoriščen procesorski čas svojih osebnih računalnikov za reševanje problemov. Seveda ta čas navadno predstavlja le del celotnega procesorskega časa računalnika. Preostanek procesorskega časa je namenjen za izvajanje uporabniških programov. V sisteme Grid mora biti tako vključenih precej več računalnikov, če želimo dobiti enako računsko moč kot pri skupkih namenskih računalnikov,
- v sistemih Grid so vozlišča šibko sklopljena: šibka sklopljenost sistemov omogoča boljšo razširljivost (angl. scalability) zaradi medsebojne neodvisnosti vozlišč. Tako lahko povežemo med seboj večje število vozlišč.

1.1.3 Uporaba

Prvi predstavniki sistemov Grid so se pojavili v letih 1997 (distributed.net) in 1999 (SETI@home). Ti sistemi so izkoriščali procesorsko moč velikega števila osebnih računalnikov, ki v danem trenutku niso bili v uporabi, za reševanje računsko zahtevnih problemov. Na ta način so lahko z malo stroški obdelali zelo veliko količino podatkov.

Z leti popularnost omrežnega računalništva vedno bolj narašča. Ključni razlogi za to so nizka cena, visoka stopnja dostopnosti velikega števila računskih virov ter virov za shranjevanje podatkov ter neprestano naraščanje števila računalnikov, ki so povezani v svetovni splet.

V zadnjem času se sistemi Grid uporabljajo za reševanje problemov na različnih področjih: fiziki, kemijskih in biotehničnih znanostih, strojništvu, elektrotehniki, računalništvu, ekonomiji in drugih. Med probleme, ki jih rešujemo s pomočjo sistemov Grid, spadajo: odkrivanje in testiranje novih zdravil, ekonomsko modeliranje, analize potresnih valov, simulacije fizikalnih pojavov, iskanje znanja v podatkih, podpora e-tgovanju in spletnim storitvam.

Čeprav je omrežno računalništvo še v zgodnjih fazah, se je že izoblikovala množica storitev sistemov Grid:

- Računske storitve (angl. Computing Services). Sistem Grid preko posrednika virov zagotovi ustrezne vire za skupno in porazdeljeno izvajanje računskih nalog. Omrežje, ki zagotavlja take storitve, pogosto imenujemo tudi računsko omrežje (angl. Computation Grid).
- Podatkovne storitve (angl. Data Services). Sistem Grid preko posrednika virov zagotavlja varen dostop do porazdeljenih podatkovnih virov. Hkrati skrbi za porazdeljeno shranjevanje podatkov po virih, ki so na voljo. Sisteme Grid, ki zagotavljajo take storitve, navadno imenujemo podatkovna omrežja (angl. Data Grid).
- Aplikacijske storitve (angl. Application Services). Te so osredotočene na upravljanje aplikacij in zagotavljajo dostop do oddaljene programske opreme.
- Informacijske storitve (angl. Information Services). Te so odgovorne za pridobivanje in predstavitev podatkov, pridobljenih preko aplikacijskih, računskih in podatkovnih storitev sistema Grid. Sem spadajo Splet in spletni portali.

1.2 Odkrivanje virov v sistemih Grid

1.2.1 Terminologija

Za boljše razumevanje problema odkrivanja virov moramo najprej razumeti kaj vse spada k virom. V [3] je podana naslednja definicija vira:

Definicija 1. Vir je katerikoli izvor podpore oziroma pomoči, ki ga komponenta v mrežnem okolju lahko izkorišča v poljubnem trenutku.

Primeri virov so: datoteke, procesorski čas, pomnilnik, merilne naprave, krmilne naprave, strojna oprema (tiskalniki,...) in drugi.

Definirajmo sedaj še izraz "odkrivanje virov":

Definicija 2. Odkrivanje virov je proces iskanja virov, ki ustrezajo množici pogojev, podanih v poizvedbi.

Proces odkrivanja virov vključuje 3 glavne akterje:

- **ponudnik virov**: katerakoli mrežna entiteta, ki omogoča deljenje svojih virov,
- **odjemalec virov**: katerakoli mrežna entiteta, ki uporablja deljene vire,
- **komponenta za odkrivanje virov**: storitev, ki vrača lokacije ustreznih virov kot odziv na poizvedbe, ki jih pošiljajo odjemalci.

1.2.2 Opis

Cilj vsakega sistema za deljenje virov je zbiranje podatkov o virih v sistemu in streženje teh podatkov zainteresiranim uporabnikom in aplikacijam. Pri teh sistemih je ključnega pomena storitev za odkrivanje virov (angl. resource discovery). Ta omogoča iskanje virov, ki se nahajajo v različnih administrativnih domenah, na podlagi seznama vnaprej definiranih atributov. Uporaba je preprosta: storitvi podamo seznam iskalnih pogojev, ta pa nam vrne seznam lokacij virov, ki ustrezajo iskalnim pogojem.

Iskanje virov je ena izmed ključnih funkcionalnosti sistemov Grid, saj se uporablja pri množici drugih storitev. Ko uporabnik, na primer, pošlje opravilo sistemu Grid, določi množico pogojev (količina prostega pomnilnika, diskovnega prostora, vrsta operacijskega sistema itd.), ki jim morajo ustrezati računalniki, na katerih se bo opravilo izvajalo.

Viri v sistemih Grid so zelo dinamični, njihove lastnosti (npr. obremenjenost CPU, razpoložljiv pomnilniški prostor in diskovni prostor, razpoložljiva pasovna širina mrežnih povezav) se lahko zelo spreminjajo skozi čas. Aplikacije opišejo vire, ki jih potrebujejo, s pomočjo večparametrskih poizvedb (angl. multi-attribute query). Te zajemajo množico logičnih pogojev, ki jim morajo ustrezati atributi iskanih virov (npr. zahtevana procesorska moč, količina prostega pomnilnika).

Odkrivanje virov pa ni prisotno samo v sistemih Grid, temveč tudi v sistemih "enak z enakim" (angl. peer-to-peer, P2P)[4]. To so porazdeljeni sistemi, sestavljeni iz med seboj povezanih enakovrednih gostiteljev (angl. peers), ki se samostojno organizirajo v omrežje. To omrežje je odporno na izpade vozlišč in povezav, se učinkovito prilagaja spremembam v topologiji omrežja, hkrati pa vzdržuje zadovoljivo povezanost in učinkovitost sistema brez prisotnosti centralnega strežnika. V sistemih P2P se odkrivanje virov uporablja predvsem pri deljenju datotek (npr. Gnutella, KaZaA) in komunikaciji v realnem času (npr. Skype). Tu so poizvedbe navadno precej preprostejše kot pri sistemih Grid: večparametrske poizvedbe so redke, največ se išče po imenih datotek, po ključnih besedah.

Poglaviten cilj sistemov za deljenje virov, tako v sistemih Grid kot tudi P2P, je izkoriščanje čim večjega števila virov v različnih administrativnih domenah. Obe problemski domeni imata precej skupnih značilnosti: npr. heterogenost virov (tako v sistemih Grid kot tudi P2P obstaja veliko število različnih vrst virov), dinamičnost (viri v sistemih se neprestano menjajo, spreminjajo se jim atributi). Čeprav obstaja med njima tudi veliko razlik (glej tabelo 1.1), so mnogi mnenja [5, 6], da si bodo sistemi Grid in P2P v prihodnosti postali še bolj podobni in se bodo sčasoma združili v eno samo tehnologijo. Pri tem bodo poskušali slabosti tehnologij Grid odpraviti s pomočjo P2P in obratno:

- z naraščanjem velikosti sistemov Grid postajata centraliziran in hierarhičen način shranjevanja podatkov, ki trenutno prevladujeta v sistemih Grid, vedno

bolj neučinkovita. Glavna razloga za neučinkovitost sta neenakomerna obremenitev vozlišč (pri centraliziranem načinu so podatkovni strežniki veliko bolj obremenjeni kot ostala vozlišča) in prisotnost t.i. osamljene točke odpovedi (angl. single point of failure): ob izpadu katerega izmed ključnih vozlišč se lahko delovanje celotnega sistema zelo poslabša. Omenjeni slabosti bi lahko učinkovito odpravili z uporabo tehnologij P2P. S tem bi izboljšali razširljivost sistemov Grid. To je v današnjih časih ključnega pomena, saj se vanje ne priključujejo samo znanstvene inštitucije, temveč vse bolj tudi običajni uporabniki.

- po drugi strani bodo morali sistemi P2P podpreti velik del funkcionalnosti, ki je že dostopna v sistemih Grid, preden bi jih lahko uporabili pri implementaciji naprednejših aplikacij: naprednejše deljenje virov, zagotavljanje varnosti ter izvajanje bolj zapletenih poizvedb. Prvotni sistemi P2P so se uporabljali predvsem pri deljenju datotek in komunikaciji v realnem času. Potrebovali so samo mehanizme za izvajanje preprostih poizvedb (npr. iskanje po imenih datotek, uporabniških imenih). Kasneje so se začeli uporabljati tudi za naprednejše oblike komuniciranja (npr. VoIP, videokonferenčni sistemi), za realizacijo porazdeljenih podatkovnih baz ter za vojaške namene. V teh sistemih sta varnost in nadzorovano deljenje virov ključnega pomena.

Tabela 1.1: Razlike med sistemi Grid in P2P.

kategorija	Grid	P2P
uporaba	znanstvena srenja	domači uporabniki
oprema	delovne postaje, večprocesorski računalniki, računalniški skupki	namizni računalniki
mreža	hitre mrežne povezave	internetne TCP povezave
administracija	centralizirana oz. hierarhična	porazdeljena
aplikacije	znanstvene aplikacije (npr. obsežne simulacije, analize podatkov)	deljenje datotek, komunikacija v realnem času
velikost	sistemi obsegajo manjše število specializiranih lokacij (srednja velikost)	sistemi obsegajo veliko število namiznih računalnikov
zaščita	visoka	nizka
stopnja zaupanja	znani, preverjeni uporabniki	anonimni, nepreverjeni uporabniki

Iskanje podatkov, ki izpolnjujejo iskalne pogoje, je eden izmed pogosteje obravnavanih problemov pri P2P sistemih. V sistemih Grid pa je ta problem precej bolj kompleksen, saj imamo tu opravka z dinamičnimi atributi virov. Zanje je značilno,

da se njihove vrednosti lahko spreminjajo. Na primer, razpoložljiv prostor na disku se v času spreminja kot posledica dodajanja in brisanja datotek. Iskanje dinamičnih podatkov v porazdeljenih P2P sistemih je precej težje kot iskanje statičnih podatkov. Poleg tega se iskanje virov v sistemih Grid pogosto vrši s pomočjo večparametrskih območnih poizvedb (angl. multi-attribute range queries). To so poizvedbe, ki so sestavljene iz več pogojev, vsak pogoj pa je podan s pomočjo relacijskega operatorja. Z njimi iščemo vire, katerih vrednosti atributov se nahajajo znotraj podanih intervalov. Večparametrske območne poizvedbe so glavna razlika med sistemi Grid in sistemi P2P za deljenje datotek, ki uporabljajo večinoma samo ujemalne poizvedbe (angl. exact match queries).

Tipičen primer večparametrske območne poizvedbe je zahteva za iskanje in zaseganje virov, ki jo izdamo pri procesu izvrševanja opravil. Oglejmo si enačbo 1.1, ki predstavlja poizvedbo za iskanje računskega vira R, pri katerem mora biti frekvenca procesorja vsaj 2 GHz, količina pomnilnika vsaj 512 MB, z uporabljanostjo zadnjih 10 minut največ 30% in količino prostega prostora na disku med 100 in 300 MB. Parametra "Utilization" in "FreeSpace" sta dinamična atributa, ob različnih trenutkih imata lahko različne vrednosti.

$$Q = \{R \in \{R_1, ..., R_N\} | CpuSpeed[R] \geq 2.0GHz \ \textbf{\textit{and}} \ RamSize[R] \geq 512MB \\ \textbf{\textit{and}} \ Utilization10[R] \leq 0.3 \ \textbf{\textit{and}} \ 100MB \leq FreeSpace[R] \leq 300MB\} \quad (1.1)$$

Poizvedbo si predstavljamo kot predikat, ki je sestavljen iz več logičnih pogojev. Pogoji vsebujejo bodisi ujemalne (pri ujemalnih poizvedbah), bodisi relacijske operatorje (pri območnih poizvedbah). Na tem mestu velja še omeniti, da mora obstajati linearna urejenost v domeni vrednosti atributov virov, sicer ne moremo uporabljati območnih poizvedb.

Prvi nestrukturirani sistemi P2P so za iskanje podatkov uporabljali t.i. poplavljanje (angl. flooding). Pri tej metodi iskalec "poplavlja" omrežje z iskalnimi zahtevami, dokler ne najde iskanih podatkov oziroma ni izpolnjen pogoj za zaustavitev. Največja slabost poplavljanja je velika količina mrežnega prometa, ki se generira med iskanjem. Zaradi tega so imeli prvotni nestrukturirani sistemi P2P zelo slabo razširljivost. Problem prevelikega mrežnega prometa so poskušali odpraviti z zasnovo sistemov za indeksiranje podatkov ter z uvedbo strukturiranih sistemov P2P, od katerih jih večina uporablja porazdeljene zgoščene tabele (angl. distributed hash table, DHT). Strukturirani sistemi P2P pri iskanju sicer generirajo manj mrežnega prometa kot nestrukturirani, vendar pa podpirajo samo ujemalne poizvedbe. Ker se v sistemih Grid najpogosteje uporabljajo večparametrske območne poizvedbe, bi bilo treba strukturirane sisteme P2P ustrezno nadgraditi.

1.2.3 Formalna definicija problema

V tem razdelku bomo opisali problem iskanja virov, kot je podan v [7]. Predpostavimo, da vsako vozlišče v sistemu hrani množico podatkovnih elementov (ta množica je lahko tudi prazna). To množico bomo poimenovali lokalno skladišče (angl. local repository). Vsak podatkovni element si lahko predstavljamo kot množico urejenih parov $< ime\_atributa, vrednost\_atributa >$. Kljub temu, da je lahko število podatkovnih elementov v sistemu zelo veliko, je število različnih vrst atributov omejeno.

V porazdeljeni relacijski podatkovni bazi bi bil podatkovni element posamezen zapis v bazi, urejeni pari ime-vrednost pa imena in vrednosti atributov v tabeli, kateri pripada zapis. V sistemih Grid pa so podatkovni elementi kar posamezni viri. Ti so prav tako predstavljeni z množico atributov in njihovih vrednosti. Predpostavimo tudi, da so nekateri njihovi atributi dinamični (njihove vrednosti se lahko spreminjajo). Cilj vseh sistemov za odkrivanje virov je poiskati podatkovne elemente, katerih atributi zadostijo množici iskalnih pogojev, ki so predstavljeni v obliki območnih poizvedb.

Dan je sistem Grid z množico vozlišč $\mathcal{P}$: $\mathcal{P} = \{P_1, P_2, ..., P_N\}$. Lokalno skladišče vozlišča P_i označimo z $Data(P_i)$. Vsak podatkovni element $D \in Data(P_i)$ je določen z množico atributov. Predpostavimo, da je v $\mathcal{P}$ prisotno omejeno število M različnih vrst atributov in pripadajočih tipov: $\mathcal{A} = \{A_1 : T_1, ..., A_M : T_M\}$. Vsak T_i je lahko poljuben podatkovni tip, edini pogoj je, da je njegova domena linearno urejena. Naj bo $AttList(D) \subseteq \mathcal{A}$ množica atributov, ki so povezani s podatkovnim elementom $D \in Data(P_i)$, $D.A$ pa vrednost atributa $A \in AttList(D)$.

Sistem $\mathcal{P}$ mora definirati storitev za iskanje podatkovnih elementov na podlagi območne poizvedbe Q, ki jo definira uporabnik. Privzamemo, da poizvedbe ustrezajo spodnji gramatiki (veljajo običajne prednosti operatorjev):

$$\begin{aligned} Q &:= Q \ \textbf{\textit{and}} \ Q \quad | \quad Q \ \textbf{\textit{or}} \ Q \quad | \quad v_1 \leq A \leq v_2 \\ A &:= A_1 | ... | A_M \end{aligned}$$

Kot smo že predhodno omenili, bomo obravnavali območne poizvedbe nad podmnožico atributov, torej logične kombinacije območnih predikatov tipa $v_1 \leq A \leq v_2$. To seveda pomeni, da lahko določena poizvedba vsebuje več pogojev z različnimi atributi. Pogoje, ko so $A \leq v_2$, $A \geq v_1$ in $A = v_1$, obravnavamo kot posebne primere pogoja $v_1 \leq A \leq v_2$, do katerih pridemo s prirejanjem vrednosti: $v_1 = -\infty$, $v_2 = +\infty$ in $v_1 = v_2$.

Komponenta za odkrivanje virov mora podpirati operacije:

- ***insert***$(D, \{A_1 : V_1, ..., A_r : V_r\})$: vstavi nov podatkovni element D v lokalno skladišče. Podatkovni element obsega atribute $A_1, ..., A_r$ in njihove vrednosti $V_1, ..., V_r$.
- ***update***$(D, A : V_{new})$: spremeni vrednost atributa A pri podatkovnem elementu D, ki se nahaja v lokalnem skladišču. Nova vrednost atributa bo V_{new}.

- ***delete***(D): odstrani podatkovni element D iz lokalnega skladišča.
- ***lookup***(Q): v celotnem sistemu Grid poišči podatkovne elemente, ki ustrezajo poizvedbi Q. Ta operacija v bistvu vrne naslove vozlišč, na katerih se nahajajo ustrezni podatkovni elementi. Vsa ta vozlišča moramo kontaktirati neposredno, če želimo pridobiti podatkovne elemente.

Za sistem Grid velja, da vozlišča prihajajo in odhajajo ob poljubnih trenutkih. Podobno kot pri sistemih P2P, se želimo tudi tu čim manj zanašati na centralizirano informacijo.

Najpreprostejši način iskanja virov bi bilo poplavljanje, kot se uporablja pri nestrukturiranih sistemih P2P: poizvedbe bi poslali vsem vozliščem, ki se nahajajo dovolj blizu vozlišča, ki je sprožilo iskalno zahtevo. Ta pristop pa ni priporočljiv, saj ima precej slabosti:

- ustvarja veliko omrežnega prometa in s tem obremenjuje vozlišča, tudi tista, ki ne vsebujejo virov, ki bi ustrezali dani poizvedbi,
- zaradi omejevanja dosega poplavljanja, s čimer mu preprečimo, da bi preobremenilo celotno omrežje, nimamo zagotovila, da bomo uspeli najti vse ustrezne vire.

Zaradi omenjenih slabosti je smiselno zamenjati poplavljanje s t.i. prekrivnimi omrežji (angl. overlay networks). Zanje je značilno, da vozlišča vsebujejo posebne usmerjevalne tabele, s katerimi si pomagajo pri usmerjanju sporočil skozi omrežje.

2 Obstoječi sistemi za odkrivanje virov

Sistemi za odkrivanje virov skrbijo za pridobivanje, porazdeljevanje, indeksiranje, arhiviranje in obdelavo podatkov o konfiguraciji in stanju storitev in virov. Med razlogi za zbiranje podatkov so možnost kasnejšega odkrivanja storitev in virov ter spremljanje stanja sistema.

V tem poglavju si ogledamo nekaj obstoječih sistemov za odkrivanje virov. Le-te v grobem razdelimo v 3 skupine: sistemi za odkrivanje virov na osnovi hierarhične strukture, sistemi za odkrivanje virov na osnovi nestrukturiranih P2P tehnologij ter sistemi za odkrivanje virov na osnovi strukturiranih P2P tehnologij. Za vsak sistem si ogledamo njegove prednosti in slabosti, poglavje pa zaključimo s splošnim opisom problemov pri obstoječih sistemih za odkrivanje virov.

2.1 Globus Toolkit 4 Monitoring and Discovery Service

Globus Toolkit (GT) (`http://www.globus.org`) je najpopularnejše ogrodje za izgradnjo sistemov Grid. Pojavilo se je konec 90. let za podporo razvoju storitveno usmerjenih aplikacij in infrastruktur v omrežnem računalništvu. Ključne komponente GT služijo zaščiti, dostopu do virov in upravljanju z njimi, prenosu in obdelavi podatkov, odkrivanju virov, ...

Globus Toolkit različica 4 (GT4) [8], ki so jo izdali aprila 2005, predstavlja velik napredek glede na različico 3 (GT3) na področju izvedbe spletnih storitev, tako v številu komponent, njihovi funkcionalnosti in uporabnosti, kot tudi v kakovosti dokumentacije in skladnosti z obstoječimi standardi. Obe različici temeljita na člankih I. Fosterja [2] in [9], ki ju mnogi obravnavajo kot temeljni deli na področju sistemov Grid. Pravzaprav velja celotno ogrodje Globus Toolkit za nekakšen de-facto standard za izgradnjo sistemov Grid, po katerem se zgledujejo tudi ostala ogrodja.

V GT4 za nadzorovanje in odkrivanje storitev in virov skrbi komponenta Monitoring and Discovery Service (MDS). Odkrivanje je proces iskanja ustreznih virov za opravljanje danih nalog: npr. iskanje gostitelja za izvajanje določenega opravila. Ta proces vključuje tako iskanje virov, ki zadostujejo navedenim zahtevam (npr. imajo

ustrezno CPU arhitekturo, operacijski sistem), kot tudi izbiro najprimernejšega vira (npr. vira, ki ima najkrajšo čakalno vrsto).

Tako nadzorovanje kot odkrivanje potrebujeta sistem za zbiranje informacij o različnih virih. V ta namen vsebuje MDS t.i. storitve za kopičenje (angl. aggregation services), ki zbirajo informacije o trenutnem stanju informacijskih virov. Za dostop do teh informacij ima uporabnik na voljo različne ukaze, specializirane spletne storitve in brskalniške vmesnike. Za kopičenje so na voljo tri vrste storitev:

- **MDS-Index.** Podatke, zbrane s pomočjo informacijskih virov, ponudi v obliki XML dokumentov. Uporabniki do njih dostopajo s pomočjo standardnih vmesnikov spletnih storitev (metoda GT4 *get-property*), ki ob klicu vrne podatke o stanju spletne storitve, izvedba standarda WS-Notification, ki omogoča spremljanje podatkov o spletni storitvi po principu *Subscription/Notification*), ukaza wsrf-get-property ali internetnega brskalnika s pomočjo vmesnika WebMDS.
- **MDS-Trigger.** Definira vmesnik, s pomočjo katerega lahko odjemalec registrira poizvedbo Xpath in pripadajoč program, ki se bo izvršil, ko bo določena poizvedba uspela.
- **MDS-Archiver.** Shranjuje vse podatke, ki jih dobi od informacijskih virov.

MDS uporablja XML in vmesnike spletnih storitev za poenostavitev registracije informacijskih virov in odkrivanje ter dostop do želenih informacij. Vsi zbrani podatki so shranjeni v XML obliki, zato lahko po njih iščemo s poizvedbami XPath (`http://www.w3.org/TR/xpath`). Ključnega pomena za MDS je t.i. ogrodje zbiralnik-informacijski vir (angl. aggregator-information source framework). Njegove značilnosti so:

- Informacijski viri, po katerih iščemo, morajo biti obvezno prijavljeni pri storitvah za kopičenje.
- Prijave imajo življenjsko dobo: če jih občasno ne obnavljamo, jim le-ta poteče. Tako je zagotovljeno odstranjevanje zapadlih registracij.
- Zbiralnik občasno obnavlja podatke o stanju vseh prijavljenih informacijskih virov.
- Zbiralnik s pomočjo vmesnikov spletnih storitev nudi uporabnikom na razpolago vse podatke, pridobljene od informacijskih virov.

Podatke o virih lahko pridobivamo na več načinov:

- s pomočjo uporabniškega programa, ki ga izvajamo periodično, da dobimo ažurne podatke. Program mora generirati podatke v obliki XML,

- preko vmesnika WSRF *get resource property*, s pomočjo katerega zbiralnik bere podatke o viru,
- s pomočjo standarda WS-Notification, ki omogoča informacijskemu viru, da sam obvešča zbiralnik o vsaki spremembi stanja.

Globus Toolkit omogoča tudi povezovanje MDS komponent na različnih računalnikih v globalno imeniško strukturo. Ta je ključnega pomena za gradnjo sistemov Grid, saj bi bilo brez nje odkrivanje nemogoče. Pri MDS2 in MDS3 (storitvah MDS v Globus Toolkit, različicah 2 in 3) je bila imeniška struktura realizirana v obliki centralnega imenika, do katerega so vozlišča dostopala s pomočjo protokola LDAP. Posledica tega je bila zelo velika obremenitev vozlišča, ki je gostilo centralni imenik, in njeno naraščanje z večanjem sistema Grid. Šibki točki te arhitekture sta bili še slaba razširljivost in prisotnost osamljene točke odpovedi, saj je v primeru okvare vozlišča s centralnim imenikom prenehala delovati celotna imeniška struktura.

2.1.1 Hierarhična struktura sistema MDS4

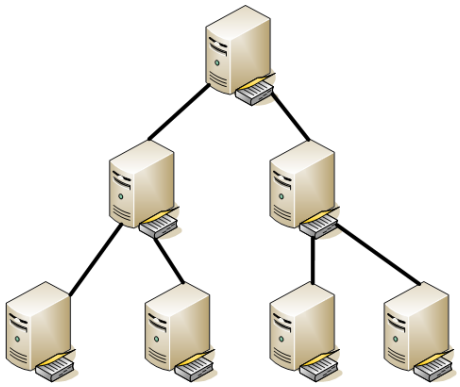
Zaradi zgoraj navedenih slabosti strežniške arhitekture so se pri načrtovanju MDS4 odločili za hierarhičen način povezovanja vozlišč v imeniški infrastrukturi, podoben tistemu, ki ga določa standard za imenike spletnih storitev Universal Description, Discovery and Integration (UDDI, [1]).

Pri MDS4 se vozlišča povezujejo med seboj na hierarhičen način, kot je prikazano na sliki 2.1. Hierarhično drevo vozlišč določa uporabnik statično s pomočjo ustreznih nastavitvenih datotek. Vozlišča, ki se nahajajo višje v drevesu, zbirajo vsebino imenikov MDS vseh njihovih potomcev (tj. vozlišč, ki se nahajajo v poddrevesu, katerega koren je dano vozlišče). Ko objavimo podatek v določenem vozlišču MDS, se ta razmnoži navzgor do korena drevesa (lahko tudi navzdol do listov, če smo tako določili v nastavitvenih datotekah MDS). To pomeni, da so v korenskem vozlišču drevesa zbrani vsi podatki, ki so objavljeni v sistemu Grid. To vozlišče je podobno centralnemu imeniku, kljub temu pa je njegova vloga v sistemu precej drugačna, kot bomo spoznali v nadaljevanju.

Hierarhična struktura ima precej prednosti v primerjavi s strežniško:

- manjša obremenitev korenskega vozlišča: kljub temu, da se vsi objavljeni podatki še vedno hranijo v korenskem vozlišču, ki zaradi tega spominja na centralni strežnik, dostopamo do njega samo v skrajni sili, ko nismo nikjer našli želenega podatka. Iskalnim zahtevam navadno zadostimo precej nižje v drevesu.
- odsotnost osamljene točke odpovedi: v primeru izpada korenskega vozlišča se hierarhično drevo ne podre, temveč se samo razdeli na ločeni poddrevesi. Zno-

[1] `http://www.oasis-open.org/committees/uddi-spec/doc/tcspecs.htm`



Slika 2.1: Hierarhična struktura vozlišč MDS4.

traj teh poddreves je še vedno mogoče iskanje. Hierarhična globalna imeniška infrastruktura torej ne odpove v celoti, kar se je zgodilo pri strežniški.

2.2 Sistemi za odkrivanje virov na osnovi nestrukturiranih P2P tehnologij

2.2.1 Iaminitchi in sodelavci

V [10] Iaminitchi s sodelavci opiše popolnoma decentraliziran sistem za odkrivanje virov v sistemih Grid na osnovi tehnologij P2P. Zanj je značilno, da vsak udeleženec v navidezni organizaciji objavi svoje podatke na enem ali več vozliščih. Posamezno vozlišče skrbi za shranjevanje in streženje podatkov, ki jih hrani. Uporabniki pošiljajo poizvedbe enemu ali več znanim vozliščem (navadno je to kar lokalno vozlišče). Če dano vozlišče hrani kak vir, ki ustreza poizvedbi, pošlje njegov opis uporabniku. V nasprotnem primeru posreduje poizvedbo svojim sosedom. Sporočilo potuje po omrežju, dokler ne najde ustreznega vira oziroma dokler mu ne poteče življenjski čas (angl. time-to-live, TTL). Če vozlišče hrani podatke o ustreznem viru, jih pošlje vozlišču, ki mu je posredovalo poizvedbo, to pa jo pošlje naprej. Postopek se ponavlja toliko časa, dokler odgovor ne prispe do vozlišča, ki je sprožilo poizvedbo.

Sistem, kot je opisan v [10], je sestavljen iz 4 sklopov:

- konstrukcija prekrivnega omrežja (angl. overlay construction): izbira sosede, s katerimi se bo dano vozlišče povezalo,

- protokol vzdrževanje članstva v omrežju (angl. membership protocol): določa postopek prijavljanja novih vozlišč v omrežje in medsebojnega obveščanja vozlišč,

- predprocesiranje: lokalno procesiranje prihajajočih poizvedb pred njihovim izvajanjem. Namen tega sklopa je izboljšati hitrost in uspešnost iskanja.
- procesiranje iskalnih zahtev: skrbi za prenos iskalnih zahtev po omrežju. Ta komponenta se odloča, katerim sosedom bo lokalno vozlišče posredovalo iskalno zahtevo. Vozlišča lahko, poleg naslovov sosedov, hranijo tudi statistične informacije o predhodno izpolnjenih iskalnih zahtevah.

Pri usmerjanju iskalnih zahtev v P2P omrežju lahko izbiramo med 4 strategijami:

- naključno usmerjanje (angl. random walk): vozlišče, ki mu predamo iskalno zahtevo, se izbere naključno,
- usmerjanje z učenjem (angl. learning-based): pri tej strategiji predamo iskalno zahtevo vozlišču, ki je v preteklosti že uspešno odgovorilo na podobno poizvedbo. Če tako vozlišče ne obstaja, ga izberemo naključno,
- usmerjanje k najboljšemu sosedu (angl. best-neighbour): tu beležimo število odgovorov na poizvedbe, ki smo jih dobili od vsakega izmed sosedov. Iskalno zahtevo posredujemo vozlišču z največjim številom odgovorjenih iskalnih zahtev,
- kombinacija usmerjanja z učenjem in usmerjanja k najboljšemu sosedu: ta strategija je identična usmerjanju z učenjem, le da tu pošljemo zahtevo vozlišču z največjim številom odgovorjenih zahtev kadar nismo uspeli najti vozlišča, ki je v preteklosti že odgovorilo na podobno zahtevo.

Rezultati poskusov, ki so jih avtorji opravili na emulatorju sistemov Grid, so pokazali, da je usmerjanje z učenjem uspešnejše od usmerjanja k najboljšemu sosedu in naključnega usmerjanja, pri vsaki porazdelitvi iskalnih zahtev. Poglavitna prednost usmerjanja z učenjem je, da si zapomni podobnosti v iskalnih zahtevah z uporabo pomnjenja. Na začetku je iskanje najmanj učinkovito, pozneje pa se bistveno izboljša, ko se v medpomnilnik shranjuje vedno več iskalnih zahtev.

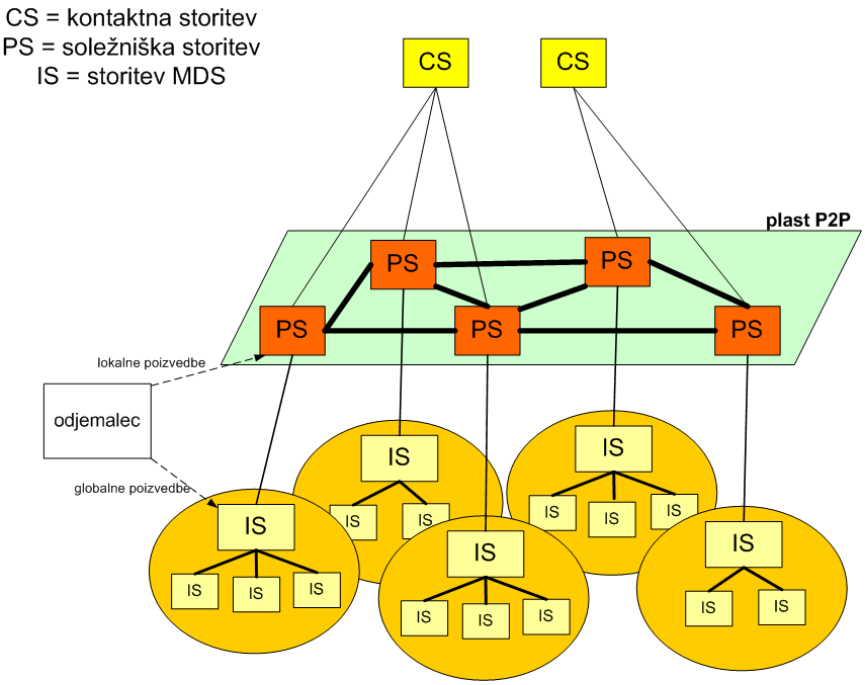
Pričakovano se je za najmanj učinkovito strategijo izkazalo naključno usmerjanje. Po drugi strani pa ima naključno usmerjanje tudi prednosti: je preprosto, poleg tega pa zanj ne potrebujemo prostora za beleženje zgodovine iskanj.

2.2.2 Talia in sodelavci

V [11] Talia s sodelavci opiše arhitekturo sistema P2P za odkrivanje virov v sistemih Grid, ki so združljivi s standardom Open Grid Service Architecture (OGSA) [12]. Arhitektura je sestavljena iz dveh plasti (glej sliko 2.2):

- spodnjo plast predstavlja drevesna struktura storitev MDS ogrodja Globus Toolkit. Te hranijo podatke o lokalnih virih, ki se nahajajo znotraj določene navidezne organizacije,

- zgornja plast je t.i. plast P2P, ki zbira podatke iz spodnje plasti in jih porazdeljuje. Ta plast zajema dva tipa spletnih storitev, združljivih s standardom OGSA: soležniške storitve (angl. Peer Services), ki služijo za usmerjanje zahtev za odkrivanje virov med posameznimi navideznimi organizacijami ter kontaktne storitve (angl. Contact Services), ki omogočajo soležniškim storitvam, da se povežejo v omrežje P2P.



Slika 2.2: Arhitektura sistema P2P, ki ga je zasnoval Talia s sodelavci.

Vsaka navidezna organizacija vsebuje svojo soležniško storitev. Soležniške storitve so povezane med seboj v omrežje P2P in si med seboj izmenjujejo poizvedbe in odzive nanje. Neposredna komunikacija je mogoča samo med sosedi. To pomeni, da lahko soležniška storitev pošlje poizvedbo samo svojim sosedom, ti pa jo potem posredujejo naprej, dokler ni izpolnjen pogoj za ustavitev. Pri obdelavi poizvedbe soležniška storitev kontaktira storitev MDS, ki se nahaja v korenu drevesne strukture za dano navidezno organizacijo. Podatki o virih, ki ustrezajo poizvedbi, se pošljejo začetnemu vozlišču po isti poti, kot je potovala poizvedba.

Pri prijavljanju nove soležniške storitve v omrežje P2P mora ta poznati URL vsaj ene soležniške storitve, ki je že prijavljena v omrežje. Pri tem si pomaga s kontaktnimi storitvami. Te hranijo URL naslove, ki jih potrebuje nova soležniška storitev za prijavo.

Za usmerjanje sporočil med soležniškimi storitvami se uporablja enak protokol kot pri sistemu Gnutella [13]. Ta protokol uporablja specializirane tehnike za učinkovito usmerjanje sporočil v omrežju P2P. Omenimo samo dve izmed njih:

- medpomnjenje sporočil (angl. message buffering): sporočila za istega naslovnika se shranijo v medpomnilnik in se pošiljajo v enem samem paketu v časovnih intervalih,
- združevanje sporočil (angl. message merging): sporočila z isto glavo (isti tip, identifikator, prejemnik) združimo v eno samo sporočilo.

Rezultati poskusov kažejo, da ustrezna uporaba medpomnjenja in združevanja sporočil izboljšuje učinkovitost iskanja v P2P omrežjih.

2.2.3 Mastroianni in sodelavci

V [14] Mastroianni s sodelavci predstavi predelavo modela super-soležnikov (angl. super-peer) za implementacijo informacijske storitve za sisteme Grid. Model super-soležnikov združuje učinkovitost centraliziranega iskanja z neodvisnostjo, uravnoteženostjo bremena ter odpornostjo na napake porazdeljenega iskanja [15]. Super-soležnik predstavlja centralizirani strežnik za množico običajnih soležnikov, ki se priključijo nanj. Hkrati pa se več super-soležnikov poveže med seboj v ločeno omrežje P2P.

Model super-soležnikov je še posebej primeren za uporabi v sistemih Grid, saj se njegova struktura ujema s strukturo sistemov Grid. Sistem Grid si lahko predstavljamo kot množico t.i. fizičnih organizacij (angl. physical organizations, PO) ki so povezane med seboj v omrežje. Vsako fizično organizacijo sestavlja množica vozlišč, ki se nahajajo znotraj skupne administrativne domene. Eno ali več vozlišč (npr. tisto z najzmogljivejšim procesorjem, največ pomnilnika) znotraj fizične organizacije prevzame vlogo super-soležnika, ostala vozlišča pa preko njih dostopajo do sistema Grid oziroma vršijo iskanje virov. Super-soležnik ime 2 vlogi: skrbi za komunikacijo z drugimi fizičnimi organizacijami in vzdržuje seznam vseh vozlišč v lokalni fizični organizaciji.

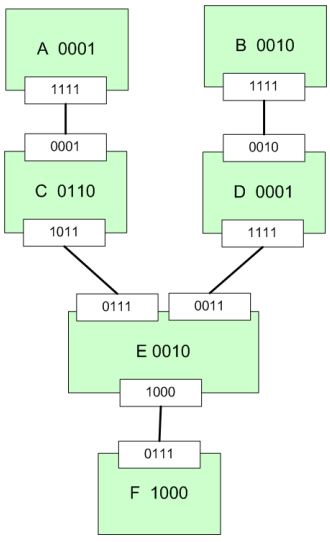
Odkrivanje virov poteka na sledeč način: poizvedbe, ki jih generirajo vozlišča v sistemu Grid se najprej posredujejo lokalnemu super-soležniku. Ta preišče bazo podatkov o objavljenih virih in preveri, če obstajajo viri v lokalni fizični organizaciji, ki ustrezajo dani poizvedbi. Če ustrezni viri obstajajo, pošlje super-soležnik seznam vozlišč, na katerih se nahajajo, vozlišču, ki je sprožilo iskalno zahtevo. Če pa v lokalni fizični organizaciji ne obstaja noben ustrezen vir, super-soležnik posreduje poizvedbo izbrani podmnožici sosednjih super-soležnikov. Ti preverijo svoje baze podatkov o virih in morebitne zadetke pošljejo začetnemu vozlišču po isti poti, kot je potovala poizvedba. Podmnožica sosedov, katerim super-soležnik posreduje poizvedbo, se izbere na podlagi statističnih podatkov o predhodnih zadetkih prejetih iz posameznih vozlišč. Avtorji v [14] predstavijo tudi več različnih strategij za

zmanjševanje mrežnega prometa, krajšanje odzivnega časa na poizvedbe in večanje verjetnosti zadetka (verjetnost, da bomo za vsako poizvedbo dobili vsaj en vir, ki ji ustreza).

2.2.4 Marzolla in sodelavci

V [16] Marzolla s sodelavci predstavi še en sistem za odkrivanje virov v sistemih Grid, ki deluje na osnovi usmerjevalnih kazalcev (angl. routing indices). V tem sistemu so vozlišča organizirana v prekrivno omrežje, ki ima drevesno strukturo. Vsako vozlišče hrani podrobne podatke o množici virov, ki so shranjeni na njem ter zgoščen opis virov, ki se nahajajo v poddrevesih, katerih koreni so njegovi sosedje. Te podatke uporabljajo algoritmi za odkrivanje pri usmerjanju poizvedb k vozliščem, za katere je verjetnost zadetka večja.

Podatki o virih se preslikujejo na sledeč način: domena vsakega atributa, ki ga zajema nek vir, se razcepi v k podintervalov (število podintervalov k se pri različnih atributih lahko razlikuje). Pri posameznem viru je kazalec za dani atribut A predstavljen kot k-bitni vektor, pri katerem so vsi biti enaki 0, razen tistega, ki ustreza podintervalu, v katerem se nahaja vrednost A za deni vir. Kazalec za atribut A, ki predstavlja vse vire, ki so shranjeni lokalno na danem vozlišču, dobimo z logično operacijo ALI nad vektorji atributa A za vse lokalne vire. Kazalec atributa A za vse vire, ki se nahajajo v določenem poddrevesu, dobimo z logično operacijo ALI nad lokalnimi bitnimi vektorji vseh vozlišč v poddrevesu. Primer omrežja z usmerjevalnimi kazalci je prikazan na sliki 2.3.



Slika 2.3: Primer omrežja z usmerjevalnimi kazalci, kot ga je predlagal Marzolla.

Ko soležnik P prejme večparametrsko območno poizvedbo, jo razcepi v več pod-

poizvedb, od katerih vsaka zajema po en atribut. Vsaka podpoizvedba Q se potem preslika v binarni vektor dolžine k, v katerem so z 1 označeni vsi elementi, katerih podintervali so zajeti v Q. Podpoizvedbe se najprej izvedejo nad lokalnimi kazalci virov, da se poišče vse lokalne vire, ki ustrezajo poizvedbi. Lokalnemu iskanju sledi posredovanje poizvedbe sosednjim vozliščem, pri čemer si pomagamo z usmerjevalnimi kazalci. Podpoizvedbe se posredujejo samo tistim sosedom, katerih kazalci se ujemajo z vsemi podpoizvedbami.

Vsakič, ko se vrednost katerega izmed atributov spremeni in se spremeni tudi lokalni binarni vektor za dano vozlišče, se sosedom razpošljejo sporočila za ažuriranje usmerjevalnih kazalcev. Rezultati simulacij kažejo, da lahko omenjena algoritma za obdelavo poizvedb in ažuriranje usmerjevalnih kazalcev tudi pri velikih omrežjih ne generirata prevelikega mrežnega prometa; oba algoritma imata dobro razširljivost.

2.3 Sistemi za odkrivanje virov na osnovi strukturiranih P2P tehnologij

2.3.1 MAAN

V [17] avtorji opišejo sistem MAAN, nadgradnjo strukturiranega P2P sistema Chord [18], kjer pri iskanju virov lahko uporabljamo tudi večparametrske območne poizvedbe. Vsako vozlišče v sistemu je del prekrivnega omrežja Chord. Vrednosti atributov virov se preslikajo v m-bitni ključ s pomočjo zgoščevalne funkcije (angl. hash function), ki ohranja sosednost med vrednostmi. Za vsak atribut vira se ustvari ločena registracija. Le-to si lahko predstavljamo kot urejeni par $< vrednost - atributa, podatki - o - viru >$. Registracije se pošljejo po prekrivnem omrežjo do vozlišča, ki je odgovorno za njihovo shranjevanje. Usmerjanje v prekrivnem omrežju se izvaja na podlagi m-bitnega ključa.

Iskanje s pomočjo večparametrskih območnih poizvedb je implementirano na 2 načina:

1. Prvi je t.i. iterativni prostop: poizvedba se razcepi na M podpoizvedb. Vsaka poizvedba se razreši ločeno, s pomočjo algoritmov sistema Chord. Rezultati vseh podpoizvedb se zberejo v začetnem vozlišču, kjer se izvede tudi presek nad njimi, s čimer dobimo vire, ki ustrezajo vsem pogojem v poizvedbi. Prednost tega načina je preprostost, slabost pa neučinkovitost. Njegova zahtevnost je namreč $O(\sum_{i=1}^{M}(logN + N * s_i))$, kjer je M število podpoizvedb, N število vozlišč v prekrivnem omrežju in s_i selektivnost podpoizvedbe i.

2. Drugi način je definiran kot usmerjanje na osnovi dominantnega atributa (angl. single attribute dominated routing). Naj bo X množica virov, ki zadoščajo pogojem v poizvedbi Q. To pomeni, da X zadošča tudi vsem podpoizvedbam Q. Velja torej $X = \cap_{1 \leq i \leq M} X_i$, kjer je X_i množica virov, ki zadoščajo poizvedbi na

osnovi atributa a_i. Kot pri prvem načinu, se tudi tu podpoizvedbe razrešujejo s pomočjo algoritmov sistema Chord. S tem dobimo množice X_k. Vsaka X_k je nadmnožica X, torej vsebuje vse vire, ki zadoščajo pogojem poizvedbe Q. Ker vsaka registracija (urejeni par $< vrednost - atributa, podatki - o - viru >$) vsebuje tudi vrednosti ostalih atributov, lahko že iz ene same množice X_k ugotovimo, kateri viri ustrezajo vsem podpoizvedbam Q. Zahtevnost te metode je $O(logN + N * S_{min})$, kjer je S_{min} minimalna selektivnost izmed atributov.

2.3.2 Andrzejak in sodelavci

V [19] Andrzejak in Xu predstavita informacijsko storitev za sisteme Grid, ki temelji na strukturiranem P2P sistemu CAN [20]. Storitev med drugim tudi omogoča uporabo območnih poizvedb pri iskanju virov. Podobno kot v ostalih sistemih za odkrivanje virov, so tudi tu viri predstavljeni kot množica atributov. Podatki o virih se objavljajo v dveh vrstah prekrivnih omrežij, odvisno od tipa atributa, iz katerega je generiran ključ za objavljanje:

- če se za generiranje ključa uporabi atribut, ki ima omejeno število vrednosti, se podatki o viru shranijo v običajnem prekrivnem omrežju (npr. prekrivnem omrežju sistema Chord),

- če se za generiranje ključa uporabi atribut, ki ima zvezne vrednosti, se podatki shranijo v prekrivnem omrežju sistema CAN.

Odkrivanje virov s pomočjo večparametrskih poizvedb poteka ne podoben način kot pri sistemu MAAN: za vsak atribut v poizvedbi pošlje sistem ločeno podpoizvedbo v ustrezno prekrivno omrežje. Ko zbere rezultate vseh podpoizvedb, iz njih določi končni rezultat s pomočjo operacije preseka.

Del strežnikov, ki se nahajajo v sistemu Grid, igra vlogo vozlišč v prekrivnem omrežju P2P. Ti strežniki hranijo urejene pare $< vrednost - atributa, id - vira >$. Vsako vozlišče P2P je zadolženo za shranjevanje določenega podintervala vrednosti za posamezen atribut. Vsak strežnik v sistemu Grid pošilja ažurne podatke o svojih virih vozliščem v prekrivnih omrežjih P2P, ki so zadolžena za ustrezne podintervale vrednosti. V [19] avtorji tudi predlagajo različne strategije za posredovanje območnih poizvedb, katerih cilj je zmanjšati odvečen mrežni promet, ki nastane zaradi ažuriranja vrednosti atributov.

2.3.3 SWORD

Sistem SWORD [21] podpira iskanje virov tako po statičnih kot tudi po dinamičnih atributih. Sistem vsebuje poleg komponente za iskanje s pomočjo večparametrskih območnih poizvedb tudi vrsto mehanizmov, ki omogočajo učinkovitejše iskanje virov:

- integrirane mehanizme za določanje razširljivosti in za učinkovitejše iskanje po atributih, ki vključujejo več vozlišč (primera takšnih atributov sta zakasnitev in pasovna širina med dvema vozliščema), ki ne bi imelo časovne in prostorske zahtevnosti $O(n^2)$,
- mehanizme, ki omogočajo uporabnikom definiranje kriterijev za izbiranje virov. Ti kriteriji se uporabljajo v primerih, ko je število odkritih virov preveliko,
- optimizator, ki vrši izbiranje virov na podlagi kriterijev, opisanih v prejšnji točki.

V sistemu SWORD obstajata dva tipa vozlišč: običajna vozlišča, ki periodično pošiljajo podatke o virih, ki jih hranijo, in strežniška vozlišča, ki zbirajo podatke o podatke o virih in odgovarjajo na uporabniške poizvedbe. Strežniška vozlišča so združena v prekrivno omrežje P2P s pomočjo sistema Bamboo [22], strukturiranega sistema P2P, ki je podoben sistemu Pastry [23]. Za vsak atribut A_k, ki ga zajema določen vir, pošljejo vozlišča podatke o viru tistemu vozlišču, ki je zadolženo za hranjenje podatkov z dano vrednostjo atributa. To vozlišče določimo na podlagi ključa k_k, ki ga generiramo iz vrednosti atributa A_k. Podatke o viru nato pošljemo v prekrivno omrežje in ko pripotujejo do ciljnega vozlišča, jih to shrani v svojo lokalno bazo kazalcev na vire.

Odkrivanje virov s pomočjo območnih poizvedb poteka na naslednji način: za sistem SWORD velja, da se poljubno območje vrednosti zveznega atributa A preslika v zvezen podinterval v domeni ključev za A (za to poskrbi preslikovalna funkcija f_A, ki ohranja lokalnost). Seznam virov, katerih vrednosti atributa A se nahajajo na intervalu $[x_{min}, x_{max}]$ dobimo tako, da pošljemo iskalno zahtevo vozliščem, ki so zadolžena za ključe, ki se nahajajo med $f_A(X_{min})$ in $f_A(X_{max})$.

Ko uporabnik sproži poizvedbo, se odpre nova TCP povezava, po kateri se pošlje poizvedba do najbližjega strežniškega vozlišča. Tam se sproži porazdeljeno območno iskanje (angl. distributed range search), temu pa sledi preverjanje atributov, ki vključujejo več vozlišč, in izbira rezultatov na podlagi uporabniško definiranih kriterijev.

2.3.4 Schmidt in sodelavci

Sistem, ki ga predlaga v [24] Schmidt s sodelavci, podpira večparametrske poizvedbe s pomočjo enega samega prekrivnega omrežja (večina ostalih strukturiranih P2P sistemov vzdržuje ločeno prekrivno omrežje za vsako vrsto atributa). Za preslikavo n-dimenzionalnih virov (vsak vir, ki zajema n atributov, si lahko predstavljamo tudi kot n-dimenzionalen vektor vrednosti) v eno samo dimenzijo se uporabljajo prostorsko zapolnjevalne krivlje (angl. space filling curves). Ključ, na podlagi katerega se določi vozlišče, ki bo hranilo podatke o danem viru, dobimo s prepletanjem bitov posameznih vrednosti atributov v binarni obliki. Na primer, podatki o viru s 3

atributi, katerih vrednosti so $(1(01_2), 2(10_2), 3(11_2))$, bi se shranili na vozlišče, katerega identifikator je najbolj podoben 011101. Na tem mestu moramo poudariti, da potrebujemo pri virih, ki zajemajo veliko atributov z velikim razponom vrednosti, prekrivna omrežja z daljšimi identifikatorji. Daljši identifikatorji imajo za posledico večje usmerjevalne tabele ter število sosedov, ki jih mora vzdrževati posamezno vozlišče.

V tem sistemu se območne poizvedbe obravnavajo podobno kot ujemalne poizvedbe. Razlika med njimi je le v tem, da so pri območnih poizvedbah nekateri biti lahko tudi nedefinirani. Na primer, območna poizvedba, s katero iščemo vire oblike (1 ,2, 0 do 3), je 01*00*. Zaradi takšne izvedbe območnih poizvedb lahko za spodnjo mejo in velikost intervala uporabimo samo vrednosti, ki so večkratnik števila 2. Ne moremo pa, na primer, sestaviti območne poizvedbe, s katero bi iskali vire oblike (1, 2, 1 do 3).

Ko pri pošiljanju območne poizvedbe po prekrivnem omrežju naletimo na nedefiniran bit, posredujemo poizvedbo v več smereh hkrati. Podroben opis usmerjanja območnih poizvedb, ki vsebujejo nedefinirane bite, presega okvirje tega dela, bralec se z njim lahko seznani v [24].

2.3.5 XenoSearch

Sistem XenoSearch, ki je predstavljen v [25], za indeksiranje in usmerjanje sporočil v omrežju P2P uporablja strukturirani P2P sistem Pastry [23]. XenoSearch omogoča večparametrsko iskanje virov, tako da vzdržuje ločeno prekrivno omrežje za vsak atribut. Vsako novo vozlišče se mora prijaviti v vsa prekrivna omrežja. Območne poizvedbe so mogoče, ker sistem vzdržuje globalno drevesno strukturo, v katero se shranjujejo podatki (vozlišča se nahajajo v listih drevesa). V notranjih vozliščih drevesa so t.i. točke zbiranja (angl. aggregation points, AP). Vsaka AP vzdržuje seznam intervalov, ki zajemajo vrednosti atributov virov, ki so shranjeni v danem poddrevesu. Vsaka AP je enolično določena s ključem, ki se nahaja v isti domeni kot ključi, ki jih generiramo iz vrednosti atributov virov. Ključ AP je definiran kot predpona vseh ključev, ki so shranjeni v vozliščih v danem poddrevesu. Na ta način lahko iz ključa AP določimo interval vrednosti atributov, ki so shranjeni v poddrevesu.

Pri iskanju z večparametrskimi poizvedbami se začetna poizvedba razdeli razdeli na več podpoizvedb, po ena za vsak atribut. Vsaka podpoizvedba se razrešuje ločeno, s pomočjo zgoraj opisane strukture za iskanje z območnimi poizvedbami. Ko so zbrani rezultati vseh podpoizvedb, naredimo presek nad njimi. S tem dobimo naslove vseh vozlišč, ki hranijo vire, ki zadoščajo vsem pogojev v večparametrski poizvedbi. Odjemalec mora poslati poslati še eno poizvedbo vsakemu izmed teh vozlišč, s čimer se prepriča, da so dani viri v ustreznem stanju. Podatki o virih, ki so objavljeni v prekrivnem omrežju, se namreč ažurirajo periodično, zato je mogoče,

da se je medtem stanje vira že spremenilo, ni se pa še ažuriral kazalec nanj.

2.3.6 Mercury

Mercury [26] je sistem Grid, ki med drugim omogoča tudi iskanje virov z večparametrskimi območnimi poizvedbami. Za vzdrževanje prekrivnega omrežja uporablja strukturiran P2P sistem Symphony [27]. Vsakemu atributu je dodeljeno ločeno prekrivno omrežje, imenovano razdelilnik (angl. hub). Nov vir moramo registrirati v ustreznem prekrivnem omrežju za vsak atribut, ki ga vir zajema. Pri tem se v vsakem prekrivnem omrežju shranijo vsi podatki o viru, zaradi česar nam pri procesu iskanja virov ni treba pošiljati poizvedb v več prekrivnih omrežij. Ker vsak razdelilnik indeksira vire na podlagi enega samega atributa, se tudi območne poizvedbe razrešujejo na podlagi enega samega atributa. Razreševanje poizvedb poteka iterativno: najprej v ustreznem prekrivnem omrežju poiščemo vire z najmanjšo vrednostjo danega atributa (navadno izberemo tisti atribut v poizvedbi, katerega interval veljavnih vrednosti je najkrajši). Za razliko od ostalih sistemov pri Mercury pošljemo celotno poizvedbo vozlišču, ki je zadolženo za shranjevanje virov z dano vrednostjo atributa, to pa nam pošlje samo tiste vire, ki ustrezajo vsem pogojem v poizvedbi. Postopek ponavljamo z vedno večjimi vrednostmi atributa, dokler ne dosežemo zgornje meje, ki je določena v poizvedbi.

Izenačevanje obremenjenosti (angl. load balancing) vozlišč se vrši s periodičnim zaznavanjem morebitnih neravnovesij v omrežju. Izkaže se, da naključno sprehajanje po $logN$ vozliščih v grafu sistema Symphony omogoča učinkovito vzorčenje omrežja. Preobremenjena vozlišča lahko z naključnim sprehajanjem odkrijejo vozlišča, ki so manj obremenjena, in nanje prenesejo del virov, ki jih hranijo.

2.4 Problemi v obstoječih sistemih za odkrivanje virov

Odkrivanje virov v sistemih Grid je ključnega pomena, saj sistemi Grid obsegajo veliko število virov, ki morajo biti dostopni v najkrajšem možnem času. Strežniška struktura, ki se je uporabljala za odkrivanje virov v prvotnih sistemih Grid, je primerna samo pri sistemih manjšega obsega. Kot smo že omenili v razdelku 2.1, se pri sistemih Grid, ki zajemajo veliko število vozlišč, ne obnese najbolje, predvsem zaradi prevelike obremenitve centralnega strežnika in prisotnosti osamljene točke odpovedi (v primeru izpada centralnega strežnika odkrivanje virov ni mogoče).

Hierarhična struktura, na kateri temelji sistem za odkrivanje virov MDS, v določeni meri odpravlja te slabosti. Po drugi strani pa se tudi MDS precej slabo obnese pri velikih sistemih Grid. Glavni razlog za to je njegova statična, vnaprej definirana struktura. Ta je v primerjavi z dinamično učinkovitejša in preprostejša za izvedbo, vendar ima precej slabosti:

- Bolj zapletena administracija: dodajanje novih vozlišč v imeniško strukturo ni avtomatizirano, temveč ga mora opraviti sistemski administrator s spremembo ustreznih nastavitvenih datotek.

- Slabša odpornost na izpade vozlišč in povezav: kljub temu, da je pri MDS4 v precejšnji meri odpravljena osamljena točka odpovedi, še vedno ni v celoti odporen na izpade vozlišč. V primeru okvare korenskega vozlišča v hierarhični strukturi se imenik razdeli na dva ločena podimenika. V vsakem podimeniku je iskanje sicer še vedno možno, ni pa več možno iskanje med podimenikoma. Tako ne bomo uspeli najti podatka, ki je objavljen samo v drugem podimeniku, če ga iščemo iz prvega in obratno. Težava se še stopnjuje, ko odpove več ključnih vozlišč v drevesni strukturi, saj se takrat uporabnost hierarhičnega imenika drastično zmanjša.

- Večja poraba prostora za shranjevanje: kot smo že izvedeli v razdelku 2.1, se podatki v hierarhični strukturi podvajajo od nižjih nivojev proti korenskemu vozlišču. To omogoča hitrejše iskanje in nekoliko povečuje odpornost na izpade vozlišč, hkrati pa precej zmanjšuje razpoložljiv prostor za shranjevanje podatkov. Temu problemu so bolj izpostavljena vozlišča, ki se nahajajo višje v drevesu, saj hranijo več reproduciranih podatkov. Bolj ko se pomikamo proti korenu hierarhičnega drevesa, zmogljivejšo strojno opremo potrebujemo, tako v smislu kapacitet za shranjevanje, kot tudi računske moči za hitrejšo obdelavo in iskanje podatkov. Posledica tega je seveda precej večja cena sistema.

Navedene slabosti hierarhične strukture učinkovito odpravljajo tehnologije "enak z enakim" (P2P). Te so še posebej primerne za uporabo v sistemih Grid zaradi podobne zasnove in ciljev [5]. Na področju sistemov P2P je bil v preteklem desetletju opazen velik napredek, raziskovalci so razvili vrsto sistemov, ki temeljijo na omrežjih P2P. Vsakega izmed njih lahko v grobem razvrstimo v enega izmed dveh razredov:

- sistemi, ki temeljijo na nestrukturiranih omrežjih P2P,

- sistemi, ki temeljijo na strukturiranih omrežjih P2P.

Tako strukturirani kot tudi nestrukturirani sistemi P2P so primerni za odkrivanje virov v sistemih Grid. Oba razreda imata svoje prednosti in slabosti.

Glavna prednost uporabe nestrukturiranih sistemov P2P pri odkrivanju virov je preprostost implementacije in nezahtevnost vzdrževanja omrežja P2P: pri teh sistemih ni potrebno periodično preverjati, ali se določeno vozlišče še nahaja v omrežju, prav tako ni treba vzdrževati usmerjevalnih tabel (tabel, ki služijo za usmerjevanje sporočil v omrežju P2P na podlagi njihovega ključa). Slabost nestrukturiranih sistemov P2P pa je neučinkovitost algoritmov za iskanje virov. Veliko sistemov uporablja za razširjanje poizvedb po omrežju kar poplavljanje, ki pa generira veliko mrežnega

prometa. Zato je njegova razširljivost zelo slaba. Obstaja sicer veliko mehanizmov, ki uspešno omejujejo mrežni promet zaradi poplavljanja, vsi pa imajo skupno slabost: pri nobenem ni zagotovljeno, da bo uporabnik dobil vse vire v omrežju, ki ustrezajo dani poizedbi. Boljša rešitev od poplavljanja so t.i. *usmerjevalni kazalci*. Ti usmerjajo poizvedbe proti vozliščem, ki bi lahko hranili ustrezne vire. Pri iskanju z usmerjevalnimi kazalci se generira bistveno manj mrežnega prometa kot pri poplavljanju. Po drugi strani je verjetnost zadetka precej višja kot pri uporabi mehanizmov za omejevanje mrežnega prometa pri poplavljanju. Vseeno imajo usmerjevalni kazalci tudi slabosti:

- tabele z usmerjevalnimi kazalci je treba nenehno osveževati. To generira nekaj dodatnega prometa,
- v strukturi kazalcev so pri določenih topologijah omrežja mogoči cikli. V tem primeru se lahko zgodi, da bo sporočilo krožilo po ciklu v nedogled in po nepotrebnem obremenjevalo vozlišča. To težavo rešujemo na podoben način kot pri poplavljanju: z uvedbo števca vozlišč, ki jih je prečkalo dano sporočilo. Ta števec omogoča, da se sporočilo zavrže po določenem številu prečkanih vozlišč. Slabost uvedbe števca je nezmožnost zagotavljanja odkritja vseh virov, ki ustrezajo dani poizvedbi.
- pri uporabi usmerjevalnih kazalcev se domena vsakega atributa razdeli na toliko intervalov, kolikor bitov zajema kazalec (ob uporabi n-bitnih usmerjevalnih kazalcev se domena atributa razdeli na n intervalov). Določanje najprimernejše dolžine kazalcev za posamezen atribut je zelo težavno: če so kazalci predolgi, je poraba pomnilnika velika, še posebej pri velikemu številu različnih vrst atributov. Če pa je dolžina kazalcev premajhna, so intervali, na katere se razdeli domena, preveliki, zaradi česar se lahko iskanje izrodi v poplavljanje (to se zgodi, ko imamo v omrežju prisotnih veliko podobnih virov, katerih atributi se nahajajo v istem podintervalu domene). Še posebej slabo se usmerjevalni kazalci obnesejo pri atributih, katerih domene so zelo obsežne, vrednosti atributov pa so enakomerno razporejene po celotni domeni.

Pri sistemu Iaminitchijeve in sodelavcev (glej razdelek 2.2.1) se kot najučinkovitejša strategija za odkrivanje virov izkaže tista, ki temelji na usmerjanju z učenjem. Ta strategija je osnovana na statističnem opazovanju preteklih poizvedb in ni povezana s trenutnim stanjem virov. Ko imamo opravka z dinamičnimi viri, se vrednosti njihovih atributov lahko spreminjajo zelo pogosto. V takih primerih je zelo verjetno, da viri, ki so ustrezali določeni poizvedbi v preteklosti, ne bodo ustrezali isti poizvedbi v sedanjosti. Zaradi tega imamo od usmerjanja z učenjem zelo omejene koristi.

Sistem, ki ga je opisal Talia s sodelavci (glej razdelek 2.2.2), temelji na supersoležnikih. Z medpomnjenjem in združevanjem sporočil ji uspe precej zmanjšati promet, ki se generira pri iskanju virov. Kljub vsemu iskanje še vedno temelji na

poplavljanju, kar pomeni, da bodisi še vedno generira preveč mrežnega prometa, bodisi ne uspe najti vseh virov, ki ustrezajo dani poizvedbi. Pri tem sistemu je problematična tudi večja obremenitev super-soležnikov, saj se vse iskanje vrši preko njih. S podobnimi težavami se sooča tudi Sistem Mastroiannija in sodelavcev (glej razdelek 2.2.3), saj tudi ta temelji na modelu super-soležnikov.

Sistem Marzolle in sodelavcev (glej razdelek 2.2.4) temelji na usmerjevalnih kazalcih. Zaradi tega se pri iskanju generira manj mrežnega prometa kot pri ostalih nestrukturiranih P2P sistemih. Poleg tega ima zaradi nestrukturiranosti določene prednosti pred strukturiranimi sistemi P2P: manjši mrežni promet, ki je potreben za vzdrževanje prekrivnega omrežja, ter učinkovitejše iskanje virov, katerih atributi se pogosto spreminjajo. Kljub temu pa ima tudi ta sistem slabosti:

- sistem ni primeren v primerih, ko imamo opravka z velikim številom atributov, saj zahteva pri vsakem vozlišču ločen usmerjevalni kazalec za vsak atribut vira. Poleg tega morajo biti vsi atributi podani vnaprej, kar pomeni, da je nabor virov, po katerih lahko iščemo, statičen in vnaprej določen.
- neenakomerna obremenitev vozlišč pri iskanju virov: vozlišča, ki se nahajajo bližje korena hierarhično urejenega omrežja, so bolj obremenjena,
- sistem podeduje tudi vse zgoraj naštete slabosti usmerjevalnih kazalcev.

Strukturirani P2P sistemi za odkrivanje virov, ki smo jih opisali v tem poglavju, imajo skupno lastnost: vsi poskušajo prilagoditi strukturirana omrežja P2P za iskanje virov v sistemih Grid. Na različne načine poskušajo implementirati podporo za večparametrske območne poizvedbe, ki v osnovnih strukturiranih sistemih P2P ni prisotna.

Njihova prednost je zmožnost hitrega usmerjanja poizvedb proti ciljnemu vozlišču. Ti sistemi zagotavljajo omejeno število vozlišč, ki jih mora prečkati sporočilo, preden doseže ciljno vozlišče (pri sistemih Chord in Pastry je to število reda $O(logN)$, pri CAN pa $O(N^{1/d})$, kjer je N število vseh vozlišč v omrežju P2P, d pa število dimenzij, uporabljenih v omrežju CAN). Zaradi tega je iskanje hitrejše in generira manj mrežnega prometa. Žal imajo tudi strukturirani sistemi določene slabosti:

1. Slabost strukturiranih sistemov je, da zahtevajo strukturirana omrežja. Zaradi ohranjanja strukture omrežja in osveževanja usmerjevalnih tabel morajo vozlišča neprestano komunicirati med seboj. Pri tem se lahko generira veliko mrežnega prometa, še posebej v primerih, ko se vozlišča v omrežju pogosto menjajo: vsakič, ko se vozlišče prijavi v omrežje oziroma ga zapusti, se spremeni struktura omrežja in treba je popraviti usmerjevalne tabele vozlišč. Še ena slabost strukturiranih sistemov je, da podpirajo samo ujemalne poizvedbe (poizvedbe, pri katerih se uporablja samo operator enakosti). Ta slabost je še posebej opazna pri sistemih Grid, saj se pri njih za odkrivanje virov največ uporabljajo večparametrske območne poizvedbe.

2. Za implementacijo večparametrskih poizvedb se pri vseh opisanih strukturiranih P2P sistemih za odkrivanje virov uporabljajo večnivojska omrežja. V njih se za iskanje po vsakem atributu vira uporablja ločeno prekrivno omrežje. Ta način implementacije večparametrskih poizvedb ni razširljiv, saj količina podatkov, ki so potrebni za vzdrževanje prekrivnih omrežij, in zahtevnost vzdrževanja naraščata s številom atributov (in pripadajočih prekrivnih omrežij). Glede na to, da pogosto menjanje vozlišč generira veliko mrežnega prometa že pri enem prekrivnem omrežju, je ta promet še toliko večji, ko moramo vzdrževati več prekrivnih omrežij. V zgornjih razdelkih sta bila predstavljena dva načina za razreševanje večparametrskih poizvedb. Pri prvem načinu se poizvedba razdeli na več podpoizvedb, ki se razrešujejo ločeno, vsaka poizvedba v ustreznem prekrivnem omrežju. Končni rezultat dobimo s presekom rezultatov podpoizvedb. Tu se deloma zgubijo prednosti strukturiranih sistemov P2P, saj število vozlišč, ki jih prečka določena poizvedba, ni več logaritmično, temveč ga je treba pomnožiti s številom prekrivnih omrežij, v katerih se bodo razreševale podpoizvedbe. Število obiskanih vozlišč se tako linearno veča s številom udeleženih atributov. Pri drugem načinu se podatki o virih replicirajo v vsako prekrivno omrežje. Pri iskanju se poizvedba ne deli na podpoizvedbe, ampak se v celoti razreši v enem samem prekrivnem omrežju. Prekrivno omrežje se izbere na podlagi atributa, ki ima v dani poizvedbi najožji interval (na ta način najbolj omejimo število vozlišč, ki jih prečka poizvedba). Ta rešitev odpravi slabosti prvega pristopa, vendar pa ima tudi sama določene slabosti: ob vsaki spremembi atributa se morajo podatki o viru ponovno objaviti v vseh prekrivnih omrežjih, v katerih je vir indeksiran. Če so spremembe atributov zelo pogoste, se generira veliko mrežnega prometa.

3. Vsi strukturirani P2P sistemi za odkrivanje virov, ki smo jih opisali v tem poglavju, implementirajo podporo za iskanje z območnimi poizvedbami s pomočjo zgoščevalnih funkcij, ki ohranjajo lokalnost. Vozlišča, ki hranijo vire s sosednjimi vrednostmi izbranega atributa, se v prekrivnem omrežju, ki je dodeljeno danemu atributu, nahajajo drug ob drugem. Tako lahko zaporedoma dostopamo do njih, ko razrešujemo območno poizvedbo. Ta postopek iskanja virov z območnimi poizvedbami se izkaže za precej neučinkovitega, ko so intervali sprejemljivih vrednosti atributov obsežni ali celo odprti. Pri takih poizvedbah moramo kontaktirati veliko število vozlišč, razreševanje poizvedb pa postaja vedno bolj podobno poplavljanju.

Zaradi učinkovitejšega iskanja (manjše količina generiranega prometa in krajše zakasnitve med pošiljanjem poizvedbe in prejemom rezultatov), smo se odločili implementirati sistem za odkrivanje virov na osnovi strukturiranih omrežij P2P. Pri tem smo si zadali 2 cilja:

- zasnovati učinkovitejšo podporo za večparametrske območne poizvedbe, ki bo

generirala manj mrežnega prometa od obstoječih algoritmov,

- zasnovati učinkovitejše algoritme za indeksiranje dinamičnih atributov (atributov, katerih vrednosti se pogosto spreminjajo). Največji problem pri obstoječih strukturiranih P2P sistemih za odkrivanje virov je zahteva po ponovnem indeksiranju virov vsakič, ko se jim spremeni katera od vrednosti atributov. To lahko precej obremeni omrežje, še posebej, ko so spremembe vrednosti atributov zelo pogoste.

3 Algoritem za odkrivanje virov s pomočjo porazdeljene zgoščene tabele

V tem poglavju je predstavljen glavni prispevek pričujočega dela. V njem opišemo algoritme našega sistema za odkrivanje virov, ki smo ga razvili v okviru magistrske naloge. Začnemo s predstavitvijo porazdeljene zgoščene tabele ter s podrobnejšim opisom algoritmov sistema Pastry. Pastry je implementacija porazdeljene razpršene tabele, ki smo jo uporabili pri zasnovi našega sistema za odkrivanje virov. Sledi opis sistema za odkrivanje virov, ki smo ga poimenovali DHT-RD. Poleg osnovne ideje sistema predstavimo tudi algoritma za objavljanje in odkrivanje virov.

3.1 Porazdeljena zgoščena tabela

Zgoščena tabela (angl. hash table) je ena izmed najpogosteje uporabljanih podatkovnih struktur v računalništvu. Omogoča zelo hitro shranjevanje in iskanje podatkov. Zgoščeno tabelo si lahko predstavljamo kot nekakšen slovar, v katerem so podatki shranjeni v obliki urejenih parov $< kljuc, vrednost >$, do posameznih vrednosti pa dostopamo s pomočjo ključev. Pri tem si pomagamo z zgoščevalno funkcijo (angl. hash function), ki preslika ključ v ustrezen indeks v tabeli podatkov.

Običajna zgoščena tabela je centralizirana, kar pomeni, da so vsi podatki shranjeni na enem samem računalniku. Zaradi te centraliziranosti so zgoščene tabele neprimerne za gradnjo porazdeljenih sistemov, saj so neodporne na izpade vozlišč, poleg tega pa so še slabo razširljive. Rešitev problema je porazdelitev podatkov zgoščene tabele po vozliščih v omrežju s čimer dobimo novo podatkovno strukturo, ki se imenuje *porazdeljena zgoščena tabela* (angl. distributed hash table, DHT). DHT podobno kot običajna zgoščena tabela skrbi za učinkovito preslikovanje "ključev" v "vrednosti". Ključi so posebni nizi, ki se uporabljajo za shranjevanje in odkrivanje podatkov v DHT, podobno kot se primarni ključi uporabljajo pri shranjevanju zapisov v podatkovnih bazah.

V DHT si vozlišča izmenjujejo sporočila, s čimer ohranjajo skladnost podatkov. Podobno kot pri običajni zgoščeni tabeli tudi pri DHT uporabljamo zgoščevalno funkcijo, le da jo tu uporabljamo za določanje vozlišča, ki bo hranilo dano vrednost (pri sistemih P2P so to navadno različne datoteke, kazalci na vire). Vozlišča se med seboj organizirajo v omrežje, pri čemer se poskuša doseči najugodnejši kompromis med dolžino usmerjevalne poti in stopnjo vozlišč. Tako je število povezav s sosedi, ki jih mora vzdrževati posamezno vozlišče, manjše, usmerjevalne poti do ciljnih vozlišč pa niso predolge. DHT učinkovito odpravi slabosti običajnih zgoščenih tabel. Njena razširljivost je veliko boljša od centralizirane različice, predvsem zaradi manjše količine generiranega mrežnega prometa ter enakomernejše obremenjenosti vozlišč. Poleg tega je DHT bolj odporna na izpade vozlišč: deluje lahko učinkovito tudi pri sočasnem izpadu več vozlišč.

Porazdeljene zgoščene tabele se uporabljajo za gradnjo bolj kompleksnih storitev, kot so porazdeljeni datotečni sistemi, sistemi za porazdeljevanje vsebin ter sistemi za pošiljanje sporočil. Vmesnik za shranjevanje in odkrivanje podatkov, ki ga definirajo, je splošen, zato je njihova integracija v aplikacije zelo preprosta: podatki se shranjujejo v DHT z ukazom "put", odkrivamo pa jih z ukazom "get".

Vsako vozlišče v DHT vzdržuje množico povezav na podmnožico ostalih vozlišč v omrežju. Vozlišča so med seboj povezana v t.i. *prekrivno omrežje* (angl. overlay network). Prekrivno omrežje je logično omrežje, ki se zgradi preko fizičnega omrežja z medsebojnim povezovanjem vozlišč. Za DHT je značilno, da se vrednosti z enakimi ključi shranjujejo na istem vozlišču. Vsako vozlišče je zadolženo za shranjevanje določene skupine ključev. Skupine ključev so med seboj disjunktne. Sporočila v omrežjih DHT ne potujejo naključno, ampak jih vozlišča usmerjajo proti ciljnemu vozlišču na podlagi ključa. V tem se DHT odločilno razlikujejo od nestrukturiranih sistemov P2P, pri katerih se vozlišča razširjajo po omrežju s poplavljanjem.

Trenutno obstaja že precej implementacij DHT: Chord [18], Pastry [23], Tapestry [28], CAN [20], Viceroy [29] in Kademlia [30]. Te implementacije so se pojavile zaradi naraščajočega zanimanja uporabnikov za aplikacije P2P. Čeprav se implementacije razlikujejo med seboj v podrobnostih, imajo veliko skupnih lastnosti. Pri vseh implementacijah DHT ločimo med "usmerjevalno shemo" in "usmerjevalno geometrijo". Usmerjevalna shema (angl. routing scheme) je množica mehanizmov, s pomočjo katerih se izbirajo vozlišča pri usmerjanju sporočila skozi omrežje. Usmerjevalna shema pogosto določa usmerjevalno geometrijo (angl. routing geometry), graf omrežja, v katerega se povezujejo vozlišča. Pojem usmerjevalne geometrije je zelo uporaben, saj omogoča preprostejši opis nekaterih razlik med obstoječimi implementacijami DHT. Različne usmerjevalne geometrije namreč nudijo različno prožnost pri izbiri sosedov in pri izbiri usmerjevalne poti: nekatere omogočajo tako izbiro sosedov, kot tudi poti, druge samo eno od obojega, pri tretjih geometrijah pa so tako sosedi kot tudi pot sporočil strogo določena. Prožnost pri izbiri sosedov omogoča DHT da izbira vozlišča, ki so bližje danemu vozlišču in tako skrajša zakasnitve pri

usmerjanju. Tudi prožnost pri izbiri usmerjevalne poti zmanjšuje zakasnitev pri usmerjanju, poleg tega pa tudi izboljšuje prilagodljivost omrežja na pogoste odhode vozlišč.

Seveda se posamezne implementacije razlikujejo glede na uporabljeno geometrijo usmerjanja. Na primer, Chord usmerja sporočila v obroču, CAN v hiperkocki, Pastry in Tapestry pa uporabljata strukturo, ki se imenuje Plaxtonova mreža (angl. Plaxton mesh) [31]. Vsaka implementacija ima svoje rešitve pri obravnavanju sočasnega prijavljanja in odjavljanja več vozlišč. V tabeli 3.1 so podane najbolj znane implementacije DHT in njihove geometrije usmerjanja.

Tabela 3.1: Geometrije usmerjanja za različne implementacije DHT.

DHT	**geometrija**	**vozlišča so organizirana v:**
Chord	obroč	Vozlišča so organizirana v obroč in urejena po ID
CAN	hiperkocka	d-dimenzionalen torus
Pastry	Plaxtonova mreža	drevo, listi drevesa pa tvorijo tudi obroč.
Tapestry	Plaxtonova mreža	drevo
D2B, Koord	de Bruijnov graf	de Bruijnov graf
Viceroy	metuljast graf	t.i. metuljast graf (angl. butterfly graph)

Tabela 3.2 povzema performančne zahtevnosti obstoječih sistemov DHT (vir: Araujo in Rodrigues [32]).

Tabela 3.2: Performančne zahtevnosti sistemov DHT.

DHT	**stopnja vozlišča**	**premer omrežja**	**optimalna pot**	**selektivnost vozlišč**
Chord	$O(logn)$	$O(logn)$	$O(logn)$	$n^{logn/2}$
CAN	$O(d)$	$O(dn^{1/d})$	$O(logn)$	1
Pastry	$O(logn)$	$O(logn)$	1	$n^{logn/2}$
Tapestry	$O(logn)$	$O(logn)$	1	$n^{logn/2}$
D2B	$O(1)$	$O(logn)$	1	$n^{logn/2}$
Viceroy	$O(1)$	$O(logn)$	1	1
Koorde	$O(1)$	$O(logn)$	1	1

Za porazdeljene zgoščene tabele so značilne:

- decentraliziranost: vozlišča se povezujejo v omrežje brez osrednjega nadzora,
- razširljivost: sistem deluje učinkovito tudi pri zelo velikem številu vozlišč (milijon in več),
- odpornost na izpade vozlišč: sistem deluje nemoteno (do določene mere) tudi pri pogostem prijavljanju, odjavljanju in odpovedih vozlišč.

3.1.1 Zgodovina

Sistemi DHT izhajajo iz običajnih sistemov P2P, kot so Napster [33], Gnutella [13] in Freenet [34]. Ti sistemi izkoriščajo vire, ki so porazdeljeni po vsem svetu, za izvajanje vrste aplikacij (najpogosteje je to deljenje datotek oziroma komunikacija v realnem času). Razlikujejo se po pristopih, ki jih uporabljajo za iskanje podatkov v omrežju P2P:

- Napster podatke o lokaciji virov hrani centralizirano, na strežniku. Vsako vozlišče ob prijavi pošlje strežniku seznam datotek, ki jih daje v souporabo. Strežnik te podatke indeksira in jih uporabi pri usmerjanju iskalnih zahtev k ustreznim vozliščem. Tak način indeksiranja in iskanja virov je neučinkovit, predvsem zaradi velike obremenjenosti strežnika in ranljivosti na napade.

- Gnutela in njene izvedenke so se zaradi slabosti centraliziranega pristopa usmerile k poplavljanju: vsaka iskalna zahteva se razpošlje na vsa vozlišča v omrežju. Na ta način odpravimo problem osamljene točke odpovedi, slabost tega pristopa pa je velika količina mrežnega prometa, ki se generira med iskanjem.

- Freenet je popolnoma porazdeljen sistem P2P (podatki o lokaciji virov so porazdeljeni med vsa vozlišča v omrežju). Pri iskanju virov uporablja hevristično usmerjanje na osnovi iskalnega ključa. Sistem dodeli vsaki datoteki ustrezen ključ, na podlagi katerega se podatki on njeni lokaciji shranijo v omrežju. Pri tem velja, da se podatki z enakimi ključi zbirajo na istem vozlišču. Iskalni ključ se uporablja tudi pri usmerjanju poizvedb po omrežju do vozlišča, ki hrani ustrezne datoteke. Freenet za iskanje uporablja pristop, ki je zelo podoben porazdeljenim zgoščenim tabelam. Od njih se razlikuje le po tem, da ne zagotavlja odkritja, čeprav so ustrezne datoteke objavljene v omrežju.

Porazdeljene zgoščene tabele uporabljajo bolj strukturirano usmerjanje na osnovi iskalnega ključa od tistega pri sistemu Freenet. Na ta način dosežejo decentraliziranost sistemov Gnutella in Freenet ter učinkovitost in zagotavljanje odkritja virov, ki je prisotno pri sistemu Napster. Njihova slabost je, kot smo že opisali v predhodnih poglavjih, da podpirajo samo ujemalne poizvedbe, ne pa tudi območnih.

Prvi sistemi na osnovi DHT so se pojavili leta 2001: CAN [20], Chord [18], Pastry [23], Tapestry [28]. Od takrat se je število raziskav na tem področju zelo povečalo. Hkrati pa se je povečala popularnost DHT tudi izven akademskih krogov: to tehnologijo uporablja, med drugimi, tudi priljubljeni sistem za deljenje datotek BitTorrent [35].

3.1.2 Struktura

Strukturo DHT lahko razčlenimo ne 3 dele:

- prostor ključev (angl. keyspace): to je osnova DHT. Navadno se za prostor ključev uporablja množica 160-bitnih nizov,
- delitvena shema prostora ključev (angl. keyspace partitioning scheme): določa, katero izmed vozlišč bo zadolženo za shranjevanje posameznih podintervalov prostora ključev,
- prekrivno omrežje (angl. overlay network): povezuje med seboj vozlišča. Ta komponenta omogoča za vsak ključ iskanje vozlišča, ki je zadolženo zanj.

Za boljšo predstavo o vlogi vsake izmed zgoraj navedenih komponent, si oglejmo primer uporabe DHT za deljenje datotek. Preden datoteko objavimo v DHT, moramo generirati njen ključ k (naj prostor ključev predstavlja množica 160-bitnih nizov). Tega dobimo, če uporabimo zgoščevalni algoritem SHA1 [36] nad imenom datoteke. Ko imamo k, ustvarimo sporočilo s podatki za registracijo datoteke in ga pošljemo v prekrivno omrežje. Sporočilo potuje med vozlišči v prekrivnem omrežju, dokler ne doseže vozlišča, ki je zadolženo za shranjevanje ključa k (to je določeno s pomočjo delitvene sheme prostora ključev). Na tem vozlišču se shranijo podatki o lokaciji datoteke.

Odkrivanje datotek poteka na podoben način: najprej generiramo ključ k iz imena datoteke, ki jo želimo poiskati. Nato ustvarimo sporočilo z iskalno zahtevo in ga pošljemo v prekrivno omrežje. Tudi pri iskanju prekrivno omrežje poskrbi za usmerjanje sporočila k vozlišču, ki je zadolženo za shranjevanje podatkov s ključem k. Ko iskalna zahteva doseže ciljno vozlišče, le-to pošlje nazaj podatke o lokaciji datotek, ki ustrezajo iskalni poizvedbi.

Delitvena shema prostora ključev

Večina DHT uporablja pri preslikavi $kljuc \rightarrow vozlisce$ algoritem, ki se imenuje konsistentno zgoščevanje (angl. consistent hashing). Ta algoritem definira funkcijo $\delta(k_1, k_2)$, ki vrne razdaljo med ključema k_1 in k_2 (v tem kontekstu razdalja pomeni medsebojno različnost, ne geografsko oddaljenost ali mrežno zakasnitev). Poleg tega se vsakemu vozlišču dodeli lasten ključ (bodisi na podlagi njegovega naslova IP, imena, ...), ki ga imenujemo identifikator (ID). Vozlišče z identifikatorjem i, je zadolženo za shranjevanje vseh podatkov, katerih ključem je i najbližji glede na razdaljo δ.

Na primer, DHT sistem Chord obravnava prostor ključev kot krožnico, posamezne ključe pa kot točke na tej krožnici. Funkcija $\delta(k_1, k_2)$ je definirana kot dolžina loka (v smeri urinega kazalca) med točkama k_1 in k_2. Krožnica je razdeljena na zvezne odseke, katerih krajišča so identifikatorji vozlišč. Recimo, da sta i_1 in i_2 sosednja identifikatorja. Vozlišče i_2 je potemtakem zadolženo za shranjevanje vseh podatkov, katerih ključi se nahajajo na krožnem odseku med i_1 in i_2 (z drugimi besedami: $kljuc \in (i_1, i_2]$).

Prednost konsistentnega zgoščevanja je, da prihod ali odhod vozlišča vpliva samo na množico ključev, za katere je zadolženo vozlišče s sosednjim identifikatorjem. Po tem se koncept porazdeljenih zgoščenih tabel razlikuje od običajnih zgoščenih tabel, pri katerih lahko dodajanje ali odvzemanje novega elementa sproži ponovno preslikavo celotnega prostora ključev. Ker vsaka sprememba nad lastništvom podatkov navadno sproži prenos precejšnjega števila objektov, ki precej obremeni omrežje, je zmanjšanje pogostosti takih sprememb ključnega pomena za učinkovito delovanje DHT pri pogostih spremembah v topologiji omrežja.

Prekrivno omrežje

Vsako vozlišče v DHT vzdržuje usmerjevalno tabelo, množico povezav s sosednjimi vozlišči, ki se uporablja za usmerjanje sporočil v omrežju P2P. Kot sosednja vozlišča se izbirajo tista, ki imajo najkrajšo zakasnitev do danega vozlišča. Povezave med vsemi vozlišči tvorijo prekrivno omrežje. Za vsa prekrivna omrežja velja naslednja trditev:

Trditev 1. Za vsak ključ k velja, da je poljubno vozlišče bodisi lastnik k (je zadolžen za hranjenje vrednosti, katerih ključ je k), bodisi bodisi njegova usmerjevalna tabela vsebuje povezavo do vozlišča, ki je bližje k, glede na funkcijo razdalje (δ).

Zaradi zgornje trditve je usmerjanje sporočil v omrežjih DHT zelo preprosto in učinkovito. Uporabimo požrešni algoritem, imenovan *usmerjanje na podlagi ključa* (angl. key-based routing): na vsakem koraku, posreduj sporočilo tistemu sosedu v usmerjevalni tabeli, katerega identifikator je najbližji k. Ko usmerjevalna tabela ne vsebuje nobenega takega vozlišča, je trenutno vozlišče lastnik ključa k.

Poleg konsistentnega usmerjevanja (sporočilo s ključem k vedno prispe do vozlišča, ki je zadolženo zanj, ne glede na to, iz katerega vozlišča je bilo poslano) imajo omrežja DHT še dve pomembni lastnosti:

- število skokov (angl. hops) pri usmerjanju sporočil v prekrivnem omrežju je omejeno. Posledica tega je krajša zakasnitev pri iskanju podatkov,

- število sosedov pri vsakem vozlišču je omejeno (stopnja vsakega vozlišča je omejena). To se odraža v manjšem prometu, ki je potreben za vzdrževanje omrežja.

Seveda sta si zgornji lastnosti nasprotujoči: krajše usmerjevalne poti zahtevajo večjo stopnjo vozlišč in obratno. Pri večini omrežij DHT sta maksimalna stopnja vozlišč in maksimalno število skokov enaka $O(logn)$, kjer je n število vozlišč v prekrivnem omrežju.

3.1.3 Splošna usmerjevalna shema pri porazdeljenih zgoščenih tabelah

Kot smo že omenili v prejšnjem razdelku, so podatki v porazdeljenih zgoščenih tabelah shranjeni v obliki urejenih parov $< kljuc, vrednost >$, npr. $< Andrej, 32 >$, kjer je "Andrej" ključ, 32 pa vrednost, ki jo želimo shraniti (v našem primeru Andrejeva starost). Porazdeljene tabele morajo imeti definirano preslikovalno funkcijo, ki preslika ključ v psevdo-naključno vrednost (npr. število ali točka v navideznem prostoru ključa). Ena izmed ključnih zahtev za preslikovalno funkcijo je enakomerna porazdelitev ključev po zalogi vrednosti. Predpostavimo, da vozlišča v DHT tvorijo graf $\mathcal{H}$, ki si ga lahko predstavljamo kot funkcijo množice obstoječih vozlišč. Vsakemu vozlišču se dodeli naključen identifikator, s čimer se vozlišča enakomerno razporedijo po domenskem prostoru identifikatorjev. Ta se mora ujemati z zalogo vrednosti preslikovalne funkcije (zaradi tega nekatere DHT, npr. Bamboo [22], generirajo identifikatorje vozlišč kar s pomočjo preslikovalne funkcije, npr. iz kombinacije naslova IP in oznake vrat (angl. port)). Pri enakomerno razporeditvi identifikatorjev vozlišč bo vsako vozlišče hranilo približno enak delež ključev.

Usmerjevalna shema mora tvoriti tak graf $\mathcal{H}$, v katerem obstaja usmerjevalna pot med poljubnima vozliščema. Pri shranjevanju oziroma iskanju vrednosti, ki ustreza določenemu ključu, mora DHT najprej poiskati vozlišče, ki je zadolženo za shranjevanje danega ključa. Recimo, da se pri zgornjem primeru ključ "Andrej" preslika v vrednost 81. V splošnem vozlišče z identifikatorjem 81 ne obstaja (ker so identifikatorji vozlišč razporejeni po celotnem domenskem prostoru) zato DHT izbere neko drugo vozlišče, ki bo postalo zadolženo za ključ "Andrej". Predpostavimo, da je bilo izbrano vozlišče z identifikatorjem 90. Vsako vozlišče (npr. vozlišče 13), ki želi izvedeti Andrejevo starost, pošlje poizvedbo s ključem "Andrej", ki se preslika v 81. V splošnem imajo vozlišča samo delen pregled nad omrežjem, zato vozlišče 13 ne ve, ali vozlišče 81 sploh obstaja. Kljub temu bo DHT poskušala usmeriti sporočilo čim bliže identifikatorju 81 in sporočilo bo na koncu končalo pri vozlišču 90. Ena od slabosti večine implementacij DHT je, da se prekrivno omrežje slabo prilega fizičnemu omrežju. Zaradi tega lahko skok v DHT predstavlja zelo veliko razdaljo v spletu, pri več zaporednih skokih pa se lahko zgodi, da sporočilo večkrat potuje tja in nazaj. Posledica tega je, med drugim, večja zakasnitev pri objavljanju in iskanju podatkov.

Prihodi in odhodi vozlišč v prekrivno omrežje se obravnavajo na podoben način pri vseh implementacijah DHT. Za ilustracijo si bomo ogledali obravnavanje prihodov in odhodov pri sistemu Pastry:

- Vozlišče z identifikatorjem N mora ob vstopu v prekrivno omrežje najprej kontaktirati vozlišče P_0, ki je že prisotno v omrežju. To pošlje posebno sporočilo "join" vozlišču P_f, ki je zadolženo za shranjevanje ključa N. To sporočilo se usmerja po prekrivnem omrežju, dokler ne doseže P_f. Na ta način P_f izve, da

bo v omrežje vstopilo novo vozlišče, kateremu mora poslati del svojih ključev. Od tu naprej je postopek vstopanja vozlišča v omrežje odvisen od implementacije DHT.

- Ob odhodu vozlišča je potrebna prerazporeditev njegovih ključev na ostala vozlišča, ki so še prisotna v omrežju.

Pri DHT velja, da mora vsako sporočilo prepotovati največ $O(logn)$ vozlišč preden doseže ciljno vozlišče. Ta lastnost omejuje količino generiranega prometa in čas iskanja.

3.2 Pastry

Pastry temelji na delu Plaxtona in sodelavcev [31]. Sporočila se v prekrivnem omrežju usmerjajo na sledeč način: recimo, da vozlišče $S = 14543_{10}$ pošlje sporočilo vozlišču $D = 84944_{10}$. Da bi pošiljanje uspelo, mora S poznati vsaj enega soseda, katerega identifikator se začne z 8 (to je prva števka v identifikatorju D). Naj bo tako vozlišče $R_1 = 83135$. R_1 mora poznati soseda, čigar identifikator se začne s predpono "84", npr. vozlišče $R_2 = 84899$. Enaka pravila veljajo za $R_3 = 84996$, $R_4 = 84945$, zatem pa sporočilo že doseže ciljno vozlišče D. Sistem Pastry kot bazo za identifikatorje uporablja 2^b, kjer je b naravno število število (parameter sistema). b je navadno nastavljen na 4, kar pomeni, da je baza identifikatorjev šestnajstiška. V primeru, da so usmerjevalne tabele vozlišč ažurne in če v bližnji prihodnosti ni prišlo do izpadov vozlišč, bi morala biti usmerjevalna pot do vsakega vozlišča dolga največ $O(logn)$ skokov. Tudi število elementov v usmerjevalni tabeli vsakega vozlišča bi moralo biti enako $O(logn)$. Tu je n število vozlišč v prekrivnem omrežju.

Vsako vozlišče ima podatke za usmerjanje shranjene v treh komponentah:

- usmerjevalna tabela: vključuje informacije, ki se uporabljajo pri usmerjanju na zgoraj opisan način,
- množica listov usmerjevalnega drevesa (angl. leaf set): vključuje vozlišča, katerih identifikatorji so zelo podobni identifikatorju danega vozlišča. Vozlišča uporabljajo to množico pri ugotavljanju, kateri ključi pripadajo njim, kateri pa njihovim sosedom. Poleg tega se množice listov uporabljajo za usmerjanje sporočil med vozlišči s podobnimi identifikatorji,
- množica sosedov (angl. neighbourhood set): v tej množici so shranjeni podatki o vozliščih, ki se v fizičnem omrežju nahajajo blizu danega vozlišča (zakasnitev pri pošiljanju sporočil k tem vozliščem je krajša kot pri ostalih vozliščih). Namen množice sosedov je izboljšati lokalnost usmerjanja. Posredovanje sporočil vozliščem, ki so blizu danega vozlišča, namreč zelo izboljša učinkovitost usmerjanja.

Tabela 3.3 prikazuje komponente za shranjevanje usmerjevalnih podatkov pri vzorčnem vozlišču sistema Pastry, pri čemer je $b = 2$, množica listov vsebuje 8 elementov, identifikatorji vozlišč pa zajemajo 5 znakov. Identifikatorji vozlišč so sestavljeni iz treh delov: predpone, ki se ujema z identifikatorjem lokalnega vozlišča; prve števke, ki se razlikuje z istoležno števko pri lokalnem vozlišču; pripone, ki je navadno različna od pripone lokalnega vozlišča. Prva vrstica hrani naslove vozlišč, ki nimajo skupne predpone z lokalnim vozliščem. Druga vrstica hrani naslove vozlišč, katerih identifikator se ujema z identifikatorjem lokalnega vozlišča v prvem znaku, in tako naprej. V vsaki vrstici je celica, katere števka se ujema z istoležno števko identifikatorja lokalnega vozlišča, obarvana v sivo. Usmerjevalna tabela vsakega vozlišča ima v povprečju $log_{2^b} n$ vrstic [23].

Tabela 3.3: Primer komponent za shranjevanje podatkov za usmerjanje (identifikator vzorčnega vozlišča je 23002).

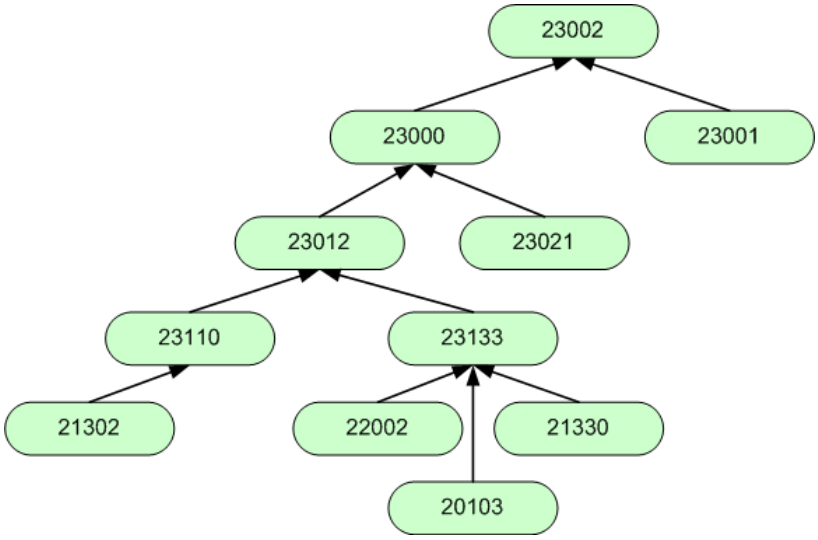
identifikatorVozlišča 23002			
Množica listov			
Manjši		Večji	
23001	22333	23022	23033
22321	22312	23100	23101
Usmerjevalna tabela			
-0-1023	-1-2131	2	-3-0231
2-0-021		2-2-032	3
0		23-2-33	
0		230-2-2	
		2	

Rowston in Druschel [23] trdita, da Pastry zelo dobro izkorišča lokalnost, saj vozlišča izbirajo za svoje sosede tista vozlišča, ki so jim bliže v fizičnem omrežju. To nam zagotavlja algoritem za izgradnjo prekrivnega omrežja.

3.2.1 Usmerjevalna geometrija

Sistem Pastry uporablja hibridno usmerjevalno geometrijo [37]. Na podlagi usmerjevalne tabele so vozlišča organizirana v drevo, na podlagi množice listov pa v obroč z redundantnimi povezavami. Glede na to, da je obroč preprost za razumevanje, se bomo osredotočili na usmerjevalno drevo. Vsako vozlišče predstavlja koren svojega usmerjevalnega drevesa. Vsako vozlišče, ki si z danim vozliščem deli predpono, se lahko nahaja v njegovem usmerjevalnem drevesu. Otroci korena drevesa so vsa vozlišča, ki imajo z danim vozliščem skupne vse števke, razen zadnje. Pod njimi se nahajajo vozlišča, ki se od lokalnega vozlišča razlikujejo samo v zadnjih dveh števkah. Tudi ostali nivoji drevesa se tvorijo po istem pravilu, na najnižjem nivoju se nahajajo vozlišča, ki se s korenom ujemajo samo v prvi števki. Primer usmerje-

valnega drevesa za vozlišče 23002 je podan na sliki 3.1.



Slika 3.1: Primer usmerjevalnega drevesa, ki ima koren v vozlišču 23002.

Usmerjanje sporočil proti vozlišču 23002 je zelo preprosto: sporočilo potuje po usmerjevalnem drevesu proti korenu, dokler ne doseže ciljnega vozlišča. Dolžina usmerjevalne poti med začetnim in končnim vozliščem je določena s poddrevesom, ki ju povezuje. Zaradi spremenljive strukture prekrivnega omrežja se lahko zgodi, da se usmerjevalno drevo katerega izmed vozlišč razcepi na več disjunktnih poddreves. V tem primeru si pri usmerjanju pomagamo z obročem, ki ga tvorijo vozlišča v množicah listov.

Izbira sosedov pri sistemu Pastry ni strogo določena, posebej pri vnosih v usmerjevalni tabeli, katerih identifikatorji se slabo ujemajo z identifikatorjem lokalnega vozlišča. Na primer, vozlišče 21330 s slike 3.1 lahko izbira med veliko vozlišči, ki se začnejo s predpono $23 * **$, medtem ko je vozlišč s predpono $2133*$ precej manj.

3.2.2 Usmerjanje

Algoritem usmerjanja sporočil v sistemu Pastry je prikazan v psevdokodi 3.1. Algoritem se izvede vsakič, ko sporočilo s ključen D prispe do vozlišča z identifikatorjem A. Pri opisu algoritma uporabljamo naslednje oznake:

- R_l^i: vnos v usmerjevalni tabeli R, vrstica l ($0 \leq l \leq \lfloor 128/b \rfloor$), stolpec i ($0 \leq i \leq 2^b$).

- L_i: i-ti najbližji identifikator vozlišča v množici listov L ($-\lfloor |L|/2 \rfloor \leq i \leq \lfloor |L|/2 \rfloor$). Pri tem imajo negativni indeks i identifikatorji vozlišč, ki so manjši,

pozitiven indeks pa identifikatorji vozlišč, ki so večji od identifikatorja trenutnega vozlišča.

- D_l: vrednost l-te števke v ključu sporočila D,
- $shl(A, B)$: dolžina skupne predpone (v števkah) med ključema A in B.

Izpis izvorne kode 3.1: Psevdokoda algoritma za usmerjanje sporočil v sistemu Pastry.

```
1  if (L_{-⌊|L|/2⌋} ≤ D ≤ L_{⌊|L|/2⌋}) {
2    // D se nahaja v mnozici listov.
3    poslji sporocilo tistemu vozliscu L_i, pri katerem je
     |D − L_i| najmanjsa;
4  }
5  else {
6    // uporabi usmerjevalno tabelo.
7    l = shl(D, A);
8    if (R_l^{D_l} ≠ null) {
9      posreduj sporocilo R_l^{D_l};
10   }
11   else {
12     // ta primer se zgodi zelo redko.
13     posreduj sporocilo tistemu vozliscu T ∈ L ∪ R ∪ M,
       pri cemer velja, da je shl(T, D) ≥ l, |T − D| < |A − D|;
14   }
15 }
```

Pri vsakem prispelem sporočilu najprej preverimo, ali se njegov ključ nahaja v območju, ki ga pokriva množica listov (vrstica 1). V tem primeru posreduje vozlišče sporočilo neposredno ciljnemu vozlišču, torej vozlišču v množici listov, katerega identifikator je najbližji ključu sporočila (vrstica 3). To vozlišče je lahko tudi lokalno vozlišče.

Če se ključ ne nahaja v območju, ki ga pokriva množica listov, se uporabi usmerjevalna tabela. Sporočilo se posreduje vozlišču, katerega dolžina predpone, ki se ujema s ključem sporočila, je vsaj za eno števko večja od tiste pri lokalnem vozlišču (vrstice 6-10). V določenih primerih se lahko zgodi, da je iskani vnos v usmerjevalni tabeli prazen oziroma vozlišče, na katerega kaže, ni dosegljivo (vrstice 11-14). Takrat se sporočilo posreduje vozlišču, katerega dolžina ujemajoče se predpone je vsaj tolikšna, kot pri lokalnem vozlišču, poleg tega pa je numerično bliže ključu od lokalnega vozlišča. Tako vozlišče se mora nahajati v množici listov, razen če ni lokalno vozlišče tisto, ki je numerično najbližje ključu sporočila.

Pri tem usmerjevalnem algoritmu se sporočilo na vsakem koraku približuje proti ciljnemu vozlišču, ker ga na vsakem koraku pošljemo vozlišču, za katerega velja:

- bodisi si s ključem sporočila deli daljšo predpono od lokalnega vozlišča,
- bodisi je ujemajoča predpona enaka kot pri lokalnem vozlišču, je pa numerično bližje ključu od njega.

3.2.3 Obravnavanje prihodov vozlišč

Ko se novo vozlišče želi prijaviti v omrežje sistema Pastry, mora najprej inicializirati svojo usmerjevalno tabelo in množico listov, nato pa obvesti ostala vozlišča o svojem prihodu. Predpostavimo, da novo vozlišče na začetku pozna samo naslov vozlišča A, ki je že prijavljeno v omrežje (ta naslov mu navadno poda uporabnik kot vhodni argument programa). Naj bo identifikator novega vozlišča označen z X (generiranje identifikatorjev vozlišč je specifično za posamezno aplikacijo. Navadno jih določamo z uporabo preslikovalne funkcije SHA-1 nad IP naslovom računalnika oziroma njegovim javnim ključem). Vozlišče X najprej pošlje A t.i. sporočilo "join", katerega ključ je enak X. Sistem Pastry usmerja to sporočilo k vozlišču Z, katerega identifikator je numerično najbližje X.

Vsakič, ko katero od vozlišč na poti od A proti Z prejme sporočilo "join", pošlje svojo usmerjevalno tabelo in množico listov vozlišču X. X s pomočjo teh podatkov zapolni svojo usmerjevalno tabelo in množico listov. Na koncu X obvesti vozlišča o svojem prihodu. Na ta način se ažurira stanje v omrežju.

Ob predpostavki, da se A nahaja v bližini vozlišča X, se njegova množica sosedov uporabi kot osnova za nastavitev množice sosedov X. Zatem se lotimo zapolnjevanja usmerjevalne tabele, od vrstice 0 naprej. Oglejmo si najbolj splošen primer, ko identifikatorja vozlišč A in X nimata skupne predpone. Naj A_i označuje i-to vrstico v usmerjevalni tabeli vozlišča A. V A_0 se nahajajo naslovi vozlišč, katerih identifikatorji nimajo skupne predpone z vozliščem A. Ta vrstica torej vsebuje vnose, ki so primerni za X_0. Ostali nivoji v usmerjevalni tabeli vozlišča A niso primerni za X, ker si A in X ne delita skupne predpone.

Po drugi strani pa lahko vnose za X_1 dobimo iz B_1, kjer je B prvo vozlišče na poti od A do Z. Vnosi v B_1 si z X zagotovo delijo skupno predpono, saj imata X in B enako prvo številko v identifikatorju. Na podoben način pridobi X vnose za X_2 iz vozlišča C, ki je drugo vozlišče na poti proti Z, ter za vse ostale vrstice X_i usmerjevalne tabele.

Na koncu pošlje X vsebino svojo usmerjevalno tabelo, množico listov in množico sosedov vsem vozliščem, ki jih najde v teh strukturah. Ta vozlišča uporabijo dobljene informacije za ažuriranje svojih struktur za usmerjanje. Mogoče je dokazati, da je v tem trenutku vozlišče X zmožno usmerjati in sprejemati sporočila ter sodelovati v omrežju sistema Pastry. Zahtevnost prijave novega vozlišča v omrežje, v smislu števila izmenjanih poročil, je $O(log_{2^b} N)$, pri čemer je kvocient približno enak $C = 3 * 2^b$.

Za nadzorovanje sočasnega prihajanja oziroma odhajanja več vozlišč Pastry upo-

rablja optimističen pristop. Ta pristop je smiseln, ker prihodi/odhodi vozlišč vplivajo samo na majhno število obstoječih vozlišč v sistemu in so zasičenja mrežnih povezav zelo redka. Kadarkoli vozlišče A pošlje vsebino svojih usmerjevalnih struktur vozlišču B, sporočilu doda časovni žig (angl. time stamp). Na podlagi poslanih podatkov B ažurira svoje usmerjevalne strukture. Po določenem času B ponovno zahteva od A pošiljanje ažurnega stanja usmerjevalnih struktur. Zahtevi doda časovni žig zadnjega ažuriranja, kar omogoča A, da preveri če se je njegovo stanje sploh spremenilo od zadnjega ažuriranja. V primeru, da se je, A ponovno pošlje podatke o svojih usmerjevalnih strukturah vozlišču B.

3.2.4 Obravnavanje odhodov in odpovedi vozlišč

V omrežju Pastry lahko vozlišča odpovedo ali odidejo brez opozorila. Vozlišče se obravnava kot odsotno, ko njegovi neposredni sosedi v domenskem prostoru identifikatorjev ne morejo več komunicirati z njim.

Pri zamenjavi odsotnega vozlišča v množici listov, lastnik kontaktira vozlišče s skrajnim indeksom v množici listov (bodisi na pozitivni strani, bodisi na negativni, odvisno od tega, kje se je nahajalo odsotno vozlišče) in zahteva prenos njegove množice listov. Na primer, če se indeks odsotnega vozlišča nahaja v območju $\lfloor |L|/2 \rfloor < i < 0$, se kontaktira vozlišče s skrajnim indeksom iz $L_{-\lfloor |L|/2 \rfloor}$. Naj bo prejeta množica listov označena z L'. Ta množica se delno prekriva s trenutno množico listov L, vključuje pa tudi vozlišča s podobnimi identifikatorji, ki se ne nahajajo v L. Izmed teh se izbere najprimernejše. Preden se to vozlišče vključi v L se preveri, ali je sploh prisotno v omrežju. Ta postopek zagotavlja popravilo množice listov razen v primeru, ko istočasno izpade $\lfloor |L|/2 \rfloor$ vozlišč s sosednjimi identifikatorji. Zaradi razpršenosti vozlišč je tak izpad zelo malo verjeten tudi pri omrežjih z manj vozlišči.

Izpad vozlišča, ki se nahaja v usmerjevalni tabeli katerega izmed ostalih vozlišč, se zazna, ko ga poskuša tisto vozlišče kontaktirati in na svoje sporočilo ne dobi odziva. Ta dogodek sicer ne podaljša bistveno časa usmerjanja sporočila (ker to še vedno lahko usmerimo k drugemu vozlišču, ki ima enako predpono), vseeno pa moramo poiskati zamenjavo za izpadlo vozlišče, da ohranimo konsistentnost usmerjevalne tabele.

Pri popravljanju okvarjenega vnosa R_l^d usmerjevalne tabele vsako vozlišče najprej kontaktira vozlišče, na katerega kaže vnos R_l^i, $i \neq d$ v isti vrstici tabele, in zahteva prenos vnosa, ki se nahaja na istem položaju v usmerjevalni tabeli oddaljenega vozlišča. V primeru, da noben od vnosov v vrstici l ne vsebuje reference na obstoječe vozlišče v omrežju z ustrezno predpono, vozlišče kontaktira vnos R_{l+1}^i, $i \neq d$. Na ta način se sčasoma zelo verjetno najde ustrezno vozlišče, če le-to obstaja.

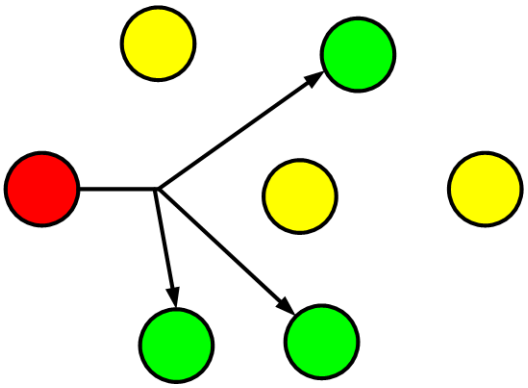
Množica sosedov se navadno ne uporablja pri usmerjanju sporočil, igra pa zelo pomembno vlogo pri vzdrževanju konsistentnosti usmerjevalnih tabel, saj skrbi za izmenjavo informacij o sosednjih vozliščih. Pri tem poskuša vozlišče periodično kontaktirati vsako vozlišče da preveri, če je le-to še prisotno v omrežju. Če se

vozlišče ne odziva, se pošlje ostalim vozlišča v množici sosedov zahtevo po pošiljanju svojih množic sosedov.

Rezultati poskusov kažejo [23], da so ti algoritmi učinkoviti pri popravljanju usmerjevalnih struktur v primeru izpadov vozlišč.

3.2.5 Oddajanje več prejemnikom

Oddajanje več prejemnikom (angl. multicast) je način komuniciranja v računalniškem omrežju, pri katerem pošiljatelj pošlje določeno sporočilo več naslovnikom hkrati. Njegova prednost je predvsem učinkovita raba mrežne infrastrukture, saj se sporočilo pošlje po določeni povezavi samo enkrat, četudi ga moramo dostaviti več vozliščem. To je mogoče, ker vozlišča na usmerjevalni poti skrbijo za razmnoževanje sporočila, ko je to potrebno. Oddajanje več prejemnikom je prikazano na sliki 3.2.



Slika 3.2: Oddajanje več prejemnikom.

V omrežjih P2P se večinoma uporablja oddajanje več prejemnikom na aplikacijskem nivoju (angl. application multicast). Od oddajanja več prejemnikom na nivoju IP se razlikuje po tem, da se sporočila ne razmnožujejo na mrežnih usmerjevalnikih (angl. routers) temveč že na samih vozliščih v prekrivnem omrežju. Cilj oddajanja več prejemnikom na aplikacijskem nivoju je izgradnja in vzdrževanje učinkovitega prekrivnega omrežja za prenos podatkov. Ker se pri oddajanju na aplikacijskem nivoju po določeni povezavi pošilja več identičnih paketov, ta način ni tako učinkovit kot oddajanje na nivoju IP.

Za merjenje kvalitete prekrivnih omrežij s stališča oddajanja več prejemnikom na aplikacijskem nivoju se uporabljata dve meri iz [38]:

- obremenitev: ta mera se nanaša na posamezno povezavo in določa povprečno število identičnih sporočil, ki se pošljejo čez povezavo pri enem samem oddajanju,

- razteg: ta mera je definirana kot razmerje med dolžino poti med dvema vozliščema v prekrivnem omrežju in dolžino v fizičnem omrežju

V splošnem se pri ocenjevanju protokolov za oddajanje več prejemnikom na aplikacijskem nivoju uporabljajo 3 kriteriji:

- kvaliteta usmerjevalnih poti v prekrivnem omrežju: kvaliteto drevesa multicast določamo z zgoraj navedenimi metrikami (obremenitev, razteg, stopnja vozlišč),

- robustnost prekrivnega omrežja: ker so vozlišča manj stabilna kot mrežni usmerjevalniki, je za protokole oddajanja več prejemnikom na aplikacijskem nivoju ključnega pomena učinkovito obravnavanje izpadov vozlišč. Robustnost protokolov multicast se meri na podlagi motnje pri prenosu podatkov, ki nastane pri izpadu vozlišč. Poleg tega je pomemben tudi čas, ki je potreben, da se vzpostavi stabilno stanje v omrežju,

- stroški vzdrževanja omrežja: za učinkovito izrabo mrežnih virov morajo biti stroški vzdrževanja omrežja čim manjši. Manjši stroški vzdrževanja hkrati pomenijo tudi boljšo razširljivost omrežja.

Sistem Pastry sam po sebi ne implementira algoritmov za oddajanje več prejemnikom. Obstajajo sicer sistemi, ki implementirajo te algoritme in pri tem uporabljajo Pastry (npr. SCRIBE [39]), vendar so ti sistemi namenjeni za komunikacijo v skupinah (angl. group communication) in distribucijo multimedijskih vsebin, zato so njihovi protokoli za oddajanje preveč zapleteni in generirajo preveč mrežnega prometa. Zato smo se odločili razširiti Pastry z lastno implementacijo algoritma za oddajanje več prejemnikom, ki bo preprosta in učinkovita. Psevdokoda algoritma je prikazana na shemi 3.2. Algoritem se izvede vsakič, ko sporočilo multicast s predpono α prispe do vozlišča z identifikatorjem A. Pri opisu algoritma uporabljamo podobne oznake kot pri psevdokodi 3.1:

- R_i^j: vnos v usmerjevalni tabeli R, vrstica i ($0 \leq l \leq \lfloor 128/b \rfloor$), stolpec j ($0 \leq i \leq 2^b$).

- α: predpona za oddajanje. Pri oddajanju več prejemnikom sporočilo prečka vsa vozlišča, katerih predpona je enaka α.

- $shl(A, \alpha)$: dolžina skupne predpone (v številkah) med ključema A in predpono za oddajanje α.

- id_seje: identifikator seje multicast. Uporablja se za preprečevanje večkratnega pošiljanja sporočil multicast istemu vozlišču.

Izpis izvorne kode 3.2: Psevdokoda algoritma za oddajanje več prejemnikom.

```
1  // ce se identifikator trenutnega vozlisca ne ujema dovolj
2  // s predpono za oddajanje se oddajanje niti ne zacne.
3  if (shl(A, α) < dolzina(α))
4    return;
5
6  // ce je sporocilo v okviru dane seje multicast ze obiskalo
7  // vozlisce A, se zavrze.
8  if (multicast_seja_obstaja(id_seje))
9    return;
10
11 i = dolzina(α);
12
13 // sporocilo multicast posljemo vsem vozliscem, katerih
14 // indeks vrstice v usmerjevalni tabeli je enak dolzini
15 // ujemalne predpone.
16 for (j = 0; j < b; j++) {
17
18   // sporocila posiljamo samo obstojecim vozliscem.
19   if (R_i^j == null)
20     continue;
21
22   // sporocila ne posiljamo samemu sebi.
23   if (R_i^j == A)
24     continue;
25
26   // predponi α dodaj znak j.
27   α_1 = α ∘ j;
28
29   posreduj sporocilo s predpono α_1 vozliscu R_i^j;
30 }
31
32 izvedi lokalno operacijo na podlagi vrste sporocila multicast;
```

Ob vsakem prispelem sporočilu najprej preverimo, ali se identifikator lokalnega vozlišča ujema z multicast predpono α sporočila (vrstici 3 in 4). Cilj algoritma za oddajanje več prejemnikom, ki smo ga implementirali, je obiskati samo vozlišča, katerih predpone identifikatorjev se ujemajo z α.

Sledi preverjanje, ali ni katero izmed sporočil iz iste seje multicast že prečkalo danega vozlišča (vrstici 8 in 9). V ta namen nosijo sporočila s seboj identifikator seje. V primeru, da je dano sporočilo prvo iz seje multicast, ki je obiskalo vozlišče, se spusti naprej, identifikator njegove seje multicast pa se shrani v posebno podatkovno

strukturo, ki omogoča hitro iskanje. Vsako naslednje sporočilo, katerega identifikator seje multicast je že shranjen v podatkovni strukturi, se zavrže. Na ta način poskrbimo, da bodo sporočila iz iste seje obiskala poljubno vozlišče največ enkrat (pot sporočil od pošiljatelja do prejemnikov dobi obliko drevesa). To je ključnega pomena pri prekrivnih omrežjih, ki vsebujejo cikle, saj bi se v njih brez zavračanja podvojenih sporočil omrežje preobremenilo.

Preverjanju o nepodvajanju sporočila sledi pošiljanje (vrstice 11-30). Sporočilo multicast se razmnoži in posreduje vsem vozliščem katerih indeks vrstice v usmerjevalni tabeli je enak dolžini ujemalne predpone. Pošiljanje se izvaja na nivoju DHT. Kljub temu, da to pošiljanje ni tako učinkovito kot pošiljanje na nivoju fizičnega omrežja (pot, ki jo prepotuje sporočilo v omrežju DHT, je navadno nekajkrat daljša kot pri pošiljanju v fizičnem omrežju, kar ima za posledico večji promet in daljše zakasnitve), smo se zanj odločili iz preprostega razloga: na ta način je mogoče dostaviti sporočilo multicast, tudi ko se njegova predpona ne ujema z nobenim izmed identifikatorjev vozlišč. V takih primerih nam usmerjevalni algoritem sistema Pastry zagotavlja, da se bodo sporočila z enako predpono usmerjala proti istemu končnemu vozlišču. Pri neposrednem pošiljanju v fizičnem omrežju to ni mogoče: če se nobeden izmed IP naslovov vozlišč ne ujema z danim naslovom, pošiljanje sporočila ni mogoče.

Pred samim pošiljanjem se ujemalni predponi še doda znak j. Na ta način se postopoma specializira nabor ciljnih vozlišč, ko potujemo proti listom drevesa multicast. V primeru, da bi v vsakem vozlišču pošiljali sporočilo vsem vnosom v usmerjevalni tabeli, katerih predpona je enaka α, bi bile pogoste situacije, ko bi se sporočilo multicast poslalo določenemu vozlišču iz več različnih sosedov. V teh primerih bi se bistveno povečalo število zavrženih sporočil in s tem tudi količina odvečnega mrežnega prometa.

Po razpošiljanju sporočila oddaljenim vozliščem v prekrivnem omrežju se izvede še lokalna operacija. Ta je odvisna od vrste sporočila multicast: pri iskanju virov, na primer, se preišče lokalna shramba kazalcev, da se ugotovi, ali obstaja kakšen kazalec na vir, ki ustreza iskalni poizvedbi v sporočilu.

Primer oddajanja več prejemnikom v sistemu Pastry je prikazan na sliki 3.3. Recimo, da želimo iz vozlišča 12300 poslati sporočilo vsem vozliščem, katerih identifikator se začne s predpono 123:

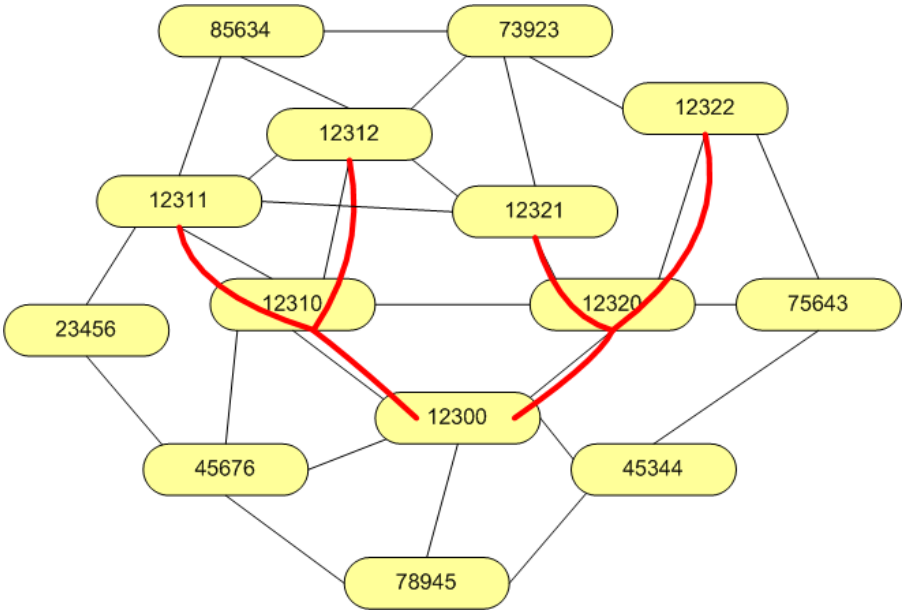
1. Edini soseda vozlišča 12300, ki ustrezajo temu pogoju, so 12310, 12320 in 12313, zato se sporočilo multicast najprej pošlje njim. Pri tem se ujemalni predponi sporočila multicast, ki ga pošljemo vozliščema 12310 in 12313 doda števka 1 (dobimo ujemalno predpono 1231), predponi sporočila, ki ga pošljemo vozlišču 12320, pa se doda 2 (dobimo ujemalno predpono 1232).

2. Vozlišče 12310 ima 3 sosede, katerih predpona je enaka 1231: 12311, 12312 in 12313. Sporočila multicast se zato posredujejo tem vozliščem. Pred pošiljanjem

se jim še ustrezno podaljšajo predpone, ki se na tej stopnji izenačijo z identifikatorji vozlišč. Enak postopek se ponovi pri vozliščih 12313 in 12320. Pri 12313 se sporočilo posreduje vozlišču 12310 (ujemalna predpona se podaljša v 12310), pri 12320 pa vozliščema 12321 in 12322 (ujemalni predponi se podaljšata v 12321 in 12322).

3. Ko sporočilo iz vozlišča 12310 prispe do 12313, sistem zazna na podlagi identifikatorja seje, da je identično sporočilo že potovalo preko tega vozlišča. Sporočilo se zato zavrže. Enaka se zgodi s sporočilom, ki ga vozlišče 12313 pošlje 12310.

   Ostanejo še vozlišča 12311, 12312, 12321 in 12322. Ker so v tretjem koraku ujemalne predpone že enake identifikatorjem teh vozlišč, se ne morejo več podaljševati. Sporočila multicast se zato ne posredujejo več naprej.

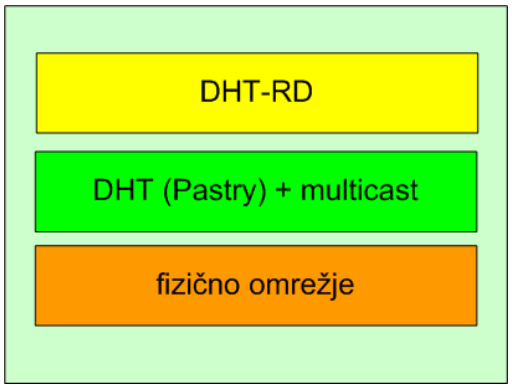


Slika 3.3: Primer oddajanja več prejemnikom v sistemu Pastry.

3.3 DHT-RD

Za objavljanje in odkrivanje virov v DHT se uporablja plast DHT-RD (slika 3.4). Pri tem uporablja funkcije sistema Pastry v nespremenjeni obliki (sistem smo razširili samo z algoritmom za oddajanje več prejemnikom, ki je opisan v razdelku 3.2.5). Uporaba algoritmov DHT v nespremenjeni obliki se izkaže za veliko prednost RD-DHT, saj lahko na ta način izkoriščamo napredek na področju strukturiranih sistemov P2P za učinkovitejše odkrivanje virov. Trenutno obstaja že precej algoritmov

za zmanjševanje mrežnega prometa, za optimizacijo usmerjanja v sistemih DHT in za čim boljše prileganje prekrivnega omrežje fizičnemu omrežju [40, 41, 42, 43]. Sistemu DHT-RD ni treba poznati podrobnosti usmerjanja sporočil po prekrivnem omrežju, za vse to skrbi plast DHT.



Slika 3.4: Arhitektura našega sistema za objavljanje in odkrivanje virov.

V okviru objavljanja in odkrivanja virov je sistem DHT-RD zadolžen za različne naloge (podrobnejši opis posameznih nalog bo podan v razdelkih, ki sledijo):

- izbira atributa v dani poizvedbi, ki se bo preslikal v iskalni ključ za usmerjanje v prekrivnem omrežju, generiranje ključa,
- shranjevanje podatkov o objavljenih virih in njihovo periodično osveževanje,
- izvajanje uporabniško definiranih poizvedb nad shranjenimi kazalci na vire.

Sistem RD-DHT deluje na podoben način kot sistemi za odkrivanje virov, ki smo jih opisali v razdelku 2.3. Od njih se razlikuje v načinu generiranju ključa, ki se uporablja za usmerjanje sporočila v omrežju DHT. Medtem, ko se pri vseh sistemih iz razdelka 2.3 usmerjevalni ključ generira na podlagi vrednosti atributov virov, se pri DHT-RD generira iz imen atributov. Za lažje razumevanje si bomo to razliko ogledali na praktičnem primeru. Recimo, da želimo v sistemu Grid objaviti računski vir z dvema atributoma: $frekvenca\_cpu = 2.3GHz$ in $velikost\_pomnilnika = 1GB$. Preden pošljemo sporočilo v omrežje DHT, moramo generirati usmerjevalni ključ. Ta ključ se pri obstoječih sistemih za odkrivanje virov generira z uporabo zgoščevalnega algoritma SHA-1 nad vrednostjo 2.3 oziroma 1, odvisno kateri atribut izberemo. Tudi pri sistemu DHT-RD uporabimo pri generiranju ključa algoritem SHA-1, le da ga tu izvedemo nad nizoma “frekvenca_cpu” oziroma “velikost_pomnilnika”.

Posledica generiranja usmerjevalnih ključev iz imen atributov je združevanje kazalcev na vire, ki se indeksirajo po istem atributu, na skupnih vozliščih. Taka ureditev podatkov o virih nam omogoča preprosto implementacijo večparametrskih

območnih poizvedb v sistemih DHT: ustvarimo sporočilo, ki vsebuje poizvedbo, kot usmerjevalni ključ pa mu določimo preslikano ime izbranega atributa v poizvedbi. Sporočilo predamo DHT, ta pa ga posreduje vozlišču, ki je zadolženo za shranjevanje podatkov z danim ključem. Na tem vozlišču je shranjenih več virov enakega tipa, z enakim imenom izbranega atributa. S pomočjo poizvedbe, ki je shranjena v sporočilu, izberemo tiste vire, ki ustrezajo iskalnim pogojem. Ko imamo vse kazalce na vire zbrane na enem samem vozlišču, je izvajanje večparametrskih območnih poizvedb preprosto.

Iz zgornjega opisa lahko razberemo, da je glavna prednost sistema DHT-RD možnost uporabe večparametrskih območnih poizvedb brez spreminjanja algoritmov DHT. Zaradi periodičnega osveževanja podatkov o objavljenih virih (tega bomo podrobneje opisali v razdelku 3.3.1) je sistem odporen tudi na izpade vozlišč. V primeru izpada vozlišča, ki je zadolženo za določeno ime atributa, je iskanje virov na podlagi tega imena onemogočeno samo za krajši čas, do naslednjega osveževanja. Takrat bo sistem DHT usmeril podatke k enemu izmed vozlišč, ki so še prisotni v prekrivnem omrežju. Tudi količina mrežnega prometa, ki se generira pri odkrivanju virov, je precej manjša kot pri poplavljanju, poleg tega pa nam sistem DHT zagotavlja dospetje poizvedbe do vozlišča z ustreznimi viri za povpraševanje.

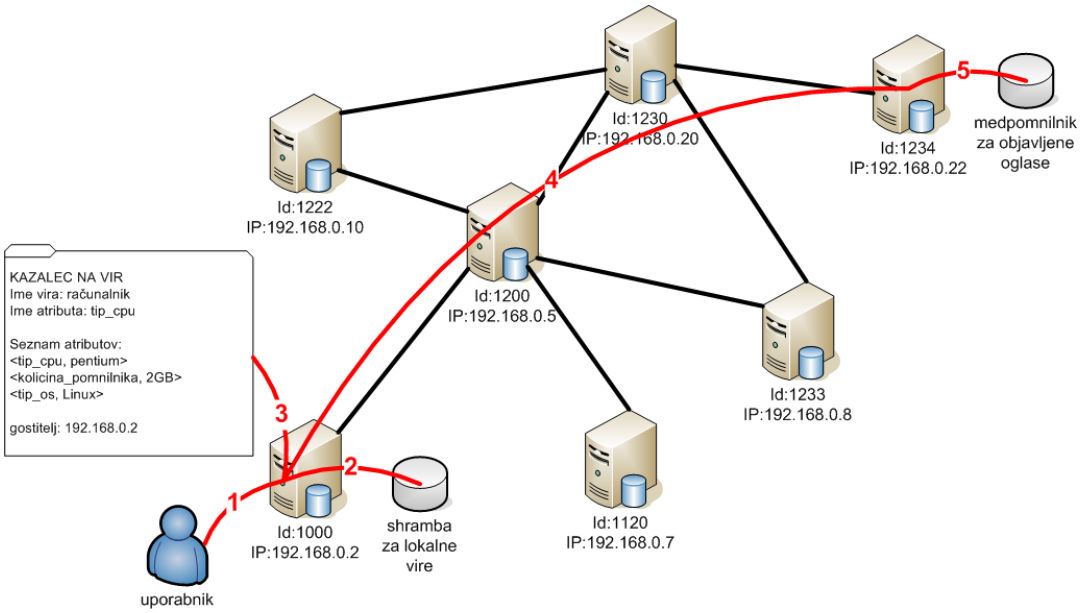
Seveda pa ima centralizirano shranjevanje kazalcev na vire veliko slabost: zelo neenakomerno obremenitev vozlišč v prekrivnem omrežju. Vozlišča, na katerih so shranjeni kazalci, sprejmejo in oddajo več sporočil, zaradi česar sta njihova procesorska obremenitev in obremenitev mrežnih povezav večji. Problematična je tudi večja poraba njihovega pomnilniškega prostora, zaradi shranjevanja kazalcev na vire. Ta poraba se veča linearno s številom objavljenih virov v omrežju.

Z namenom, da bi v čim večji meri odpravili zgornje slabosti, smo se odločili nekoliko spremeniti osnovni koncept sistema DHT-RD. Usmerjevalni ključ smo razdelili na 2 dela: prvi del, ki predstavlja predpono ključa, se še vedno tvori iz imena atributa. Drugi del pa predstavlja pripona, ki jo generiramo na različne načine (bodisi naključno, bodisi na podlagi IP naslova vozlišča, na katerem je shranjen vir ipd.). S tem dosežemo, da se pri objavljanju podatkov o virih kazalci za določen atribut ne usmerjajo k enemu samemu vozlišču, ampak se porazdelijo med različna vozlišča v omrežju DHT. Na ta način s prerazporedi tudi obremenitev vozlišč. Algoritma sistema DHT-RD za objavljanje in odkrivanje virov sta podrobneje opisana v razdelkih 3.3.1 in 3.3.2.

3.3.1 Objavljanje virov

Postopek objavljanja virov v sistemu DHT-RD je prikazan na sliki 3.5. Razdelimo ga lahko na 5 stopenj:

1. Uporabnik se odloči objaviti določen vir v sistemu Grid, zato izvede metodo



Slika 3.5: Objavljanje virov v sistemu DHT-RD.

publish iz programskega vmesnika sistema DHT-RD. Kot argument metode poda referenco na vir, ki ga želi objaviti.

2. Podatki o viru se najprej dodajo v shrambo za lokalne objekte. Ta služi za periodično obnavljanje kazalcev na vir in osveževanje vrednosti dinamičnih atributov. S periodičnim obnavljanjem kazalcev učinkovito rešimo problem osamljene točke odpovedi, ki se pojavlja v večini centraliziranih in hierarhičnih sistemov za odkrivanje virov. V primeru izpada vozlišča, ki hrani množico kazalcev, so ti nedostopni samo do naslednje obnove. Takrat se kazalci zopet objavijo v prekrivnem omrežju, algoritmi DHT pa poskrbijo, da se objave usmerjajo proti nadomestnemu vozlišču.

3. Po uspešnem dodajanju v shrambo objektov se za vsak atribut v viru tvori ločen kazalec. Ta vsebuje osnovne podatke o viru in njegovi lokaciji: seznam statičnih atributov, ki služijo za izločanje neustreznih virov v prvi stopnji odkrivanja (glej razdelek 3.4 za podrobnejši opis postopka); IP naslov vozlišča, na katerem je shranjen vir. Za vsak kazalec se generira usmerjevalni ključ. Ta je sestavljen iz dveh delov. Prvi del predstavlja predpona, ki jo dobimo z uporabo zgoščevalnega algoritma SHA-1 nad združenim imenom vira in imenom atributa (uporaba imena vira poleg imena atributa nam omogoča ločevanje istoimenskih atributov pri različnih virih). To nam omogoča učinkovito porazdeljevanje kazalcev po vozliščih, saj je verjetnost, da se bodo različna imena

virov in atributov preslikala v isti ključ, zelo majhna. Drugi del ključa predstavlja pripona, ki služi za porazdeljevanje kazalcev z istim imenom vira in atributa med več vozlišč. S tem uravnotežimo njihovo obremenjenost. Pripono generiramo bodisi naključno, bodisi z uporabo zgoščevalne funkcije nad delom naslova IP gostitelja vira (ta pristop je še posebej učinkovit, saj omogoča združevanje kazalcev na vire, ki se nahajajo v istem podomrežju, na skupnem vozlišču).

4. Sledi objavljanje kazalcev v sistemu DHT. Sporočila se usmerjajo po prekrivnem omrežju toliko časa, dokler ne prispejo do korenskega vozlišča (vozlišča, ki je zadolženo za shranjevanje danega kazalca na vir). Korensko vozlišče je tisto vozlišče, ki ima od vseh najbolj podoben identifikator ključu sporočila.

5. Ko sporočilo s kazalcem na vir prispe do korenskega vozlišča, se oglas odda v medpomnilnik. S tem je proces obnavljanja končan.

V okviru kazalcev na vire velja še omeniti, da obstajata dve vrsti atributov virov:

- statični: zanje je značilno, da se njihove vrednosti pri posameznem viru ne spreminjajo. Primeri statičnih atributov so: tip procesorja, skupna količina pomnilnika, vrsta operacijskega sistema ipd.

- dinamični: pri teh atributih se njihova vrednost lahko spreminja v času. Primeri teh atributov so: zasedenost procesorja, količina zasedenega pomnilnika, število opravil v čakalni vrsti ipd.

Večina sistemov za odkrivanje virov, ki so osnovani na porazdeljenih zgoščenih tabelah ima težave pri indeksiranju dinamičnih atributov, saj morajo seliti kazalce na drugo vozlišče, ko se katera izmed vrednosti atributov spremeni. Ker se pri sistemu DHT-RD kazalci objavljajo na podlagi imen atributov (ne pa na podlagi njihovih vrednosti), tu objavljanje in iskanje na podlagi dinamičnih atributov ni problematično.

Ker se večina dinamičnih atributov spreminja precej pogosteje kot se izvajajo iskalne zahteve, smo se odločili, da kazalcev na vire ne bomo ažurirali vsakič, ko se spremeni eden od njegovih atributov, temveč jih bomo osveževali v rednih časovnih intervalih. Na ta način bomo generirali manj mrežnega prometa in s tem manj obremenjevali omrežje. Slabost tega pristopa je sicer neažurnost kazalcev, ki pa je rešljiva na preprost način: vsak uporabnik pri procesu odkrivanja določi, kolikšna je lahko največja starost vrednosti atributov, da jih bo še upošteval pri izbiranju kazalcev. Tako ne moremo zavreči nekega kazalca na podlagi vrednosti atributa, ki je zastarela. Ta pristop je podrobneje opisan v razdelku o odkrivanju virov (glej 3.3.2).

Psevdokoda algoritma za objavljanje virov je prikazana na shemi 3.3.

Izpis izvorne kode 3.3: Psevdokoda algoritma za objavljanje virov v sitemu DHT-RD.

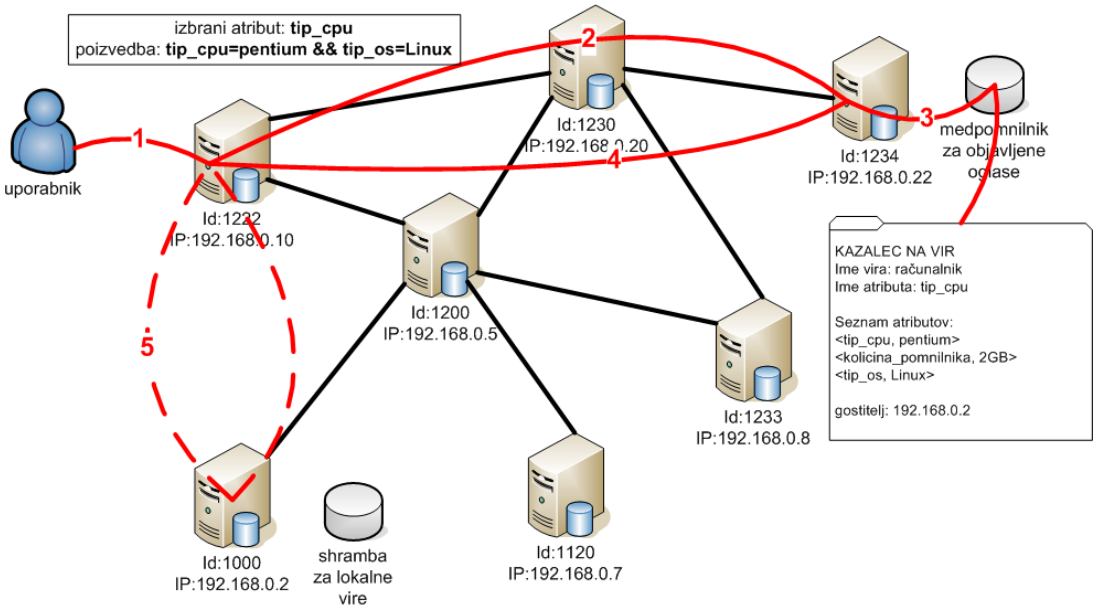
```
1  // doba zastaranja kazalca na vir je enaka stirikratniku
2  // osvezevalne periode. to pomeni, da ima lastnik vira
3  // stirikrat moznost poslati zahtevo za podaljsanje
4  // veljavnosti kazalca, preden se ta izbrise iz
5  // medpomnilnika.
6  doba_zastaranja = 4 * osvezevalna_perioda;
7
8  // najprej shranimo vir lokalno.
9  if (!lokalni_viri.dodaj(vir, osvezevalna_perioda, doba_zastaranja)) {
10   return false;
11 }
12
13 // objavi kazalce na vir po oddaljenih vozliscih
14 for (i = 0; i < stevilo_atributov_vira; i++) {
15   // predpona kljuca je sestavljena iz imena vira in
16   // imena atributa.
17   predpona = SHA1(ime_vira ∘ ime_atributa[i]);
18   // pripono generiramo nakljucno. lahko uporabimo, na
19   // primer, tudi razprsevalno funkcijo nad IP naslovom
20   // gostitelja.
21   pripona = random();
22   // generiraj kljuc za usmerjanje objave po prekrivnem omrezju.
23   kljuc_vir = predpona ∘ pripona;
24
25   kazalec_vir = new KazalecVir(kljuc_vir, IP_gostitelj, atributi_vira);
26
27   poslji kazalec na vir oddaljenemu vozliscu;
28 }
```

3.3.2 Odkrivanje virov

Odkrivanje virov je obsežnejše od objavljanja. V grobem ga delimo na 2 fazi. Prva faza zajema izbiranje ustreznih kazalcev na vire. Pri tem se upoštevajo samo statični atributi ter tisti dinamični atributi, katerih vrednosti še niso zastarele. To izbiranje se vrši na korenskih vozliščih, kjer so shranjeni kazalci. Njegov cilj je izločiti čim več neustreznih virov (to pomeni, da bo v drugi fazi treba kontaktirati manj vozlišč). Prva faza se zaključi s pošiljanjem kazalcev, ki ustrezajo poizvedbi, začetnemu vozlišču, ki je sprožilo iskanje.

Druga faza ni obvezna, po potrebi izvede jo uporabnik pri poizvedbah, ki vse-

bujejo dinamične atribute. V tej fazi se poizvedbe pošljejo iz začetnega vozlišča podmnožici vozlišč, na katera kažejo kazalci, ki smo jih izbrali v prvi fazi. Velikost podmnožice definiramo kot parameter sistema. Na ta način preprečimo, da bi se pošiljanje poizvedb v drugi fazi izrodilo v razprševanje (angl. broadcast). V drugi fazi odkrivanja se poizvedbe izvedejo neposredno na vozliščih gostiteljih virov. Na ta način zagotovimo, da se pogoji v poizvedbah primerjajo z najbolj ažurnimi vrednostmi dinamičnih atributov.



Slika 3.6: Odkrivanje virov v sistemu DHT-RD.

Celoten postopek je razdeljen na 5 stopenj in je prikazan na sliki 3.6:

1. Uporabnik izvede metodo *discover*, kateri poda 2 argumenta: poizvedbo, ki določa pogoje za izbiranje virov, ter referenco na objekt, ki služi za obdelavo odkritih virov. Ta objekt implementira uporabnik sam.

2. Izbere se atribut, ki se bo uporabil za generiranje usmerjevalnega ključa. Ta izbira je zelo pomembna, saj nam lahko precej zmanjša število vozlišč, ki jih moramo kontaktirati v drugi fazi odkrivanja. Trenutno se za izbiranje atributa uporablja naključna metoda, pri kateri se algoritem za generiranje usmerjevalnega ključa izbere naključno izmed množice atributov v poizvedbi. V prihodnosti bi to metodo lahko zamenjali s tako, ki izbere tisti atribut, ki statistično najbolje omejuje iskalno množico.

   Iz imena vira in imena izbranega atributa se s pomočjo algoritma SHA-1 generira predpona multicast. Ustvari se sporočilo z iskalno zahtevo, ki se s pomočjo

oddajanja več prejemnikom razpošlje med vsa vozlišča, ki so zadolžena za shranjevanje kazalcev, katerih ključ je zgeneriran na podlagi izbranega imena atributa.

3. Ko sporočila multicast prispejo do korenskih vozlišč z ustrezno predpono, se izvedejo poizvedbe nad vsemi kazalci za dani tip vira, ki so shranjeni na njih. Izberejo se kazalci, ki ustrezajo poizvedbi. Pri preverjanju pogojev se upoštevajo samo statični atributi in tisti dinamični atributi, katerih vrednosti še niso zastarele. Ostali atributi ne sodelujejo pri filtriranju kazalcev (predpostavi se, da je ustrezen pogoj v poizvedbi izpolnjen). Tako ločimo med tremi vrstami ujemanja posameznega vira glede na pogoje v poizvedbi: **popolno ujemanje** (vir izpolnjuje vse pogoje v poizvedbi, vrednosti dinamičnih atributov niso zastarele), **delno ujemanje** (nekatere vrednosti dinamičnih atributov vira so že zastarele, zato je treba neposredno kontaktirati gostitelja vira; ostale vrednosti ustrezajo pogojem v poizvedbi) ter **ni ujemanja** (najmanj ena izmed vrednosti atributov, ki ni zastarela, ne ustreza pogoju v poizvedbi).

4. Kazalce, ki se bodisi popolnoma, bodisi delno ujemajo s poizvedbo pošljemo vozlišču, ki je sprožilo iskanje. V primeru, da nismo našli nobenega ustreznega vira, pošljemo nazaj prazno množico. Iz prejetih kazalcev lahko začetno vozlišče razbere IP naslove gostiteljev, ki hranijo želene objekte. Za vse kazalce, ki se le delno ujemajo z dano poizvedbo, bi bilo treba poslati iskalno zahtevo neposredno gostitelju vira in preveriti ali najnovejše vrednosti za vir ustrezajo pogojem v iskalni poizvedbi. Tega preverjanja ne izvaja sistem DHT-RD, ker bi bilo v tem primeru iskanje preveč zamudno. Izvesti ga mora uporabnik sam, v razredu za obdelavo podatkov o virih. Sistem DHT-RD mu pri tem pomaga tako, da razvrsti kazalce na vire v dva seznama: v enem so viri s pravim ujemanjem, v drugem pa tisti z delnim ujemanjem. Ta način ločevanja kazalcev ima pomembno prednost: če smo uspeli odkriti dovolj virov s pravim ujemanjem, lahko tiste z delnim ujemanjem preprosto prezremo. Dodatno povpraševanje je potrebno le v primerih, ko je število odkritih virov majhno in je pomemben vsak kazalec.

5. Začetno vozlišče nad vsemi podatki o viru izvede operacijo, ki jo je kot argument podal uporabnik. Kot smo že predhodno omenili, se tu po potrebi izvede druga faza odkrivanja. V njej iz začetnega vozlišča pošljemo poizvedbe neposredno gostiteljem virov. Ti prejmejo zahtevo za preverjanje ustreznosti lokalnega vira in izvedejo poizvedbo nad ažurnimi podatki o viru. Na ta način se nedvoumno določi, ali lastnosti vira ustrezajo iskalnim pogojem. V primeru pozitivnega odgovora pošljemo podatke o lokalnem viru začetnemu vozlišču.

Psevdokoda algoritma za objavljanje virov je prikazana na shemi 3.4.

Izpis izvorne kode 3.4: Psevdokoda algoritma za odkrivanje virov v sitemu DHT-RD.

```
1  // izberi atribut iz poizvedbe, na podlagi katerega se bo
2  // generiral usmerjevalni kljuc (mogoce je bodisi nakljucno
3  // izbiranje, bodisi izbiranje na podlagi hevristik).
4  // Nas cilj je izbrati atribut, ki bo najbolj omejil mnozico
5  // potencialnih zadetkov.
6  atribut_za_iskanje = izberi__atribut(poizvedba);
7
8  // predpona kljuca je sestavljena iz imena vira in
9  // imena atributa.
10 p = poizvedba.ime_vira ∘ atribut_za_iskanje;
11
12 // registriraj callback funkcijo za procesiranje odkritih virov.
13 registriraj_funkcijo(funkcija_za_procesiranje);
14
15 // poslji poizvedbo vsem vozliscem, katerih predpona se ujema
16 // s preslikano vrednostjo imena atributa (preslikava se vrsi, na
17 // primer s pomocjo algoritma SHA-1).
18 multicast_p = SHA-1(p);
19 multicast(multicast_p, zahteva_za_odkrivanje_vira);
```

4 Preizkus prototipa

V tem poglavju predstavimo rezultate prizkusa sistema DHT-RD, s katerim ocenimo njegove prednosti in slabosti. Preizkuse smo opravili tako v simuliranem omrežju (s pomočjo simulatorja sistema Pastry) kot tudi v realnem omrežju (v omrežju za testiranje PlanetLab).

4.1 Izračun porabe pomnilniškega prostora za shranjevanje kazalcev na vire

Največja slabost sistemov za odkrivanje virov, ki temeljijo na strukturiranih tehnologijah P2P (mednje spada tudi DHT-RD), je večja poraba pomnilnika, predvsem zaradi shranjevanja kazalcev na vire. Ker kazalci na vire niso enakomerno razporejeni po vozliščih, se pogosto dogaja, da je določena podmnožica vozlišč zelo obremenjena, medtem ko ostala vozlišča ne hranijo nobenega kazalca na vir. Sistem DHT-RD sicer vsebuje algoritme za čim enakomernejšo porazdelitev kazalcev po vozliščih, kljub temu je smiselno nekoliko podrobneje raziskati problem pomnilniške porabe pri koncentriranju kazalcev na majhni podmnožici vozlišč.

Vzemimo primer, ko uporabljamo sistem DHT-RD za porazdeljevanje podatkov o lokalnem računalniku, ki smo jih pridobili s pomočjo programa Ganglia [44] (slika 4.1).

```
<ns1:GLUECE xmlns:ns1="http://mds.globus.org/glue/ce/1.1">
<ns1:Host ns1:Name="berto.xlab" ns1:UniqueID="berto.xlab">
    <ns1:Processor ns1:CacheL1="0" ns1:CacheL1D="0"
        ns1:CacheL1I="0" ns1:CacheL2="0" ns1:ClockSpeed="1678" ns1:InstructionSet="x86"/>
    <ns1:MainMemory ns1:RAMAvailable="80" ns1:RAMSize="503"
        ns1:VirtualAvailable="239" ns1:VirtualSize="796"/>
    <ns1:OperatingSystem ns1:Name="Linux" ns1:Release="2.6.13-15.8-default"/>
    <ns1:ProcessorLoad ns1:Last15Min="27" ns1:Last1Min="31" ns1:Last5Min="23"/>
</ns1:Host>
</ns1:GLUECE>
```

Slika 4.1: Primer podatkov o lokalnem računalniku, ki ga dobimo s pomočjo program Ganglia.

Iz slike 4.1 je razvidno, da so podatki v programu Ganglia predstavljeni v obliki dokumenta XML. Povprečen dokument obsega 550 bajtov. Zajema 15 atributov: 5 se jih nanaša na splošne značilnosti CPE (CacheL1, CacheL1D, CacheL1I, CacheL2, ClockSpeed, InstructionSet), 4 na pomnilnik (RAMAvailable, RAMSize, VirtualAvailable, VirtualSize), 2 na operacijski sistem (Name, Release) ter 3 na obremenitev CPU (Last15Min, Last1Min, Last5Min). To pomeni, da moramo na podlagi zgornjega dokumenta 15-krat (po enkrat za vsako ime atributa) generirati kazalec na vir in ga objaviti v omrežju DHT. Ker se različna imena atributov preslikajo v različne identifikatorje, se navadno vsak izmed 15 kazalcev na vir shrani na ločenem vozlišču. S tem se breme porazdeljuje med več vozlišč, kar zmanjšuje nevarnost prevelike porabe pomnilnika pri posameznem vozlišču.

Pri kazalcih, katerih ciljni identifikator se generira iz istega imena atributa, pa se lahko zgodi, da se shranijo na isto vozlišče. Čeprav pri sistemu DHT-RD je verjetnost za to zelo majhna, predpostavimo, da se tak primer zgodi. Pomnilniška poraba na obremenjenem vozlišču je prikazana v tabeli 4.1. Glede na trenutno velikost največjih sistemov Grid (npr. sistem Grid, ki so ga postavili partnerji v okviru raziskovalnega projekta EGEE (`http://www.eu-egee.org/`), je sestavljen iz 80000 procesorskih jeder), se vsi njihovi računski viri brez težav shranijo v pomnilnik enega računalnika. Poleg tega se količina pomnilnika v računalnikih venomer povečuje, tako da bo v prihodnosti ta problem še manjši.

V primeru, da bi bile kapacitete pomnilnikov v določenih primerih dejansko premajhne za shranjevanje kazalcev na vire, bi le-te še vedno lahko hranili v podatkovnih bazah. V teh primerih bi bilo iskanje ustreznih virov počasnejše, vendar še vedno dovolj hitro za uporabo v sistemih Grid.

Tabela 4.1: Primer porabe pomnilniškega prostora pri različnem številu objavljenih virov. Predpostavimo, da je velikost kazalca na vir enaka velikosti XML dokumenta, ki ga generira Ganglija (torej 550B).

Število objavljenih računskih virov	**Pomnilniška poraba**
10	5,5kB
100	55kB
1000	550kB
10000	5,5MB
100000	55MB
1000000	550MB

4.2 Poskusi s pomočjo simulatorja

Sistemi Grid, v katerih se uporablja storitev za odkrivanje virov, obsegajo veliko število vozlišč.

Kot smo že omenili v predhodnem poglavju, je sistem Grid, ki so ga sestavili v okviru projekta EGEE, sestavljen iz 80000 procesorskih jeder. Le-ta se nahajajo na 300 lokacijah, v 50 državah širom po svetu. Če želimo preveriti, kako se bo naš sistem za odkrivanje virov obnašal v sistemih Grid, ga moramo preizkusiti na omrežjih z velikim številom računalnikov (1000 in več). Žal so taka testna omrežja zelo redka (eno izmed njih je PlanetLab, ki ga bomo opisali v razdelku 4.3), poleg tega je z njimi zelo težko upravljati.

Zato poskusov, ki vključujejo veliko število vozlišč, navadno ne izvajamo v realnih omrežjih, ampak s pomočjo simulatorjev. To so programi, ki omogočajo izvajanje celotnega poskusa na enem samem računalniku, pri tem pa simulirajo mrežne povezave. Uporabnik lahko s pomočjo nastavitvenih datotek določi povezanost vozlišč, zakasnitve povezav in druge parametre. Prednost uporabe simulatorjev pri preizkušanju porazdeljenih sistemov je preprostejše izvajanje poskusov (ni nam treba skrbeti za rezervacijo testnih računalnikov, namestitev potrebnih orodij) ter krajši čas trajanja posameznega poskusa (ker uporabljamo simulirane mrežne povezave in se celoten poskus izvaja na enem samem računalniku, je izmenjevanje sporočil med vozlišči precej hitrejše kot pri resničnih povezavah).

Pri preizkušanju sistema DHT-RD smo uporabili kar simulator, ki je vgrajen v ogrodje Pastry. Ta omogoča načrtovanje simulacij na omrežjih z 10000 do 100000 vozlišči, kar je več kot jih vsebuje katerikoli sistem Grid. Vgrajeni simulator Pastry spada med simulatorje na podlagi diskretnih dogodkov [45]. To pomeni, da so vse operacije v sistemu predstavljene kot zaporedje dogodkov. Vsak dogodek se zgodi ob točno določenem trenutku in povzroči spremembo stanja sistema. Na primer: če bi simulirali dvigalo v stolpnici, bi bil eden izmed možnih dogodkov "uporabnik je pritisnil gumb za 5. nadstropje". Ta dogodek bi povzročil spremembo stanja v "dvigalo se premika" in sčasoma v "dvigalo je v 5. nadstropju".

V simulacijskem načinu ogrodje Pastry simulira fizično omrežje ter vozlišča v enem samem Javanskem procesu. Glavna nit vzpostavi omrežje in zažene vozlišča. Le-ta zatem komunicirajo med seboj na podlagi deljenega seznama, v katerega vstavljajo svoja sporočila. Za dostavljanje sporočil ciljnim vozliščem skrbi poseben razvrščevalnik. Ta na vsakem koraku izbere sporočilo iz deljenega seznama, in ga preda ciljnemu vozlišču. Na ta način simulira fizično omrežje.

Značilnosti vgrajenega simulatorja Pastry:

- izvajanje aplikacij Pastry na simuliranem omrežju z minimalnimi popravki izvorne kode,

- hitrejše izvajanje kot pri omrežjih v realnem času: pri simulacijah uporabljamo navidezno uro, ki teče hitreje od sistemske ure. Po drugi strani obremenjenost računalnika, na katerem izvajamo simulacijo (ta je odvisna od velikosti omrežja, ki ga simuliramo) ne vpliva na točnost rezultatov, saj navidezna ura

napreduje samo takrat, ko v danem trenutku ni več nobenega pravila za izvajanje,

- po potrebi lahko pri simulacijah namesto navidezne ure uporabimo realno uro. To nam omogoča testiranje interaktivnih aplikacij na simuliranem omrežju.

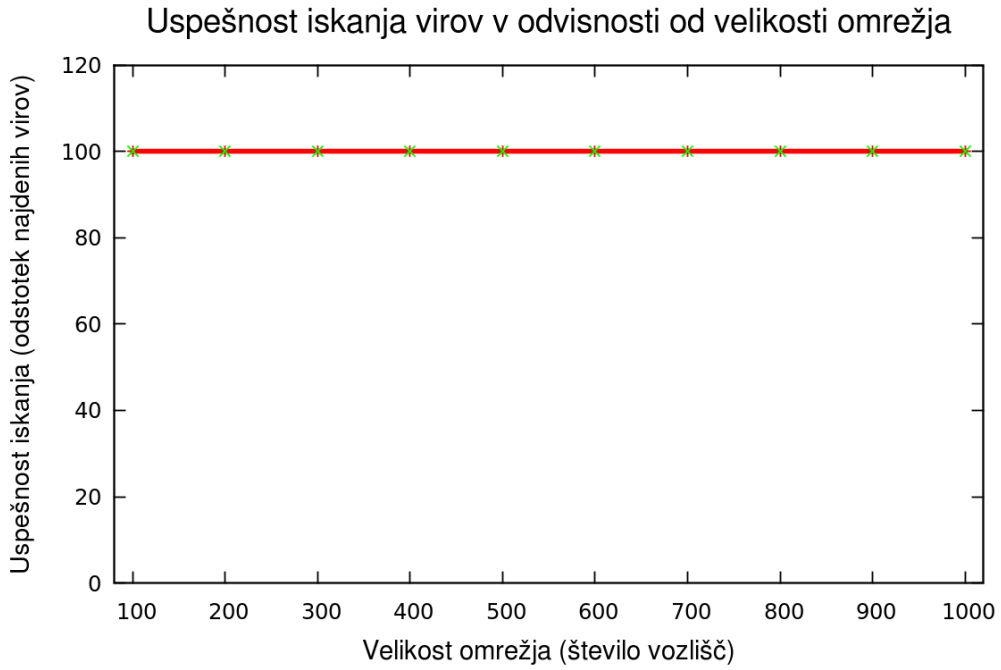
- možnost določanja zakasnitev mrežnih povezav: simulator nudi 2 modele za določanje oddaljenosti med vozlišči Evklidskega (ravninskega) in sferičnega. Pri njih se vozlišča razporedijo naključno po prostoru, zakasnitve po povezavah pa se določijo na podlagi oddaljenosti med vozlišči. Uporabnik lahko zakasnitve po povezavah določi tudi sam, s tem da poda simulatorju pot do datoteke, v kateri je shranjena matriko zakasnitev. S tem lahko izvaja poskuse na poljubnih mrežnih topologijah.

4.2.1 Poskus 1: uspešnost odkrivanja virov v odvisnosti od velikosti omrežja

Cilj tega poskusa je ugotoviti, ali velikost omrežja vpliva na uspešnost odkrivanja virov. Različno število vozlišč najprej povežemo v omrežje DHT (identifikatorji vozlišč se generirajo naključno, kar nam zagotavlja testiranje na različnih topologijah prekrivnega omrežja). Na vsakem vozlišču generiramo računski vir, ki vsebuje 5 atributov: procesorska arhitektura (i386, PowerPC, MIPS, Alpha, ...), frekvenca CPU, količina celotnega pomnilnika, količina prostega pomnilnika ter vrsta operacijskega sistema (Windows, Linux, MacOS, ...). Vrednosti za vsak atribut izberemo naključno. Iz vsakega vozlišča nato objavimo kazalec na lokalen računski vir. Kazalci se usmerjajo po omrežju DHT, dokler ne prispejo do korenskega vozlišča.

Objavljanju sledi odkrivanje računskih virov. Iz vsakega vozlišča pošljemo enako poizvedbo, ki je sestavljena iz atributov računskega vira. Pred tem je treba še ugotoviti, koliko objavljenih virov dejansko ustreza izbrani poizvedbi. Na ta način ugotovimo pričakovano število virov. Pričakovano število virov določamo za vsak poskus posebej, saj se računski viri med posameznimi poskusi razlikujejo (vrednosti atributov za vsak vir se generirajo naključno).

Rezultati poskusa so prikazani s pomočjo grafa 4.2. Poskus smo ponovili 10-krat, da bi izničili vpliv posamezne topologije prekrivnega omrežja. Iz grafa je razvidno, da sistem DHT-RD dejansko odkrije vse vire, ki ustrezajo dani poizvedbi, ne glede na to, kakšna je velikost omrežja. Poglaviten vzrok za tako učinkovito odkrivanje virov je uporaba DHT sistema Pastry za usmerjanje sporočil do ciljnih vozlišč. Kot smo že omenili v razdelku 4.2, je značilnost sistemov za odkrivanje virov, ki temeljijo na strukturiranih tehnologijah P2P, prav zagotovljeno odkrivanje vseh virov, ki ustrezajo iskalnim poizvedbam.



Slika 4.2: Graf uspešnosti iskanja virov v odvisnosti od velikosti omrežja.

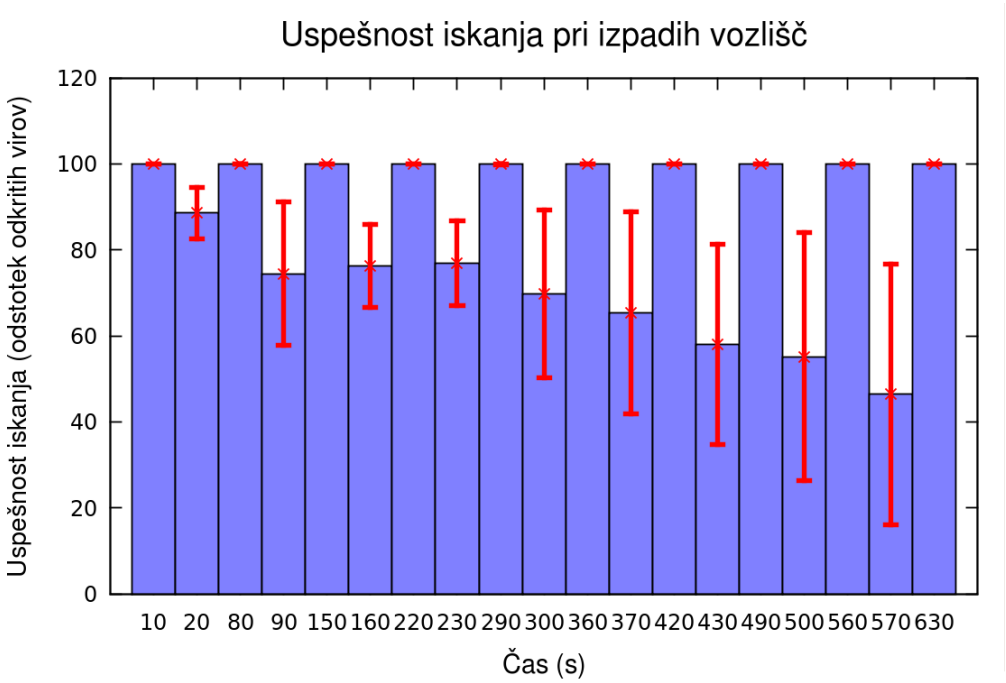
4.2.2 Poskus 2: uspešnost odkrivanja virov pri odpovedih vozlišč

Ta poskus je podoben tistemu iz predhodnega razdelka, le da smo vanj vključili še izpade vozlišč. Cilj poskusa je ugotoviti, ali periodično objavljanje kazalcev na vire učinkovito odpravlja problem odkrivanja virov v nestabilnih omrežjih, torej pri omrežjih, pri katerih so odhodi vozlišč zelo pogosti. Poskus izvajamo na omrežju s 500 vozlišči. Po izgradnji omrežja DHT (tudi pri tem poskusu se identifikatorji vozlišč generirajo naključno) pri vsakem vozlišču generiramo računski vir z naključnimi vrednostmi atributov in ga objavimo. Računski vir je sestavljen iz enakih atributov kot tisti iz razdelka 4.2.1. Po objavi počakamo 10 sekund, da se kazalci na vire zagotovo razporedijo na ustrezna vozlišč in začnemo s prvim odkrivanjem. Postopek odkrivanja je enak kot pri predhodnem poskusu: najprej generiramo naključno poizvedbo, nato pa iz vsakega vozlišča pošljemo zahtevo za iskanje virov ki ustrezajo njenim pogojem.

10 sekund po prvem odkrivanju simuliramo odpoved 50 vozlišč, kar predstavlja 10% celotnega omrežja. Neposredno za odpovedmi sprožimo iskanje virov, s čimer ugotovimo delež virov, ki ga ne uspemo najti, bodisi zaradi napak v omrežju DHT bodisi zaradi izgubljenih kazalcev na vire (pri izpadu vozlišč postanejo vsi kazalci na vire, ki so shranjeni na njih, nedostopni). Zatem počakamo 60 sekund, da se osvežijo kazalci na vire. Le-ti se shranijo na tistih vozliščih, ki so še prisotna v omrežju. Po 60

sekundah zopet sprožimo iskanje. Na ta način preverimo, kako učinkovito odpravlja periodično obnavljanje problem izpadov vozlišč.

Sledi druga iteracija simulacije odpovedi, ki je enaka prvi: tudi tu najprej simuliramo odpoved 50 vozlišč, počakamo 60 sekund in izvedemo odkrivanje. Vseh iteracij odpovedi je 9, tako da na koncu ostane v omrežjo samo še 10% vozlišč iz prvotne množice.



Slika 4.3: Prikaz uspešnosti iskanja virov pri izpadih vozlišč.

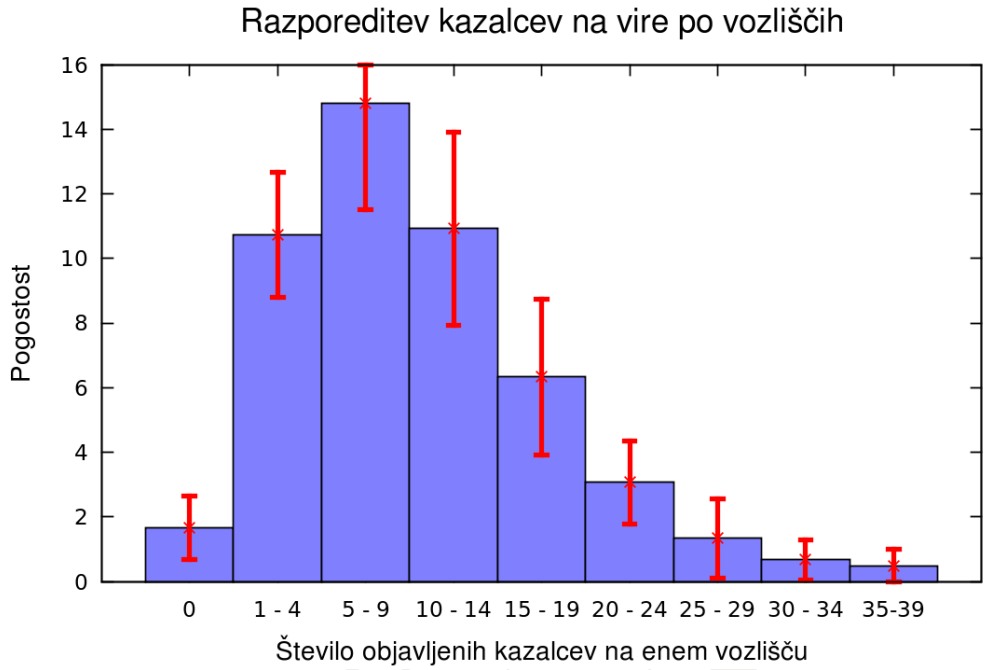
Rezultati poskusa so prikazani na sliki 4.3. Na grafu so prikazane povprečne vrednosti za serijo več poskusov, ki smo jih izvedli, da raziščemo splošno obnašanje algoritma. Iz rezultatov je razvidno, da periodično obnavljanje virov zelo učinkovito rešuje problem izpadov vozlišč: 60 sekund po vsakem izpadu nam je namreč uspelo odkriti vse vire v omrežju, ki ustrezajo dani poizvedbi (glej rezultate za čas 80, 150, 220, 290, ...). Vidimo lahko tudi, da so izpadi vozlišč precej bolj problematični v manjših omrežjih (pri času 570, ko omrežje pred izpadom obsega samo 100 vozlišč, uspemo odkriti po izpadu samo še polovico vseh virov), medtem ko v večjih omrežjih nimajo tolikšnega vpliva. Tudi ta rezultat je v skladu z našimi predvidevanji.

4.2.3 Poskus 3: porazdelitev kazalcev po vozliščih

Porazdeljevanje kazalcev po vozliščih je ključnega pomena za učinkovitost sistema DHT-RD. Če so kazalci na vire skoncentrirani samo na majhnem številu vozlišč,

so ta vozlišča preobremenjena in odkrivanje virov je počasnejše. Zaželeno je, da so kazalci na vire porazdeljeni čim bolj enakomerno med vozlišča v omrežju. Cilj tega poskusa je bil ugotoviti, koliko sistem DHT-RD odstopa od želene porazdelitve.

Poskus smo izvedli na omrežju s 500 vozlišči. Ker sistem Pastry uporablja 160-bitne identifikatorje (to pomeni, da podpira omrežja z do 2^{160} vozlišči), bi morala biti testna omrežja zelo velika, da bi vsebovala zadostno število vozlišč (vsaj 2^{120}) z dovolj podobnimi identifikatorji, na katerih bi lahko preizkušali razporeditev. Ker tako velikih omrežij ne moremo ustvariti niti v simulatorju, smo se odločili, da 10% (t.j. 50) vozliščem ne bomo generirali identifikatorjev naključno, ampak jim bomo izbrali skupno predpono. Le-ta bo obsegala 128 bitov, medtem ko bomo pripono (preostalih 32 bitov) še vedno generirali naključno. Gradnja omrežja poteka na enak način kot pri predhodnih poskusih. Prav tako objavljanje: na vsakega vozlišča generiramo ločen računski vir za naključnimi vrednostmi atributov. Kazalec na ta vir objavimo v prekrivnem omrežju sistema DHT-RD.



Slika 4.4: Prikaz porazdelitve kazalcev na vire po vozliščih.

Rezultati poskusa so prikazani na sliki 4.4. Zaradi boljše preglednosti ne prikazujemo pogostosti za vsako število kazalcev posebej, temveč smo jih združili v intervale dolžine 5. Pri tem smo se odločili, da bomo 0 prikazali ločeno od prvega intervala, saj lahko na ta način opazujemo pogostost vozlišč, na katerih ni shranjenega nobenega kazalca na vir. Zaželeno je, da bi bilo takih vozlišč čim manj, saj je

v tem primeru porazdelitev bremena bolj enakomerna.

Iz grafa je razvidno, da večina izmed 50 vozlišč s skupno predpono hrani v povprečju 1 do 14 kazalcev na vir. Ta vozlišča predstavljajo 73% vseh vozlišč, na katerih je shranjen kateri od kazalcev na vire. Glede na to, da bi pri idealni razporeditvi 500 kazalcev na vire med 50 vozlišč vsako vozlišče hranilo 10 vozlišč, lahko sklepamo, da je porazdelitev bremena v sistemu DHT-RD zelo ugodna: zelo malo je preobremenjenih vozlišč ter vozlišč z ustrezno predpono, ki ne hranijo nobenega kazalca na vir. Naključno generiranje pripon se tako izkaže za dokaj učinkovit način porazdeljevanja kazalcev po vozliščih.

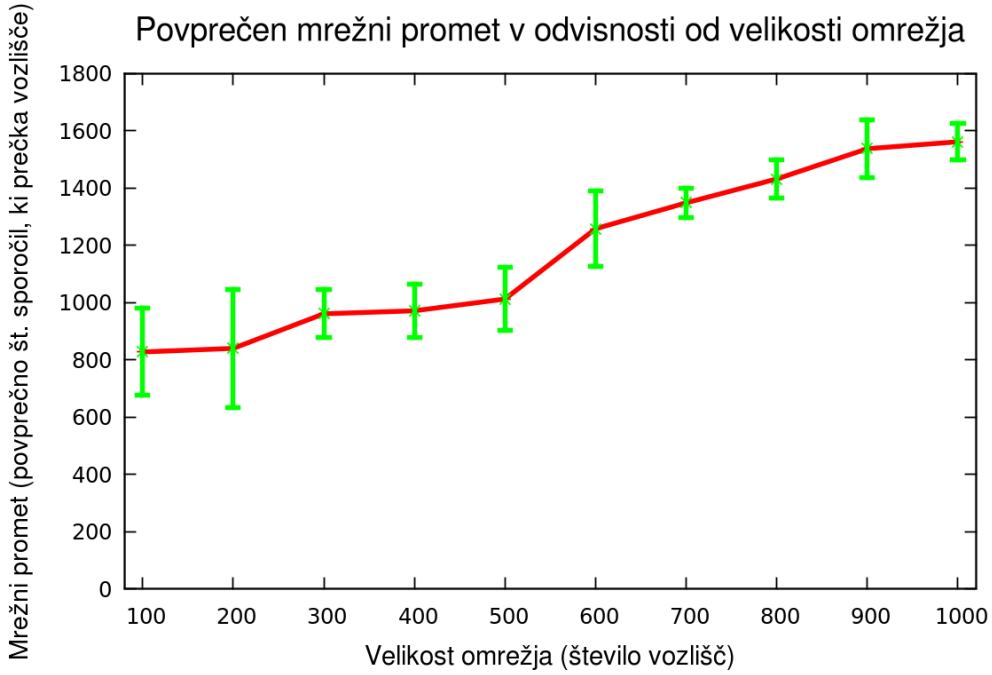
4.2.4 Poskus 4: količina mrežnega prometa v odvisnosti od velikosti omrežja

Ena izmed ključnih prednosti sistemov, ki temeljijo na tehnologijah P2P, je razširljivost. S pomočjo tega poskusa smo želeli preveriti, ali to drži tudi za sistem DHT-RD. Cilj poskusa je oceniti, kako velikost prekrivnega omrežja vpliva na količino prometa. Sistemi, v katerih mrežni promet narašča prehitro, imajo zelo slabo razširljivost, saj hitro pride do preobremenitve vozlišč.

Poskus poteka na podoben način kot tisti, ki smo ga opisali v razdelku 4.2.1. Najprej generiramo vozlišča z naključnimi identifikatorji in jih povežemo v omrežje P2P. Sledi generiranje in objavljanje računskih virov. Kot pri ostalih poskusih tudi tu generiramo pri vsakem vozlišču ločen računski vir. Po končanem objavljanju kazalcev na vire se lotimo še njihovega odkrivanja. Le-to izvedemo iz vsakega vozlišča.

V času trajanja celotnega poskusa beležimo vsa sporočila, ki prečkajo katero izmed vozlišč. Kot mero za obremenitev vozlišč (in posledično za mrežni promet) smo si namreč izbrali povprečno število sporočil, ki prečkajo vozlišče sistema DHT-RD. Upoštevali smo tako sporočila za vzdrževanje omrežja Pastry (izmenjava usmerjevalnih tabel, preverjanje prisotnosti vozlišč,...), kot tudi sporočila, ki se tvorijo pri objavljanju in odkrivanju virov. Meritve smo izvedli za omrežja različnih velikosti, od 100 do 1000 vozlišč.

Rezultati poskusa so prikazani na sliki 4.5. Iz grafa je razvidno, da rast obremenitve vozlišč zaradi mrežnega prometa ni pretirana, saj se povprečno število sporočil, ki prečka posamezno vozlišče, pri desetkrat večjih omrežjih samo podvoji. Počasnejša rast prometa je posledica uporabe strukture DHT. Zanjo je namreč značilno, da pri iskanju ne uporablja razprševanja, ki generira veliko prometa, ampak se sporočilo s pomočjo usmerjevalnih tabel usmerja neposredno k ciljnemu vozlišču. To usmerjanje je zelo učinkovito, saj poljubno sporočilo na poti do cilja prečka največ $O(logN)$ vozlišč, kjer je N velikost omrežja [46].



Slika 4.5: Prikaz povprečnega prometa po vozliščih v odvisnosti od velikosti omrežja.

4.3 Poskusi v omrežju PlanetLab

Seveda poskusi v simulatorju ne morejo popolnoma izpodriniti poskusov v realnem omrežju. Določene meritve se sicer enake, ne glede na to, ali jih merimo s pomočjo simulatorja ali v realnem omrežju (take so, na primer, meritve, ki smo jih opisali v razdelku 4.2), veliko pa takih, pri katerih bi v simulatorju dobili samo grob približek realnih vrednosti. Primer take meritve je zakasnitev med pošiljanjem iskalne zahteve in prejemom rezultatov. Pri teh meritvah dobimo natančnejše rezultate, če jih izvajamo v realnih omrežjih. Eno izmed globalnih omrežij, ki služi izključno za izvajanje poskusov na področju porazdeljenih sistemov, je PlanetLab.

Testno omrežje PlaneLab (`http://www.planet-lab.org`) so leta 2002 ustanovili predstavniki različnih akademskih, gospodarskih in vladnih ustanov. Omrežje postaja vedno bolj prepoznavno med razvijalci porazdeljenih sistemov in se nezadržno širi. Leta 2006 je obsegal 694 računalnikov na 336 različnih lokacijah po vsem svetu, danes (leta 2009) pa zajem že 1006 računalnikov 487 lokacijah.

Vsak raziskovalni projekt ima v okviru sistema PlanetLab določeno t.i. rezino (angl. slice), ki predstavlja množico računalnikov, do katerih lahko dostopa. Uporaba sistema je preprosta: uporabnik se na oddaljeni računalnik poveže s pomočjo programa SSH (Secure SHell), kateremu poda privatni RSA ključ, ki ga je registriral

s pomočjo spletnega portala PlanetLab. Če je prijava uspela, lahko na oddaljenemu računalniku izvaja poljubne programe.

Posamezni računalniki se med seboj razlikujejo tako v zmogljivosti strojne opreme kot tudi v hitrosti mrežnih povezav, vsi pa imajo nameščen operacijski sistem Linux. Tako lahko uporabnik testira porazdeljene aplikacije v realnih pogojih, saj so v omrežju PlanetLab posamezni računalniki oddaljeni med seboj več tisoč kilometrov, poleg tega pa se na njih izvajajo tudi drugi programi, kar ima za posledico realnejšo obremenitev sistema.

Seveda ima omrežje PlanetLab tudi slabosti. Največja izmed njih je dokaj velik delež nedosegljivih vozlišč. Izmed 1006 računalnikov, ki jih zajema PlanetLab, jih je okoli 300 (t.j. 30%) nedosegljivih na daljši rok, bodisi zaradi napake v strojni opremi, bodisi zaradi okvare omrežja. Ta nedosegljivost močno zmanjša nabor računalnikov, ki so na voljo za testiranje. Problematična je tudi preobremenjenost velikega števila računalnikov. Ti računalniki so sicer dosegljivi za uporabnike, vendar se na njih izvaja veliko programov. Njihova odzivnost je zato zelo slaba, kar pomeni, da so praktično neuporabni za izvajanje poskusov.

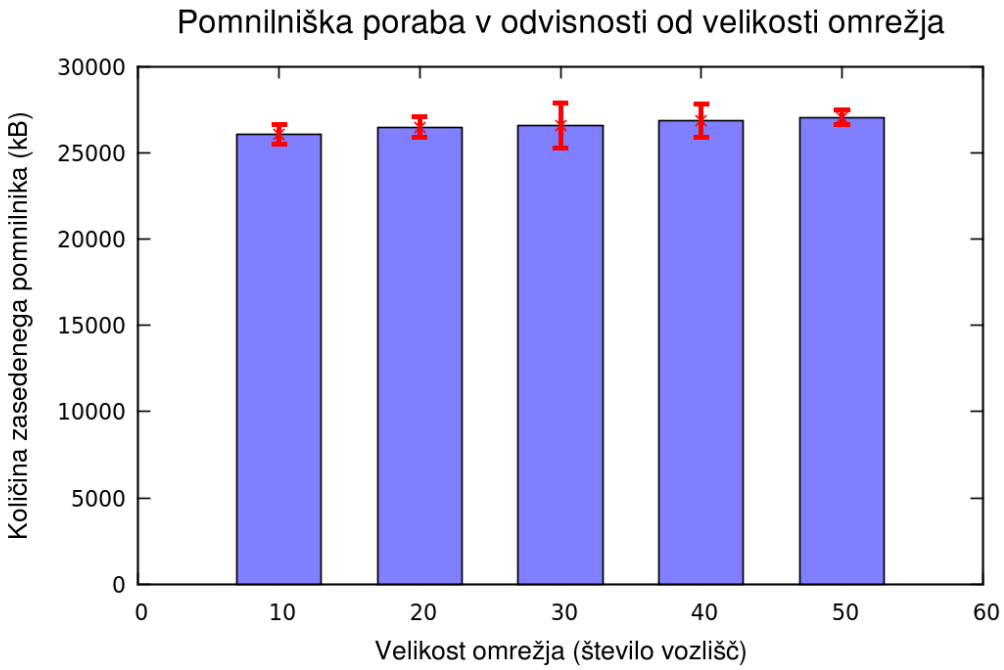
Še ena izmed slabosti so pogosti izpadi vozlišč. Zaradi njih je treba poskuse ponavljati, kar dodatno podaljšuje čas testiranja.

4.3.1 Poskus 1: pomnilniška poraba v odvisnosti od velikosti omrežja

V razdelku 4.1 smo že omenili, da je največja slabost sistema DHT-RD nekoliko večja poraba pomnilnika zaradi shranjevanja kazalcev na vire. S pomočjo tega poskusa bi radi ugotovili kakšna je ta poraba in ali nam lahko dela težave pri večjih omrežjih.

Velikost testnih omrežij PlanetLab je zaradi slabosti, ki smo jih omenili v predhodnem razdelku, je veliko manjša od omrežij v simulatorju. Testiranja v omrežjih, ki zajemajo več kot 50 vozlišč, so zelo dolgotrajna (predvsem zaradi dolgotrajne priprave računalnika PlanetLab), poleg tega jih je treba pogosto ponavljati, zaradi nenadnih izpadov vozlišč. Po drugi strani je za testiranje porabe pomnilnika zaželeno imeti čim večja omrežja, saj je v takih omrežjih število kazalcev na vire večje. Da bi tudi pri manjših omrežjih generirali enako breme, kot ga imamo pri večjih omrežjih, smo pri tem poskusu na vsakem vozlišču generirali in objavili 100 različnih računskih virov (ne samo 1, kot pri prejšnjih poskusih). Meritve smo izvajali na 10 - 50 vozliščih, kar pomeni, da smo skupno zgenerirali 1000 - 5000 računskih virov. Po končanem postopku objavljanja smo izmerili porabo fizičnega pomnilnika za proces DHT-RD na vsakem vozlišču PlanetLab.

Rezultati poskusa so prikazani na sliki 4.6. Na grafu je prikazana pomnilniška zasedenost najbolj obremenjenega vozlišča pri posameznem poskusu. Vanjo je poleg objavljenih kazalcev vključena celotna izvajalna koda programa DHT-RD, skupaj s potrebnimi knjižnicami. Iz spreminjanja zasedenosti pri različnih velikostih omrežja je razvidno, da izvajalna koda programa predstavlja večino zasedenega po-



Slika 4.6: Prikaz pomnilniške porabe programa DHT-RD na posameznem vozlišču v odvisnosti od velikosti omrežja.

mnilniškega prostora, na kazalce na vire odpade le manjši del. Na podlagi razlike v količini zasedenega pomnilnika za omrežje z 10 vozlišči in omrežje s 50 vozlišči, lahko ocenimo, da vsak kazalec na vir zasede okoli 300 bajtov (glej enačbo 4.1). Ob enakih pogojih bi potemtakem v omrežju s 1000 vozlišči kazalci na računske vire zasedli 250kB, v omrežju z 10000 vozlišči pa 2,5MB. Glede na to, da ima dandanes vsak povprečen računalnik že najmanj 1GB pomnilnika, 2,5MB ne predstavlja resnejšega problema za razširljivost sistema DHT-RD.

$$\frac{27100 - 26100}{50 * 100 - 10 * 100} = \frac{1000}{4000} = 0,25kB \tag{4.1}$$

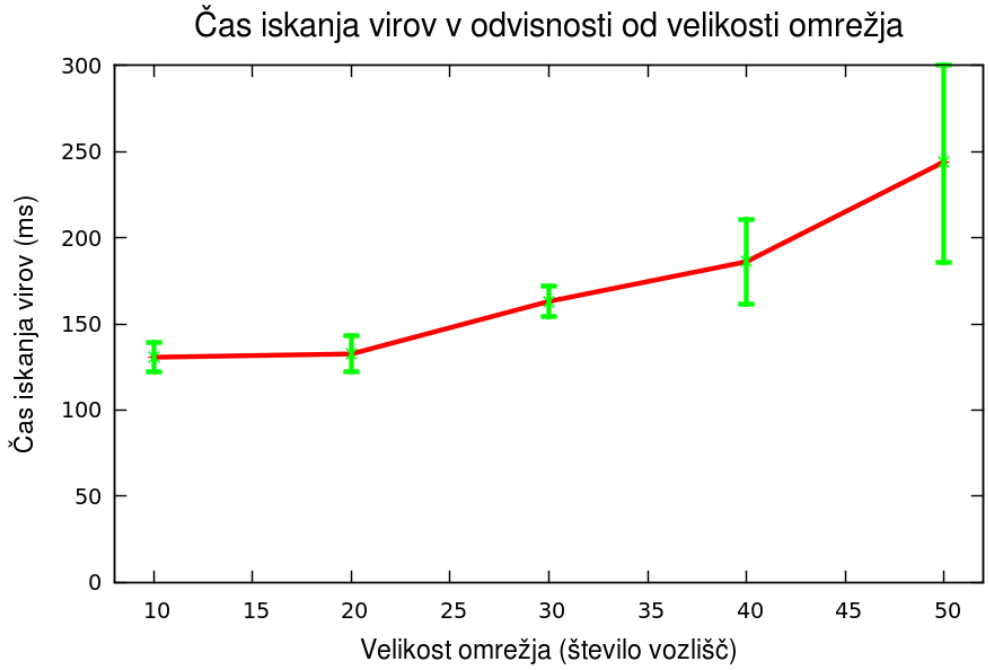
4.3.2 Poskus 2: zakasnitev pri iskanju v odvisnosti od velikosti omrežja

Cilj tega poskusa je bil izmeriti povprečen čas iskanja virov pri različnih velikostih omrežij. Za izvedbo v omrežju PlanetLab smo se odločili, ker smo želeli čase iskanja želeli izmeriti v realnem omrežju. S pomočjo simulatorja bi veliko težje dobili rezultate, ki bi bili primerljivi z realnimi.

Postopek je podoben kot pri ostalih poskusih, le da se izvaja porazdeljeno (vsako vozlišče se izvaja na ločenem računalniku): najprej se vozlišča povežejo v omrežje. Na tem mestu velja omeniti, da se identifikatorji vozlišč ne generirajo popolnoma naključno, ampak si delijo skupno predpono (postopek generiranja je enak kot pri testiranju porazdelitve kazalcev na vire v razdelku 4.2.1). Če bi identifikatorje genrirali naključno, bi si bili preveč različni. Zaradi majhnega števila testnih vozlišč bi to pomenilo, da ima vsako vozlišče v svoji usmerjevalni tabeli kazalce na vsa ostala testna vozlišča. Iskalnih zahtev tako ne bi bilo treba usmerjati po omrežju DHT ampak bi se pošiljale neposredno k ciljnim vozliščem. Čas iskanja virov se ne bi bistveno razlikoval pri različnih velikostih testnega omrežja.

Gradnji omrežja sledi objavljanje kazalcev, pri čemer iz vsakega vozlišča objavimo po en kazalec na računski vir, podobno kot pri predhodnih poskusih. Kazalci se razporedijo naključno po vozliščih. Zatem začnemo z odkrivanjem računskih virov. Iz vsakega vozlišča pošljemo enako poizvedbo s pogoji, ki jim morajo zadoščati viri. Pri tem merimo čas, ki preteče od pošiljanja iskalne zahteve do prejetja zadnjega kazalca na vir, ki ustreza iskalni poizvedbi.

Rezultati poskusa so prikazani na sliki 4.7. Iz grafa je razvidno, da se zakasnitev pri iskanju povečuje z velikostjo omrežja, kar je v skladu z našimi pričakovanji. Poleg tega so iskalni časi v skladu z omrežnimi zakasnitvami (angl. networ delay) v omrežju PlanetLab: povprečna mrežna zakasnitev, ki jo izmerimo s pomočjo ukaza "ping", se giblje med 70 in 120 milisekund. Na tem mestu pa velja omeniti, da iskalni časi včasih zelo odstopajo od tega povprečja: pri nekaterih vozliščih smo pri določenih poskusih namerili povprečen iskalni čas 12 sekund! Razlog za te odklone je velika obremenjenost omrežja PlanetLab. Na posameznih računalnikih se sočasno



Slika 4.7: Prikaz povprečnega iskalnega časa za eno poizvedbo v odvisnosti od velikosti omrežja.

izvaja več navideznih strojev (angl. virtual machine), po eden za vsako rezino, v katero je vključen dani računalnik (vsaka rezina pripada določeni organizaciji, ki je vključena v omrežje PlanetLab). Operacijski sistem na računalniku skrbi za dodeljevanje sistemskih virov (procesor, mrežna povezava,...) posameznim rezinam. Vsaka rezina dobi računalnik v uporabo samo v določenem časovnem intervalu. Po preteku tega intervala mora počakati, da pride zopet na vrsto. V tem času ne more sprejemati in obdelovati sporočil z oddaljenih vozlišč. Na računalnikih, ki gostijo veliko navideznih strojev, se čakanje na procesorski čas ustrezno podaljša. Poleg tega dodeljevanje procesorskega časa po vozliščih ni sinhronizirano: ista rezina se na različnih računalnikih lahko izvaja ob različnih trenutkih. Ob takih primerih pošiljanje ni mogoče, zato moramo počakati toliko časa, dokler se izvajanje na obeh vozliščih ne prekrije. To sta glavna razloga za velika odstopanja v določenih primerih.

5 Zaključek

V tem poglavju si ogledamo zadanih izvirnih ciljev in prispevke tega dela k področju odkrivanja virov v sistemih Grid. Zatem podamo še smernice za bodoče delo na obdelanem področju.

5.1 Prispevki dela

Pričujoče delo definira algoritme za odkrivanje virov s pomočjo strukturiranih tehnologij "enak z enakim" ter opiše porazdeljeni sistem DHT-RD, ki je prototipna izvedba algoritmov. Sisteme Grid, ki so problemsko področje magistrskega dela, smo opisali v razdelku 1.1, problematiko odkrivanja virov pa smo definirali v razdelku 1.2.

Celotno poglavje 2 je posvečeno predstavitvi obstoječih sistemov za odkrivanje virov ter oceni njihovih prednosti in slabosti. Sistemi so v grobem razdeljeni v 3 kategorije, na podlagi strukture njihovega omrežja: sistemi za odkrivanje virov na osnovi hierarhične strukture (sem spada GT4 MDS), sistemi za odkrivanje virov na osnovi nestrukturiranih P2P tehnologij ter sistemi za odkrivanje virov na osnovi strukturiranih P2P tehnologij. V zadnjem razdelku (2.4) smo primerjali med seboj predstavnike vseh treh kategorij. Opazili smo, da so omrežja P2P bolj primerna za gradnjo sistemov za odkrivanje virov od hierarhičnih omrežij, predvsem zaradi boljše razširljivosti.

Precej bolj enakovredni so si sistemi v kategorijah nestrukturirani P2P in strukturirani P2P. Glavna prednost uporabe nestrukturiranih sistemov P2P pri odkrivanju virov je preprostost implementacije in nezahtevnost vzdrževanja omrežja P2P: pri teh sistemih ni potrebno periodično preverjati, ali se določeno vozlišče še nahaja v omrežju, prav tako ni treba vzdrževati usmerjevalnih tabel (tabel, ki služijo za usmerjevanje sporočil v omrežju P2P na podlagi njihovega ključa). Slabost nestrukturiranih sistemov P2P pa je neučinkovitost algoritmov za iskanje virov. Veliko sistemov uporablja za razširjanje poizvedb po omrežju kar poplavljanje, ki generira veliko mrežnega prometa. Zato je njegova razširljivost zelo slaba. Obstaja sicer veliko mehanizmov, ki uspešno omejujejo mrežni promet zaradi poplavljanja, vsi pa imajo skupno slabost: pri nobenem ni zagotovljeno, da bo uporabnik dobil vse vire

v omrežju, ki ustrezajo dani poizedbi.

Zaradi zgoraj omenjenih slabosti nestrukturiranih sistemov P2P smo se odločili zasnovati naš sistem za odkrivanje virov na enem od strukturiranih sistemov P2P. Uporabili smo implementacijo porazdeljene zgoščene tabele Pastry, katero smo podrobneje opisali v razdelku 3.2. Naša algoritma za objavljanje in odkrivanje virov sta opisana v razdelku 3.3. Ta algoritma omogočata uporabo večparametrskih območnih poizvedb, ob tem pa ne generirata toliko mrežnega prometa, kot obstoječe izvedbe sistemov za odkrivanje virov na osnovi strukturiranih P2P tehnologij.

Prednost teh algoritmov je tudi učinkovitejše indeksiranje dinamičnih atributov (atributov, katerih vrednosti se pogosto spreminjajo). Največji problem pri obstoječih strukturiranih P2P sistemih za odkrivanje virov je zahteva po ponovnem indeksiranju virov, vsakič ko se jim spremeni katera od vrednosti atributov. To lahko precej obremeni omrežje, še posebej, ko so spremembe vrednosti atributov zelo pogoste.

Algoritme za odkrivanje virov smo tudi preizkusili, tako s pomočjo simulatorja omrežij P2P, kot tudi v realnem omrežju. Rezultati poskusov so podani v poglavju 4. Iz njih je razvidno, da s pomočjo sistema DHT-RD lahko odkrijemo vse vire, ki ustrezajo dani poizvedbi, ne glede na velikost omrežja. Poleg tega sistem zaradi periodičnega obnavljanja omogoča učinkovito iskanje tudi ob izpadih vozlišč. Poskusi so pokazali tudi, da naključno generiranje pripon ciljnih identifikatorjev učinkovito porazdeljuje kazalce na vire med vozlišča v omrežju. To omogoča enakomernejšo obremenitev vozlišč.

Povprečna količina mrežnega prometa po vozliščih sicer narašča z velikostjo omrežja, vendar pa je ta rast počasna. Isto velja za pomnilniško porabo ter zakasnitev pri iskanju. Zaradi teh lastnosti lahko uporabljamo DHT-RD tudi v večjih sistemih Grid.

5.2 Bodoče delo

Trenutno je razmerje med dolžino predpone, ki se generira iz imena atributa, in pripono, ki jo generiramo naključno, fiksno in vnaprej določeno. Daljša predpona pomeni večjo porazdelitev kazalcev na vire med vozlišča, vendar tudi večje število sporočil multicast pri iskanju. Idealno razmerje med dolžinama se pri različnih vrstah virov razlikuje. V bodoče bi bilo zaželeno, da bi ga uporabnik podal sam pri objavljanju in odkrivanju virov. To bi razbremenilo omrežje in izboljšalo hitrost iskanja. Veljalo bi opraviti tudi poskuse, kakšno razmerje v splošnem omogoča najugodnejšo porazdelitev kazalcev med vozlišča.

Pri trenutni implementaciji sistema DHT-RD se pripone ciljnih identifikatorjev generirajo naključno. Ta način nam omogoča dokaj enakomerno porazdelitev kazalcev po vozliščih, ne omogoča pa izkoriščanje lokalnosti: sistem ne zagotavlja, da

bodo viri, ki se nahajajo v istem omrežju, objavili na istem vozlišču. To bi lahko omogočili, če bi pripono identifikatorja generirali iz dela naslova IP vozlišča, na katerem se nahaja vir. Ta metoda se imenuje gručenje na podlagi IP (angl. IP-based clustering) in je podrobneje opisano v [47].

Porazdeljevanje kazalcev med vozlišča v sistemu DHT-RD bi lahko še dodatno izboljšal, če bi implementiral avtomatsko seljenje kazalcev v primeru, ko bi število kazalcev na kateremu izmed vozlišč preseglo določen prag. Takrat bi preobremenjeno vozlišče kontaktiralo eno izmed neobremenjenih vozlišč in ga prosilo za odobritev prenosa. Ob odobritvi bi vozlišče spremenilo svoj identidikator, da bi se ujemal s predpono identifikatorja, pod katerim so objavljeni kazalci. To je v sistemih DHT preprosto izvedljivo. Zatem bi se izvršil prenos dela kazalcev na novo vozlišče. Na ta način bi sistem DHT-RD avtomatsko uravnaval obremenjenost vozlišč.

Literatura

[1] Ian Foster and Carl Kesselman. *The Grid: Blueprint for a new computing infrastructure*. Morgan Kaufmann, November 1998.

[2] I. Foster, C. Kesselman, and S. Tuecke. The anatomy of the grid: Enabling scalable virtual organizations. *International Journal of High Performance Computing Applications*, 15(3):200, 2001.

[3] Koen Vanthournout, Geert Deconinck, and Ronnie Belmans. A taxonomy for resource discovery. *Personal and Ubiquitous Computing*, 9(2):81–89, March 2005.

[4] Stephanos Androutsellis-Theotokis and Diomidis Spinellis. A survey of peer-to-peer content distribution technologies. *ACM Computing Surveys*, 36(4):335–371, December 2004.

[5] Domenico Talia and Paolo Trunfio. Toward a synergy between p2p and grids. *IEEE Internet Computing*, 7(4):96–95, 2003.

[6] A. Iaminitchi and D. Talia. P2p computing and interaction with grids. *Future Gener. Comput. Syst.*, 21(3):331–332, 2005.

[7] Moreno Marzolla, Matteo Mordacchini, and Salvatore Orlando. Peer-to-peer systems for discovering resources in a dynamic grid. *Parallel Comput.*, 33(4-5):339–358, 2007.

[8] Ian Foster. A globus primer: Or, everything you wanted to know about globus, but were afraid to ask.

[9] Ian Foster, Carl Kesselman, Jeffrey Nick, and Steven Tuecke. The physiology of the grid: An open grid services architecture for distributed systems integration, 2002.

[10] Adriana Iamnitchi, Ian Foster, and Daniel C. Nurmi. A peer-to-peer approach to resource location in grid environments. In *HPDC '02: Proceedings of the 11th IEEE International Symposium on High Performance Distributed Computing*, page 419, Washington, DC, USA, 2002. IEEE Computer Society.

[11] Domenico Talia and Paolo Trunfio. Peer-to-peer protocols and grid services for resource discovery on grids. In Lucio Grandinetti, editor, *Grid Computing: The New Frontier of High Performance Computing*, volume 14 of *Advances in Parallel Computing*. Elsevier Science, 2005. ISBN 0-444-51999-8.

[12] Open grid services architecture (ogsa). `http://www.ogf.org/documents/GFD.80.pdf`, 2009.

[13] Gnutella. `http://en.wikipedia.org/wiki/Gnutella`, 2009.

[14] Carlo Mastroianni, Domenico Talia, and Oreste Verta. A super-peer model for building resource discovery services in grids: Design and simulation analysis. In *Proc. of the European Grid Conference (EGC 2005)*, volume 3470 of *LNCS*, pages 132–143, Amsterdam, The Netherlands, February 2005. Springer-Verlag.

[15] Beverly and H. Garcia-Molina. Designing a super-peer network. pages 49–60, 2003.

[16] Moreno Marzolla, Matteo Mordacchini, and Salvatore Orlando. Resource discovery in a dynamic grid environment. In *DEXA Workshops*, pages 356–360. IEEE Computer Society, 2005.

[17] Min Cai, Martin Frank, Jinbo Chen, and Pedro Szekely. Maan: A multi-attribute addressable network for grid information services. In *GRID '03: Proceedings of the 4th International Workshop on Grid Computing*, page 184, Washington, DC, USA, 2003. IEEE Computer Society.

[18] Ion Stoica, Robert Morris, David Karger, Frans M. Kaashoek, and Hari Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. In *SIGCOMM '01: Proceedings of the 2001 conference on Applications, technologies, architectures, and protocols for computer communications*, volume 31, pages 149–160. ACM Press, October 2001.

[19] Artur Andrzejak and Zhichen Xu. Scalable, efficient range queries for grid information services. In *P2P '02: Proceedings of the Second International Conference on Peer-to-Peer Computing*, page 33, Washington, DC, USA, 2002. IEEE Computer Society.

[20] Sylvia Ratnasamy, Paul Francis, Marko Handley, Richard Karp, and Scott Schenker. A scalable content-addressable network. In *SIGCOMM '01: Proceedings of the 2001 conference on Applications, technologies, architectures, and protocols for computer communications*, volume 31, pages 161–172. ACM Press, October 2001.

[21] D. Oppenheimer, J. Albrecht, D. Patterson, and A. Vahdat. Scalable wide-area resource discovery. Technical report, UC Berkeley, 2004.

[22] Sean Rhea, Dennis Geels, Timothy Roscoe, and John Kubiatowicz. Handling churn in a DHT. In *In Proceedings of USENIX Annual Technical Conference*, 2004.

[23] Antony Rowstron and Peter Druschel. Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems. *Lecture Notes in Computer Science*, 2218:329–350, 2001.

[24] C. Schmidt and M. Parashar. Flexible information discovery in decentralized distributed systems. In *Proceedings of the 12th IEEE International Symposium on High Performance Distributed Computing (HPDC'03)*, Washington, DC, USA, 2003. IEEE Computer Society.

[25] David Spence and Tim Harris. Xenosearch: Distributed resource discovery in the xenoserver open platform. In *HPDC '03: Proceedings of the 12th IEEE International Symposium on High Performance Distributed Computing*, pages 216–225, Washington, DC, USA, 2003. IEEE Computer Society.

[26] Ashwin R. Bharambe, Mukesh Agrawal, and Srinivasan Seshan. Mercury: supporting scalable multi-attribute range queries. In *SIGCOMM '04: Proceedings of the 2004 conference on Applications, technologies, architectures, and protocols for computer communications*, pages 353–366, New York, NY, USA, 2004. ACM Press.

[27] Gurmeet Singh Manku, Mayank Bawa, Prabhakar Raghavan, and Verity Inc. Symphony: Distributed hashing in a small world. In *In Proceedings of the 4th USENIX Symposium on Internet Technologies and Systems*, pages 127–140, 2003.

[28] Ben Y. Zhao, Ling Huang, Jeremy Stribling, Sean C. Rhea, Anthony D. Joseph, and John D. Kubiatowicz. Tapestry: A global-scale overlay for rapid service deployment. *IEEE Journal on Selected Areas in Communications*, 2003.

[29] Dahlia Malkhi, Moni Naor, and David Ratajczak. Viceroy: a scalable and dynamic emulation of the butterfly. In *PODC '02: Proceedings of the twenty-first annual symposium on Principles of distributed computing*, pages 183–192, New York, NY, USA, 2002. ACM Press.

[30] Petar Maymounkov and David Mazi ares. *Kademlia: A Peer-to-Peer Information System Based on the XOR Metric.* 2002.

[31] C. Greg Plaxton, Rajmohan Rajaraman, and Andréa W. Richa. Accessing nearby copies of replicated objects in a distributed environment. In *SPAA '97: Proceedings of the ninth annual ACM symposium on Parallel algorithms and architectures*, pages 311–320, New York, NY, USA, 1997. ACM.

[32] F. Araujo and L. Rodrigues. Survey on distributed hash tables. Technical report, University of Lisbon, 2006.

[33] Napster. `http://en.wikipedia.org/wiki/Napster`, 2009.

[34] Freenet. `http://en.wikipedia.org/wiki/Freenet`, 2009.

[35] Bittorrent. `http://en.wikipedia.org/wiki/BitTorrent_(protocol)`, 2009.

[36] Guoping Wang. An efficient implementation of sha-1 hash function. In *IEEE International Conference on Electro/information Technology, 2006*, pages 575–579, May 2006.

[37] K. Gummadi, R. Gummadi, S. Gribble, S. Ratnasamy, S. Shenker, and I. Stoica. The impact of dht routing geometry on resilience and proximity. In *SIGCOMM '03: Proceedings of the 2003 conference on Applications, technologies, architectures, and protocols for computer communications*, pages 381–394, New York, NY, USA, 2003. ACM Press.

[38] Yang H. Chu, Sanjay G. Rao, and Hui Zhang. A case for end system multicast. In *Measurement and Modeling of Computer Systems*, pages 1–12, 2000.

[39] Miguel Castro, Peter Druschel, Anne-Marie Kermarrec, and Antony Rowstron. Scribe: A large-scale and decentralized application-level multicast infrastructure. *IEEE Journal on Selected Areas in Communication (JSAC)*, 20(8), October 2002.

[40] Yin Li, Xinli Huang, Fanyuan Ma, and Futai Zou. Building efficient super-peer overlay network for dht systems. In *Lecture Notes in Computer Science*, pages 787–798. Springer Berlin, 2005.

[41] Miguel Castro, Peter Druschel, Y. Charlie H, and Antony Rowston. Topology-aware routing in structured peer-to-peer overlay networks. In *FuDiCo 2002: International Workshop on Future Directions in Distributed Computing*, June 2002.

[42] Marcel Waldvogel and Roberto Rinaldi. Efficient topology-aware overlay network. *SIGCOMM Comput. Commun. Rev.*, 33(1):101–106, 2003.

[43] Zhichen Xu, Chunqiang Tang, and Zheng Zhang. Building topology-aware overlays using global soft-state. In *ICDCS '03: Proceedings of the 23rd International Conference on Distributed Computing Systems*, page 500, Washington, DC, USA, 2003. IEEE Computer Society.

[44] M. Massie. The ganglia distributed monitoring system: design, implementation, and experience. *Parallel Computing*, 30(7):817–840, July 2004.

[45] Simulacija diskretnih dogodkov. `http://en.wikipedia.org/wiki/Discrete_event_simulation`, 2009.

[46] Hui Zhang, Ashish Goel, and Ramesh Govindan. Incrementally improving lookup latency in distributed hash table systems. In *SIGMETRICS '03: Proceedings of the 2003 ACM SIGMETRICS international conference on Measurement and modeling of computer systems*, pages 114–125, New York, NY, USA, 2003. ACM.

[47] Piotr Karwaczynski and Jaka Mocnik. Ip-based clustering for peer-to-peer overlays. *Journal of software*, 2(2):30–37, 2007.

Stvarno kazalo