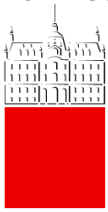
Univerza v Ljubljani
Fakulteta za računalništvo in informatiko



Uroš Čibej

Podvajanje podatkov v omrežnem računanju

DOKTORSKA DISERTACIJA

Mentor: prof. dr. Borut Robič

Ljubljana, 2007

Mojemu očetu

Kazalo

Zahvala

K nastanku pričujoče disertacije so tako ali drugače prispevali mnogi. Njihova pomoč ne sme ostati neopažena, zato so naslednje besede namenjene njim. V prvi vrsti se zahvaljujem mentorju Borutu Robiču za vodenje mojega raziskovalnega dela in pomoč pri vseh problemih, ki so se znašli pred mano. Poleg njega so mi pri delu izdatno pomagali tudi sodelavci iz laboratorija: Boštjanu se zahvaljujem za "brain-teaserje" (tiste petkove, kakor tudi tiste v ostalih dneh), Tomažu za družbo pri kavi in za vse stvari, na katere sam nisem pomislil, Juretu pa za pomoč pri sanjarjenju o reševanju sveta. Strokovno so me usmerjali še mnogi drugi sodelavci, predvsem se zahvaljujem Anhonyju Sulistiu in Rajkumarju Buyyi za razjasnitev marsikaterega pojma s področja omrežnega računanja.

Prijateljem Martinu, Urbanu, Gregorju in Tomažu se zahvaljujem za spodbudo - skupaj smo obupovali in nergali, predvsem pa veselili in uživali.

Sestrama Karmen in Suzani (ter njeni celotni družini) sem hvaležen za pozornost in brezpogojno pripravljenost priskočiti na pomoč. Iskrena zahvala pa seveda mami in očetu. Njuna predanost mi je omogočila izobraževanje in mi vedno služi kot motivacija v težkih trenutkih. Na koncu še hvala Urši za opominjanje, da svet ni lep zato, ker ga lahko opišemo z enačbami, ampak zato, ker se tega ne da.

Povzetek

Omrežno računanje postaja vse bolj pomembno področje raziskovanja, predvsem zaradi širokih možnosti za uporabo, ki jih ponuja. Omrežje je porazdeljen sistem, ki omogoča dinamično povezovanje geografsko porazdeljenih računskih virov. Glavni cilj omrežnega računanja je povezati geografsko porazdeljene vire v enovit sistem. Uporabniku s tem omogoči enostaven dostop do podatkov in računske moči, ne glede na njegovo lokacijo.

Dva osnovna tipa omrežij sta računsko in podatkovno omrežje. Glavna naloga računskega omrežja je upravljanje z računskimi viri in računsko zahtevnimi opravili, glavna naloga podatkovnega omrežja pa je upravljanje velikih količin podatkov in podatkovno zahtevnih opravil.

V podatkovnih omrežjih se za izboljšanje njihovih lastnosti pogosto uporablja podvajanje podatkov. S podvajanjem se izboljša dostopnost podatkov, obenem pa se poveča tudi odpornost omrežja na napake.

Cilj doktorske disertacije je podrobno proučiti podvajanje podatkov v podatkovnih omrežjih, predstaviti novo teoretično podlago za načrtovanje novih metod podvajanja in predlagati nekatere nove algoritme in metode.

Doslej je bilo predlaganih veliko metod za podvajanje v podatkovnih omrežjih, vendar pa so vse to *ad hoc* pristopi, t. j. brez ustrezne teoretične podlage. V disertaciji najprej podamo pregled dosedanjih raziskav in sistematično razvrstimo pristope k podvajanju.

Pri podrobnejšem pregledu problema podvajanja podatkov opazimo, da si deli kar nekaj podobnosti s teorijo razmeščanja. Na podlagi te podobnosti predstavimo množico modelov, ki vključujejo številne parametre, ki se lahko pojavijo v podatkovnih omrežjih. Dokažemo tudi, da je problem podvajanja podatkov NP-težek in celo neaproksimabilen.

Izmed zasnovanih modelov izberemo tistega, ki najbolje opisuje trenutna podatkovna omrežja in aplikacije. Da bi preverili dejansko ustreznost in kakovost tega modela, ga preizkusimo v simulacijah realnih in umetno generiranih podatkovnih omrežij. Rezultati pokažejo, da model dobro opisuje kakovost postavitve podatkov.

Predstavimo algoritme za reševanje problemov, ki jih opisuje izbrani model. Ker smo dokazali veliko časovno zahtevnost teh problemov (če $P \neq NP$), se osredotočimo le na hevristične algoritme. Predstavimo dva požrešna algoritma, dva algoritma lokalne optimizacije in genetski algoritem, ki je prilagojen za reševanje problema podvajanja. Algoritme primerjamo na množici testnih primerov in izluščimo najboljše.

Za implementacijo v podatkovnem omrežju pa zaporedni algoritmi niso najboljši, zato predstavimo tudi porazdeljeno metodo, ki uporablja zamisli našega požrešnega algoritma. To metodo smo primerjali z drugimi obstoječimi metodami in s simulacijami pokazali, da je naša metoda boljša.

Abstract

Grid computing is becoming an important new field of research, mainly because it offers such a large variety of applications. Informally, a grid is a distributed system that enables dynamic aggregation of geographically dislocated resources. It enables users to have transparent access to data and computing resouces across the grid.

Grids can be classified into two basic types: computational and data grids. The main task of a computational grid is to manage computing resources and computationally intensive tasks. The main task of a data grid is to manage huge amounts of data and data intensive tasks.

In data grids, data management applications often use data replication to improve data access time and provide better fault tolerance.

The goal of this thesis is to study data replication in data grids, present a theoretical basis for the design of new replication methods, and propose a set of new algorithms and methods.

Current approaches to data replication are mainly *ad hoc*, i.e. they have very little or no theoretical background. In this thesis we give an overview of the related work and construct a taxonomy of approaches to replication.

A detailed inspection of the data replication problem reveals many similarities with the problems in location theory. This similarities motivate us to construct a set of models that include different parameters of a data grid. We prove that the data replication problem is NP-hard and non-approximable.

From the constructed models we choose a model that describes current data grids and most typical data-intensive applications. In order to show the quality of the chosen model, we conduct simulations of real as well as artificially created data grids. The results clerly show that the model adequately describes data grids and data intensive applications.

Furthermore, we develop a set of algorithms for solving the problem described by the chosen model. Since we proved a high time complexity of this problem (if $P \neq NP$), we focus only on heuristic algorithms. The

developed algorithms are: two greedy algorithms, two algorithms for local optimization, and a genetic algorithm, which was adapted for solving the data replication problem. The algorithms are compared on a set of test cases.

However, sequential algorithms are difficult for direct implementation in a distributed system (such as the grid). Therefore we present a distributed method that uses the ideas from our greedy algorithm. We compared this method with other existing methods in simulations. The results show that our method is better.

Prvo poglavje

Uvod

V doktorski disertaciji smo se lotili področja podvajanja podatkov v omrežnem računanju. V uvodnem poglavju bomo predstavili motivacijo za raziskovanje omrežnega računanja, trenutno stanje tehnologije in smeri v razvoju novih računskih tehnologij. Natančneje bomo definirali pojma omrežnega računanja in podvajanja podatkov. Ogledali pa si bomo tudi cilje in strukturo doktorske disertacije. Nazadnje bomo v uvodnem poglavju podali tudi slovar uporabljenih izrazov.

1.1 Motivacija

V zadnjem desetletju je prišlo do korenitih sprememb v uporabi računskih tehnologij. Da bi lažje razumeli, zakaj je tehnologija omrežnega računanja postala tako zanimiva, si oglejmo trenutno stanje tehnologije, in dejavnike, ki ženejo razvoj omrežnega računanja.

1.1.1 Stanje tehnologije

Medmrežje. Računalnik je postal nepogrešljivo orodje v skoraj vsakem poklicu. V zadnjih desetih letih pa je postalo prav tako nepogrešljivo povezovanje računalnikov v medmrežje. Z dostopom do medmrežja postane slehernemu uporabniku dosegljiva ogromna količina podatkov, do katerih bi sicer težko prišel. Medmrežje pa je omogočilo tudi enostavno povezovanje računalnikov in dostopanje do njih, ne glede na zemljepisno lego.

Računska moč. Namizni računalniki nudijo povprečnemu uporabniku zelo veliko računsko moč, ki pa ostaja v veliki meri neizkoriščena. Pred nekaj leti so se začele pojavljati tehnologije, ki izkoriščajo to neizrabljeno računsko

moč namiznih računalnikov po celem svetu [124, 120, 123]. Ti projekti veljajo za predhodnike računskih omrežij. Njihova slabost pa je, da so povsem centralizirani in uporabljajo programske rešitve, ki so namenjene izključno eni aplikaciji. Podobna neizkoriščenost strojne opreme se pojavlja tudi v velikih računskih centrih. Superračunalniki in računalniške gruče so težko dostopni širšemu krogu uporabnikov zaradi varnostnih in tehnoloških razlogov. Tudi ta neizkoriščenost še dodatno kaže na potrebo po enotni, standardizirani tehnologiji, ki bo nudila boljši dostop do računskih virov.

Podatki. Poleg pojava velike količine računske moči, lahko v zadnjih letih opazimo tudi proizvajanje in pridobivanje zelo velikih količin podatkov, in sicer v različnih, predvsem pa v znanstvenih sferah. Na primer v astronomiji se količina podatkov, ki so na voljo znanstvenikom, podvoji vsako leto [13]. V genetiki pa se količina genskih zaporedij podvoji kar vsakih devet mesecev [58]. Dostop do teh podatkov je velik problem, saj želi do njih zelo veliko število ljudi. Obdelava teh podatkov pa velikokrat zahteva tudi veliko računsko moč.

1.1.2 Povezovanje virov

Kot odgovor na potrebo uporabnikov (po večji računski moči in ustvarjanju velikih količin podatkov) se je pojavilo omrežno računanje.

Omrežno računanje. Poznamo veliko definicij omrežnega računanja [128, 40, 42, 87]. Najenostavneje lahko rečemo, da je omrežno računanje standardizirana tehnologija, ki omogoča združevanje računske moči, porazdeljenih zbirk podatkov in drugih virov (npr. različnih naprav) v homogen sistem. Zaradi medmrežja je povezovanje različnih virov teoretično že izvedljivo, vendar zaenkrat ne obstaja standardna platforma, ki bi omogočila enostavno in varno souporabo porazdeljenih virov. Prav zato je postalo omrežno računanje v zadnjih nekaj letih zelo aktualno področje raziskovanja. Veliko zanimanje zanj je posledica številnih novih aplikacij, ki bodo povsem spremenile naš način dela in razmišljanja. Prvi uporabniki omrežnega računanja so bili predvsem znanstveniki, vendar se je v zadnjih letih razširilo tudi v poslovni svet in zabavno industrijo, kar je njegovemu razvoju dalo dodaten zagon.

Računska in podatkovna omrežja. Prvi korak v razvoju omrežnega računanja so bila t. i. računska omrežja (angl. *computational grid*). Slednja so naslednja stopnja v razvoju porazdeljenega računanja in za računsko zahtevne aplikacije (kot so npr. številni znanstveni izračuni) so se že zelo izkazala. Vendar pa obstajajo tudi številne aplikacije, ki niso tako zelo

računsko zahtevne, ampak za svoje izvajanje potrebujejo predvsem zelo velike količine podatkov. Take aplikacije imenujemo *podatkovno zahtevne*, saj večji del izvajalnega časa porabijo za iskanje in prenašanje podatkov (npr. iz podatkovnih baz ali datotek). Za take aplikacije osnovno računsko omrežje ni več zadostovalo. Zato se je kmalu razvila razširitev računskega omrežja, poimenovana *podatkovno omrežje* (angl. *data grid*) [20]. V takem omrežju je poseben poudarek namenjen učinkovitemu prenosu in upravljanju s podatki in metapodatki.

1.2 Podvajanje podatkov

Ker je eden od ciljev podatkovnega omrežja zagotoviti dobro dostopnost do podatkov, je pomemben del arhitekture podatkovnega omrežja namenjen optimizaciji prenosa podatkov. Za zagotavljanje boljše dostopnosti podatkov se pogosto uporablja *podvajanje podatkov* (angl. *data replication)*. Ta metoda je znana že z drugih področij porazdeljenega računanja [23], v omrežnem računanju pa je dobila še poseben pomen, predvsem zaradi podatkovno zahtevnih aplikacij.

Namen podvajanja podatkov je zagotoviti hitrejši in bolj zanesljiv dostop do podatkov v porazdeljenem sistemu. Vsak podatek ima lahko več kopij, ki so na različnih vozliščih omrežja. S tem hkrati zagotovimo tudi večjo odpornost sistema proti napakam: ko vozlišče, kjer se nahaja nek objekt, odpove, so še vedno na voljo druga vozlišča, kjer so kopije tega objekta. Ta preprosti način lahko pomembno pripomore h kakovosti sistema.

Ker je podvajanje podatkov zelo aktualno in obetavno, se je v omrežnem računanju pojavilo veliko različnih pristopov k podvajanju. Velika večina današnjih metod podvajanja pa ima skupno lastnost, da so namreč dinamične. To pomeni, da se skušajo prilagajati trenutnemu stanju v omrežju, in sicer s postavljanjem in brisanjem kopij glede na uporabo datotek in glede na druge parametre v podatkovnem omrežju. Slabost dinamičnega podvajanja je predvsem njegova teoretična neobvladljivost; podatkovno omrežje je namreč zelo zapleten sistem, ki ga je težko modelirati z dinamičnimi parametri.

1.3 Cilji disertacije

V disertaciji smo se lotili problema podvajanja podatkov na drugačen način. Podvajanje podatkov želimo opisati s formalnimi modeli, kamor bomo vklju-

čili razne parametre. Izluščili bomo tiste parametre, ki v podatkovnih omrežjih igrajo ključno vlogo pri zagotavljanju kakovosti omrežja. Kakovost postavitve podatkov pa bomo zajeli z raznimi ciljnimi funkcijami, ki zrcalijo stanje v podatkovnih omrežjih. Tako zasnovani modeli so dobra podlaga za načrtovanje sodobnih metod podvajanja. Glavni cilji disertacije so:

1. pregledati in sistematično razvrstiti obstoječe pristope,
2. definirati nabor modelov podatkovnega omrežja in podvajanja podatkov,
3. preizkusiti kakovost modelov v simulacijah,
4. zasnovati algoritme za reševanje izbranega modela,
5. eksperimentalno ovrednotiti algoritme na umetnih testnih primerih,
6. definirati porazdeljeno metodo podvajanja in
7. primerjati to metodo z obstoječimi metodami.

1.4 Pričakovani prispevki disertacije

V nadaljevanju podajamo podrobnejši opis posameznih prispevkov k znanosti.

- Pregled področja in razvrstitev obstoječih rešitev. Podvajanje podatkov je obsežno področje, zato bo naš prvi prispevek pregled področja in razvrstitev različnih pristopov k podvajanju. Razvrstitev bo služila kot podlaga za snovanje formalnih modelov podvajanja podatkov.
- Definicija modelov podvajanja in analiza zahtevnosti. Definirali bomo razne modele omrežja in razne parametre, ki lahko v njem nastopajo. Poiskali bomo dejavnike, ki najbolj vplivajo na dobro postavitev kopij v omrežju. Te dejavnike bomo lahko pozneje upoštevali pri modeliranju ciljne funkcije in pri razvoju algoritmov za podvajanje. Pokazali bomo, da je optimizacijski problem podvajanja NP-težak in celo neaproksimabilen.
- Preizkus izbranega modela v simulacijah. Izmed raznih modelov bomo izbrali tistega, ki je dovolj enostaven in obenem dobro opisuje današnja

podatkovna omrežja ter aplikacije. S simulacijami bomo pokazali, da ciljna funkcija v tem modelu dobro modelira kakovostno postavitev podatkov.

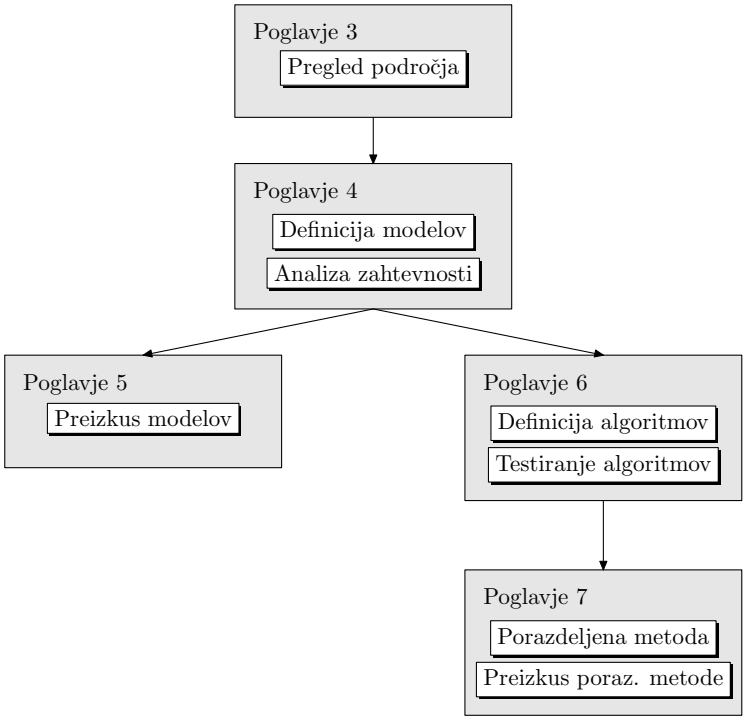
- Algoritmi za problem podvajanja podatkov. Ker je optimalno podvajanje podatkov NP-težek optimizacijski problem, se bomo osredotočili na hevristične metode za suboptimalno podvajanje podatkov. Predstavili bomo skupino algoritmov za reševanje izbranega problema. Sestavili bomo tudi ustrezne poenostavitve problemov, za katere se izkaže, da so lažje rešljive od osnovnih problemov, ter zanje poiskali natančne algoritme.

- Testiranje algoritmov na umetnih primerkih. Da bi lahko ovrednotili kakovost naših algoritmov, bo potrebno sestaviti knjižnico umetnih primerkov problemov, saj so vsi problemi novi in zanje ne obstaja knjižnica, na kateri bi lahko testirali naše algoritme. Generirani primerki problemov bodo verno odražali zapletenost današnjih omrežij. Znano je namreč, da se parametri realnih sistemov (npr. topologija , velikost datotek, velikost diskov) sledijo nekaterim verjetnostnim porazdelitvam, ki jih bomo uporabili pri generiranju testnih primerkov.

- Porazdeljena metoda za podvajanje podatkov. Razvite hevristične metode za podvajanje so klasični algoritmi, ki jih je potrebno še umestiti v podatkovno omrežje. Te algoritme je enostavno preslikati v podatkovno omrežje, če se podvajanje izvaja centralno (t.j. z enega vozlišča omrežja). Boljša rešitev pa je porazdeljena metoda za podvajanje. Predstavili bomo tako metodo, ki jo bomo umestili v trenutno arhitekturo podatkovnega omrežja. Porazdeljena metoda temelji na eni izmed zasnovanih hevristik za reševanje problema podvajanja podatkov.

- Primerjava nove metode z obstoječimi. Zadnji prispevek je primerjava porazdeljene metode podvajanja z obstoječimi metodami podvajanja. Ker zaradi težavnosti implementacije in nedostopnosti realnih sistemov primerjava v realnih podatkovnih omrežjih ni možna, se bomo za primerjavo metod ponovno zatekli k simulatorju podatkovnih omrežij.

Disertacija je sestavljena kot kaže slika 1.1.



Slika 1.1: Zgradba disertacije

1.5 Izrazoslovje

Večina znanstvene literature na tem področju je na voljo samo v angleščini. Ker želimo ohraniti slovensko terminologijo tudi v tehničnih vedah, je zelo pomembno poslovenjenje tehničnih izrazov tudi na področju omrežnega računanja. Na tako novem področju, kot je omrežno računanje, se pojavlja veliko novih izrazov, ki jih je potrebno posloveniti. V tabeli 1.1 so podani nekateri.

Podrobneje lahko razložimo izbiro izraza omrežno računanje. Področje, s katerim se ukvarjamo v disertaciji, se namreč v angleškem jeziku imenuje *grid computing*. To poimenovanje se je uveljavilo zaradi podobnosti teh porazdeljenih računalniških sistemov z električnimi omrežji (*electrical power grid*)[40]. Zato smo se odločili, da v slovenščini poimenujemo to področje *omrežno računanje*, porazdeljene sisteme te vrste pa *računska oz. podatkovna omrežja*.

Angl. izraz	Slov. izraz
grid computing	omrežno računanje
computational grid	računsko omrežje
data grid	podatkovno omrežje
semantic grid	semantično omrežje
collaboration grid	omrežje za sodelovanje
computer network	računalniška mreža
cluster	gruča
metacomputing	metaračunalništvo
peer-to-peer	enak z enakim
content delivery networks	mreže za dostavo vsebine
web service	spletna storitev
grid service	omrežna storitev
service oriented architecture	storitveno usmerjena arhitektura
data replication	podvajanje podatkov
cashing	predpomnjenje
metadata	metapodatki
replica	kopija
master copy	originalna datoteka/original
fault tolerance	odpornost na napake
authentication	overovljenje/avtentikacija
authorization	pooblastitev/avtorizacija
consistency	konsistentnost/skladnost
replica catalogue	katalog kopij
throughput	prepustnost
middleware	vmesna plast
loosely coupled	šibko sklopljeni
overhead	režija
e-Science	e-znanost
semantic web	semantična mreža
monitoring	spremljanje
taxonomy	razvrstitev
workflows	podatkovno pretokovna opravila
variable neighborhood search	lokalna opt. z menjavo sosednosti

Tabela 1.1: Poslovenjeni izrazi

Drugo poglavje

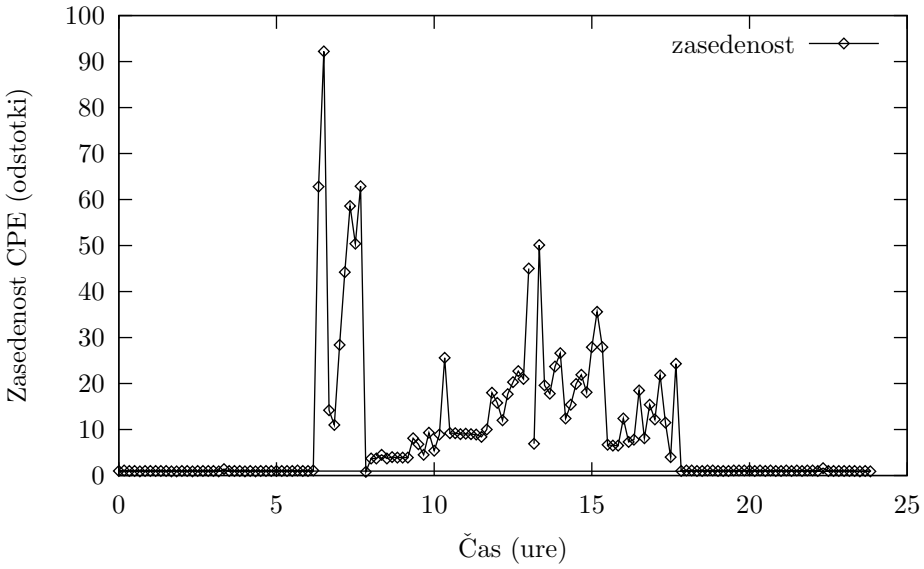
Omrežno računanje in podvajanje podatkov

Zaradi pomanjkanja ustreznih, predvsem pa standardiziranih rešitev za porazdeljeno računanje na večjem številu geografsko oddaljenih računalnikov, je v zadnjem času nastalo novo raziskovalno področje, ki so ga poimenovali omrežno računanje (angl. *grid computing*). Uporabnik omrežnega računanja naj ne bi skrbel, kje se računanje opravi. Nasprotno: te skrbi naj bi bil rešen, ustrezno orodje za omrežno računanje pa naj bi zagotavljalo hitro, varno in ponovljivo izvedbo uporabnikovih računskih opravil. Za dosego tega cilja mora tako orodje omogočiti uporabo raznih virov (angl. *resource*), ki so dostopni na računalniški mreži. Viri so različni, najpomembnejši so seveda procesor in vhodno-izhodne naprave za shranjevanje podatkov, poleg teh pa so tu še najrazličnejše specialne vhodno-izhodne naprave in računalniško vodeni znanstveni inštrumenti ipd.

Že v uvodu smo si ogledali nekatere poglavitne razloge za pojav omrežnega računanja. V nadaljevanju pa sledi bolj natančen opis dejavnikov, ki so gonilo razvoja te tehnologije.

- Potreba po sodelovanju. Delovanje organizacij se je v zadnjih desetletjih močno spremenilo. Ljudje postajajo vedno bolj mobilni, organizacije in projekti vse bolj globalni. Zato se je pojavila velika potreba po orodjih za enostavnejše sodelovanje med posamezniki in projektnimi skupinami. Med ta orodja ne spada zgolj programska oprema za konference, ampak tudi programi za souporabo podatkov, aplikacij, predstavitev, itd.

- Večja potreba po računski moči. Sodobni problemi terjajo vedno več računske moči. Vendar je ne potrebujejo stalno, temveč le občasno



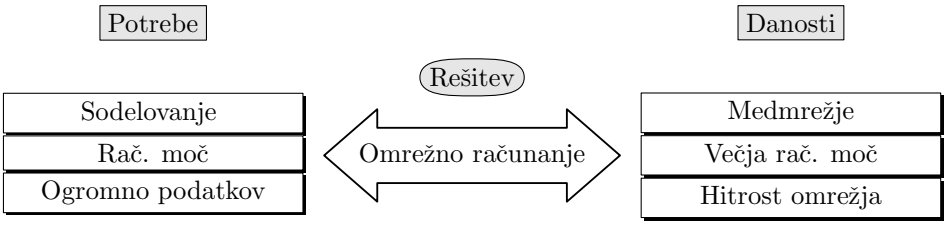
Slika 2.1: Tipična izraba računske moči

(npr. analiza enomesečnega poslovanja podjetja). Zato se nakup in vzdrževanje superračunalnikov ne izplačata. Manjša podjetja in organizacije pa si večje računske infrastrukture tudi ne morejo privoščiti.

- Ogromne količine podatkov. Zaradi vse kompleksnejših znanstvenih eksperimentov in zajemanja različnih podatkov za kasnejšo analizo, so se začeli pojavljati številni problemi v zvezi z dostopnostjo teh podatkov širšemu krogu uporabnikov, obenem pa tudi pomanjkanje ustrezne računske moči za analizo tako velikih količin podatkov.

- Vseprisotnost medmrežja. Težko je najti računalnik, ki ne bi bil povezan v medmrežje, zato je dandanes zelo enostavno in običajno komunicirati s katerimkoli računalnikom na svetu.

- Neizkoriščenost procesorske moči. Namizni računalniki so čedalje močnejši, tipičen uporabnik pa izkoristi zelo malo računske moči, ki jo ponuja njegov namizni računalnik. Na sliki 2.1 je prikazan primer obremenjenosti osebnega računalnika v enem dnevu. Jasno je razvidno, da je večino časa (še posebej pa preko noči) procesor povsem neobremenjen. Velikokrat so podobno neizkoriščeni namenski, visoko zmogljivi (in temu primerno dragi) računalniški sistemi. Očitno je, da njihovi lastniki potrebujejo tehnologijo, ki jim bo omogočila deljenje (tudi prodajanje) neizkoriščene računske moči z drugimi uporabniki.



Slika 2.2: Omrežno računanje kot rešitev

- Hitro naraščanje hitrosti mrežnih povezav. Dobro znan je zakon Gordona Moora [72], da se število elementov v integriranih vezjih podvoji vsakih 18 mesecev. Kljub starosti (leto 1965) zakon še vedno v veliki meri drži. Manj znano pa je, da se hitrost komponent v računalniških mrežah podvoji vsakih 6 mesecev [82] (od leta 1997 dalje). To pomeni, da je komunikacija med porazdeljenimi procesi v računalniški mreži vse manj ozko grlo pri izvajanju aplikacij, ki za izvajanje potrebujejo veliko količino podatkov.

Kot vidimo na sliki 2.2, lahko naštete razloge za pojav omrežnega računanja razdelimo na potrebe in danosti. Omrežno računanje pa je tehnologija, ki omogoča reševanje številnih problemov, z boljšim izkoriščanjem danosti, ki so že na voljo, a jih doslej nismo znali dovolj dobro povezati.

2.1 Razlike med omrežnim računanjem in podobnimi tehnologijami

Omrežno računanje ima veliko skupnih lastnosti z nekaterimi sorodnimi sistemi. Površno poznavanje bi lahko hitro vodilo do enačenja nekaterih porazdeljenih sistemov iz zadnjih desetletij. Zato bomo podrobneje opisali sisteme, za katere se najpogosteje zmotno trdi, da so lahko nadomestek omrežjem. Ti sistemi so sistemi enak z enakim, mreže za dostavo vsebine in računalniške gruče. Še posebej se bomo osredotočili na razlike med temi sistemi in na možnost njihove integracije v omrežno računanje.

2.1.1 Sistemi enak z enakim (P2P)

V zadnjih letih je postala zelo priljubljena tehnologija *enak z enakim* (angl. *peer-to-peer*), poimenovana tudi sistemi P2P . To so posebni primeri porazdeljenih sistemov, zgrajeni na aplikacijski plasti medmrežja [11]. Vsako

vozlišče lahko komunicira neposredno z vsakim drugim vozliščem s pomočjo usmerjevalnega protokola na P2P nivoju. Vsako vozlišče lahko povprašuje po želenih objektih (glasbi, video posnetkih, ipd.) na drugih vozliščih preko logičnih povezav mreže P2P.

Sisteme P2P lahko opišemo, če jih primerjamo s klasičnim modelom odjemalec-strežnik. Tu je strežnik osrednja nadzorna točka, pri sistemih P2P pa igra vsako vozlišče vlogo strežnika in odjemalca hkrati. Kot odjemalec deluje takrat, ko povprašuje po želenih podatkih ali prenaša najdene podatke. Kot strežnik pa deluje takrat, ko streže poizvedbam drugih vozlišč ali dostavlja zahtevane podatke.

Omrežno računanje in sistemi P2P sta tehnologiji, ki se razvijata vzporedno, imata kar nekaj podobnosti, vendar tudi nekaj ključnih razlik.

- Ciljni uporabniki: znanost proti zabavi. Ključne razlike med sistemi so posledica različnih ciljnih skupin uporabnikov. Omrežno računanje je bilo v začetku namenjeno znanstvenim aplikacijam, sistemi P2P pa razširjanju zabavnih vsebin.

- Varnost: zaprti proti odprtim sistemom. Omrežno računanje zahteva stroge varnostne storitve, da bi preprečili vsakršno nedovoljeno (zlo)rabo računskih in podatkovnih virov. Sistemi P2P pa so povsem odprti in uporabniki lahko anonimno uporabljajo podatke, ki so jim na voljo.

- Velikost: milijoni vozlišč proti tisočim vozliščem. Sistemi za omrežno računanje so v primerjavi s sistemi P2P še sorazmerno majhni (po številu vozlišč).

- Storitve: namenski proti splošnim sistemom. Omrežja so zasnovana tako, da nudijo splošne storitve, na katerih lahko uporabniki zgradijo svoje aplikacije. Sistemi P2P pa so namenski sistemi (največkrat za izmenjavo datotek), kjer ni mogoče graditi novih aplikacij.

- Informacijski sistemi: povsem porazdeljeni proti centraliziranim. Zaradi obvladljive velikosti in večje enostavnosti so trenutni sistemi za pridobivanje informacij v omrežjih centralizirani. Sistemi P2P imajo povsem porazdeljene informacijske sisteme, informacijo pa posamezna vozlišča pridobivajo z različnimi protokoli, kot so Gnutella [103], Freenet [22], Tapestry [104], CHORD [25] in drugi.

- Odpornost na napake: nestabilni proti stabilnim sistemom. Ker so vozlišča sistemov P2P večinoma osebni računalniki, ki zelo dinamično

vstopajo in izstopajo iz omrežja, so tudi podatki, ki so na teh vozliščih, velikokrat nedostopni. To dela sisteme P2P nezanesljive. Vozlišča v sistemih omrežnega računanja pa so pogosto kar celotne ustanove, kjer je za zanesljivost veliko bolje poskrbljeno kot pri osebnih računalnikih iz sistemov P2P.

Kljub tem razlikam pa se v zadnjih letih nakazuje združevanje obeh tehnologij [39, 94]. Na eni strani postajajo sistemi omrežnega računanja vse večji, zato se vanje vse pogosteje vgrajuje informacijske sisteme (razvite za sisteme P2P), ki omogočajo obvladovanje veliko večjega števila vozlišč. Po drugi strani pa tudi sistemi P2P zahtevajo čedalje več varnosti in zanesljivosti, zato se protokoli iz omrežnega računanja vpeljujejo tudi v sisteme P2P.

2.1.2 Mreže za dostavo vsebine

V medmrežju so se pojavile številne storitve, ki so zelo obremenjene, obenem pa podjetja, ki z njimi upravljajo, potrebujejo zelo dobro kakovost dostopa do svojih vsebin. Za izboljšavo kakovosti zelo obremenjenih strežnikov so bile razvite *mreže za dostavo vsebine* (angl. *content delivery networks - CDN*). Mreža za dostavo vsebine je množica geografsko porazdeljenih strežnikov, ki upravljajo s kopijami spletnih strani ali podobnih podatkov [5, 76]. Ko uporabnik dostopa do storitve ali podatkov, mreža za dostavo vsebine odjemalcu pošlje podatke z najbližjega strežnika. Na ta način se lahko razbremeni nekatere strežnike in posledično izboljša kakovost dostopa do vsebin.

Večina mrež za dostavo vsebine je namenjena zgolj komercialni uporabi. Najbolj znana ponudnika sta Akamai [105] in Digital Island [110]. Posledica komercialnosti teh sistemov je, da veliko podrobnosti o njihovi implementaciji in uporabljenih protokolih ni znanih. To je prva razlika v primerjavi z omrežnimi sistemi, ki stremijo k povsem odprtim in standardiziranim protokolom. Ostale ključne razlike so naštete v nadaljevanju.

- Storitve: splošni proti namenskim sistemom. Mreže za dostavo vsebine so namenjene zgolj enemu cilju (razbremenjevanju strežnikov), zato jih ni mogoče uporabiti za splošne aplikacije.

- Združevanje: nezmožnost združevanja proti dobri združljivosti. Kot smo že zgoraj omenili, so mreže za dostavo vsebine sistemi, ki so praviloma v lasti enega podjetja. Združevanje različnih mrež je zato zelo

težko. Omrežno računanje pa je zasnovano kot zelo prilagodljiv sistem, kjer lahko različna omrežja enostavno združujemo in tako tvorimo t. i. virtualne organizacije [42].

- Dostop do storitev: odprti proti zaprtim sistemom. Ker so v CDNjih zgolj namenske storitve, v katerih imajo uporabniki zelo omejene pravice (samo branje podatkov), so ti sistemi povsem odprti za vse uporabnike. V omrežjih lahko ponudnik storitve točno določi komu dovoljuje uporabo svoje storitve.

Tudi mreže za dostavo vsebine so začele uporabljati metode, razvite za omrežno računanje in sisteme P2P. Ena izmed aktualnih raziskovalnih tem je predvsem povezovanje različnih mrež za dostavo vsebine. V ta namen že uporabljajo protokole, razvite za omrežno računanje.

2.1.3 Računalniške gruče

Buyya [14] definira računalniško gručo kot množico šibko sklopljenih računalnikov, ki navzven delujejo kot homogena celota. Največkrat so računalniki v gruči povezani preko hitrih lokalnih mrež. Uporabljajo se za izboljšanje hitrosti ali zanesljivosti posameznih računalnikov, uveljavili pa so se predvsem zaradi zelo ugodnega razmerja cene in moči v primerjavi s primerljivimi namenskimi superračunalniki. Ključne razlike med računalniškimi gručami in omrežji so podane v nadaljevanju.

- Homogenost: homogeni proti heterogenim sistemom. Vozlišča v računalniških gručah so praviloma homogena, t.j. temeljijo na enaki arhitekturi in programski opremi. To zelo olajša izdelavo mehanizmov za izvajanje aplikacij, npr. razvrščanje opravil.

- Varnost: brez varnostnih mehanizmov proti velikemu poudarku na varnosti. Ker so vsa vozlišča računalniške gruče in uporabniki praviloma del iste organizacije, posebnih varnostih mehanizmov v gručah ni.

- Neodvisnost: vozlišča pod nadzorom administratorja proti neodvisnim vozliščem. Vozlišča v gručah so praviloma del ene same organizacije. Nastavitve parametrov vozlišč lahko organizacija enostavno samo kontrolira in nastavlja. V omrežjih so vozlišča pod nadzorom različnih organizacij. Posamezna organizacija zato nikoli nima nadzora nad celotnim omrežjem.

- Geografska porazdeljenost: povezanost preko lokalne mreže proti povezanosti preko medmrežja. Vozlišča v gručah so povezana preko lokalne mreže. promet po lokalni mreži lahko organizacija nadzoruje. Zato tudi v aplikacijah lažje predpostavimo, da nam bo komunikacija med vozlišči vedno na voljo.

Za računalniške gruče je bilo razvitih veliko tehnologij in protokolov, ki se pojavljajo tudi v računskih omrežjih. Med najbolj uveljavljenimi sta MPICH-G2 [61], implementacija vmesnika MPI za računska omrežja, in razvrščevalnik Maui [57].

2.2 Aplikacije

Omrežno računanje se je pojavilo kot odgovor na razvoj tehnologije na eni strani in potrebe znanstvene skupnosti na drugi. Aplikacije so trenutno namenjene dokaj ozki skupnosti znanstvenikov. Glavni pomislek skeptikov je pomanjkanje aplikacij za širše množice. Zagovorniki omrežnega računanja pa vztrajajo, da se bodo številne potrebe in možnosti pojavile, ko bo razvoj dosegel določen nivo in bo na voljo ustrezna infrastruktura. Podobno se je v preteklosti zgodilo z medmrežjem, ki je bil sprva uporaben le za peščico znanstvenikov, danes pa je postal ključno orodje za delo in komunikacijo.

V nadaljevanju bomo opisali nekatere možne scenarije, v katerih lahko izkoristimo omrežno računanje. Najbolj iskane so aplikacije v poslovnih okoljih, saj je tam tudi največ potencialnih končnih uporabnikov in nenazadnje tudi največ denarja, ki je pri razvoju nove tehnologije nepogrešljivo.

2.2.1 Nekaj možnih scenarijev za uporabo omrežnega računanja

Kot smo že omenili, je omrežna tehnologija uporabna na številnih področjih. Foster [42] opisuje nekatere možne scenarije, kjer bi lahko omrežno računanje največ pripomoglo k izboljšanju trenutnih pristopov.

- Podjetje, ki se odloča o odprtju nove podružnice, mora upoštevati številne dejavnike. Kompleksen finančni model mora imeti dostop do velike količine podatkov (ponavadi geografsko porazdeljenih) o preteklih dejavnostih podjetja in njihovih (ne)uspehih, iz katerih lahko pridobi ključno informacijo za odločanje. Dodatno lahko na sestankih vodilnega kadra podjetja interaktivno poženejo model z različnimi scenariji (angl. *what-if scenarios*). V takih primerih je pogosto potrebna

velika računska moč, ki jo s pomočjo omrežnega računanja lahko zagotovimo natanko takrat, ko jo potrebujemo. Še posebno pomembna zahteva takega podjetja je zagotavljanje varnosti njihovih podatkov - prav omrežno računanje je tako zastavljeno, da daje varnosti podatkov velik poudarek.

- Industrijski konzorcij začne študijo razvoja nadzvočnega letala naslednje generacije. Za to so potrebne natančne simulacije celotnega letala, kar vključuje izračune z raznih področij. Simulacija lahko vključi različne programske komponente, ki so bile razvite pri posameznih partnerjih, in tako deluje kot celota, ki ima na voljo vse podatke o letalu, ne glede na to, kje se v resnici nahajajo.

- Zgodi se razlitje strupene snovi in grozi ekološka katastrofa. Skupina za reševanje kritičnih situacij se odzove na razlitje z ustreznimi ukrepi. Za čim bolj učinkovito ukrepanj so potrebni številni izračuni gibanja razlitja v odvisnosti od geografskih značilnosti (kot so npr. reke, vodna zajetja in zalivski tokovi), izdelava načrta izseljevanja populacije, napotitev reševalnih ekip na ustrezne lokacije, obveščanje bolnišnic in zdravnikov, itd.

- Tisoči fizikov v stotinah laboratorijev in univerz po svetu sodelujejo v skupnem projektu pospeševalnika delcev v CERN-u v Švici. Ogromne količine podatkov iz pospeševalnika (petabajti) morajo biti na voljo vsem tem znanstvenikom za obdelavo in analizo.

Našteti scenariji so zelo raznoliki, tako glede uporabnikov, kot velikosti problema in tipov storitev, ki jih uporabljajo. Trenutno so storitve, ki bi naštete scenarije oživile do delujočih aplikacij, še vedno v raziskovalni fazi in veliko problemov bo potrebno rešiti, preden bodo zaživele tudi v praksi.

2.2.2 Nekatere delujoče aplikacije

Vendar so veliki koraki že storjeni, zato bomo opisali tudi nekaj aplikacij, ki že dobro delujejo v omrežjih.

- Znanstveni portali. Znanstveniki se pogosto soočajo z dokaj strmo krivuljo učenja ob nameščanju in uporabi nove programske opreme. Znanstveni portali olajšajo uporabo naprednih metod za reševanje problemov, tako da omogočajo klicanje teh metod z oddaljenih strežnikov kar s pomočjo internetnih brskalnikov ali drugih enostavnih odjemalcev. Take metode se tudi izvajajo na oddaljenih strežnikih znotraj

omrežja, vendar je razporejanje opravil za uporabnika povsem nevidno. Trenutno so v razvoju portali za biologijo, fuzijo, računsko kemijo in številne druge discipline. Primeri znanstvenih portalov so BiologyWorkbench [107], Cactus [109], GPDK [112] in LaunchPad [115].

- Porazdeljeno računanje. Zmogljive namizne računalnike in hitre mrežne povezave lahko povežemo v eno celoto in tako nudimo izjemno veliko računsko moč. Podjetje *Entropia* in njegov sistem FightAIDSAtHome [120] je zelo dober primer izkoriščanja namiznih računalnikov za kompleksne izračune. Ta sistem uporablja več kot 30.000 namiznih računalnikov (večinoma prostovoljcev) za analizo kandidatov za zdravilo za aids. Drugi primer se je zgodil v letu 2001, ko so matematiki iz ZDA in Italije rešili zelo kompleksen optimizacijski problem, imenovan Nug30 [123]. Od 630 do 1006 računalnikov je potrebovalo en teden za izračun, skupaj pa so porabili 96.000 ur procesorskega časa. V prihodnosti bo pohitritev mrežnih povezav in omrežnih tehnologij še povečala nabor in velikost problemov, ki so primerni za reševanje v računskem omrežju.

- Analiza velikih količin podatkov. Veliko zanimivih znanstvenih problemov zahteva analizo velikih količin podatkov. Za take probleme je pridobivanje porazdeljenih računskih in pomnilniških zmogljivosti zelo dobrodošlo. Porazdeljeni viri so pri takih aplikacijah še toliko bolje izkoriščeni zaradi naravne vzporednosti, ki se pojavlja pri podatkovno zahtevnih problemih. Najboljši primer podatkovno zahtevne aplikacije je analiza petabajtov podatkov, ki jih bodo dobili pri fizikalnih poskusih (fizika visokih energij), še posebej v visoko zmogljivem pospeševalniku delcev, ki je trenutno še v gradnji, t.i. veliki hadronski trkalnik [29] (angl. *large hadron collider - LHC*). Ti problemi [116, 121] bodo potrebovali deset tisoče procesorjev in stotine terabajtov pomnilniškega prostora za shranjevanje vmesnih rezultatov. Zaradi različnih tehničnih, pa tudi političnih razlogov, bi bilo zelo nepraktično sestaviti te vire na enem samem mestu. Vse sodelujoče organizacije pa lahko skupaj zagotovijo vse te vire, delijo pa si lahko tudi procedure za analizo podatkov in obdelavo rezultatov analiz.

- Izračuni v realnem času. Znanstveni inštrumenti, kot so na primer teleskopi, sinhrotroni, ciklotroni in elektronski mikroskopi, proizvajajo neobdelane podatkovne tokove, ki so običajno šele naknadno računsko

obdelani. Vendar bi obdelava teh podatkov v realnem času bistveno izboljšala potek poskusa in s tem seveda tudi rezultate. Tak primer je lahko astronom, ki proučuje Sončeve izbruhe z radijskim teleskopom. Zaznavanje izbruhov je računsko zelo zahtevno, če pa bi imeli na voljo veliko računsko zmogljivost, bi se astronomi lahko osredotočili na posamezen izbruh takoj, ko se ta zgodi, in s tem izboljšali kakovost rezultatov [118, 114].

- Sodelovanje na daljavo. Raziskovalci pogosto težijo, ne samo k združevanju podatkov in računske moči, ampak tudi človeških virov. Sodelovanje pri formulaciji problema, pri analizi podatkov ipd. so zelo važne aplikacije omrežnega računanja. Npr. astrofizik, ki je izvajal terabajtno simulacijo, bi želel, da njegovi sodelavci po vsem svetu vidijo njegove rezultate in jih v realnem času tudi prediskutirajo. Projekta, ki se ukvarjata z omrežji za sodelovanje sta Teragrid [125] in Birn [108].

- Finančne analize. Cilj omrežnega računanja pri finančni analizi je hiter dostop do trenutnih in preteklih podatkov o tržišču, s katerimi bi lahko izdelali in simulirali bolj kompleksne finančne modele in zagotovili hitrejši odziv na spremembe na trgu. Omrežno računanje omogoča povezovanje virov podatkov v homogeno celoto. Ti so zelo razdrobljeni in zato sta zbiranje in analiza relevantnih podatkov mnogo počasnejša kot bi lahko bila. Druga prednost omrežnega računanja je zmožnost povezovanja računskih zmogljivosti, kar bo omogočilo računanje mnogo kompleksnejših in zato bolj natančnih modelov trga. S finančno analizo na računskih omrežjih se ukvarja npr. podjetje IBM [113].

- Vladne aplikacije. Aplikacije v vladnih ustanovah se osredotočajo predvsem na dostop do ogromnih količin podatkov, ki jih imajo na voljo različne agencije, ministrstva in druge državne ustanove. To omogoča hiter dostop za reševanje kritičnih problemov, kot so npr. nujne situacije, in tudi v normalnem delovanju. Ključno okolje omogoča bolj učinkovito odločanje s hitrejšim odzivnim časom. Omrežno računanje omogoči ustvarjanje virtualnih organizacij, ki vključujejo številne državne ustanove. To je potrebno predvsem takrat, ko so potrebne analize velikih količin podatkov, da bi se lahko rešilo dani problem. V vladnih ustanovah je pri takem poslovanju potrebno zagotoviti predvsem varnost. V omrežnem računanju pa je predvsem varnosti po-

svečena velika pozornost, zato so aplikacije, ki so zgrajene na tej arhitekturi, tudi zelo varne. Primer vladnega projekta je NARA [122].

2.3 Vrste omrežij

Iz zgornjih aplikacij vidimo, da so možnosti uporabe omrežnih tehnologij zelo široke. Omrežja zato ponujajo tudi zelo različne tipe storitev, glede na to, za katere vrste aplikacij se omrežje najbolj uporablja.

- Računsko omrežje. Računsko omrežje je osnovna vrsta omrežja: ponuja storitve za dostop do računskih zmogljivosti geografsko porazdeljenih vozlišč. Obenem ponuja tudi informacijsko storitev za iskanje vozlišč, ki nudijo določeno računsko zmogljivost, in določen tip teh zmogljivosti (npr. operacijski sistem, različice prevajalnikov itd.).

- Podatkovno omrežje. Podatkovno omrežje je nadgradnja računskega omrežja. Poleg računskih zmogljivosti nudi dostop do porazdeljenih podatkov, učinkovit in varen prenos podatkov in informacijsko storitev za iskanje specifičnih podatkov.

- Semantično omrežje. Semantično omrežje je nadgradnja podatkovnih omrežij. V semantičnem omrežju so vse storitve, podatki in računski viri opisani na standarden način, obenem pa tako omrežje ponuja tudi storitve, ki omogočajo enostavno odkrivanje in povezovanje storitev, da lahko uporabniki enostavno ustvarijo novo storitev in odkrijejo nova znanja [83].

- Omrežje za sodelovanje. Z medmrežjem so se pojavile tudi zahteve za bolj kakovostno sodelovanje na daljavo. Omrežja za sodelovanje nudijo taka orodja, ki omogočajo skupno delo velikih projektnih skupin [119].

2.4 Arhitektura omrežja

Zaradi neprestanega spreminjanja omrežnega računanja zaenkrat še ne obstaja splošno sprejeta omrežna arhitektura. Vseeno pa obstaja nabor storitev, ki so v večini omrežij. V nadaljevanju bomo najprej opisali tri generacije sistemov omrežnega računanja, ki so se pojavile v zadnjih desetih letih. Opisali bomo tudi splošno arhitekturo omrežja, ki je trenutno najširše sprejeta in uporabljena v številnih implementacijah sistemov za omrežno računanje.

- Prva generacija. To so predhodniki današnjih omrežnih sistemov. V začetku devetdesetih let se je razvilo področje, z imenom *metaračunalništvo* (angl. *metacomputing*). Cilj metaračunalništva je bila vzpostavitev infrastrukture za izvajanje računsko zahtevnih aplikacij. Primera omrežij prve generacije sta sistema FAFNER [127] in I-WAY [38].

- Druga generacija. Druga generacija omrežnih sistemov je nastala z definicijo, da mora biti infrastruktura omrežja univerzalna in standardizirana [42]. Nastal je pojem t. i. *vmesne plasti* (angl. *middleware*), ki definira nabor orodij, s katerimi je mogoče heterogen sistem povezati v enovito celoto.

- Tretja generacija. Tretja generacija sistemov omrežnega računanja se je usmerila v večjo modularnost in lažjo razširljivost programske opreme. Osnovna zamisel je t. i. *storitveno usmerjena arhitektura* (angl. *service oriented architecture* - SOA) [35]. Pod tem pojmom razumemo sisteme, katerih arhitektura je sestavljena iz šibko sklopljenih komponent. Najbolj znan primer storitveno usmerjene arhitekture so *spletne storitve* (angl. *web services*). K podobnemu konceptu kot spletne storitve teži tudi zasnova arhitekture omrežja tretje generacije, imenovana *open grid service architecture* (OGSA) [41]. Vmesna plast je namreč sestavljena iz storitev, ki so poimenovane omrežne storitve (angl. *grid services*).

Novak [74] podaja podrobnejši opis posameznih generacij omrežnega računanja.

2.4.1 Splošna arhitektura

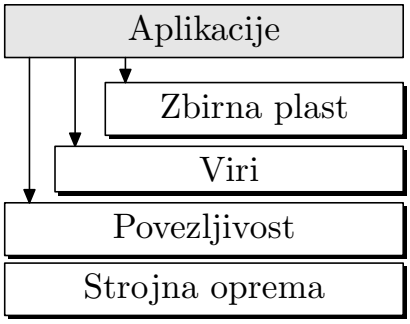
Slika 2.3 prikazuje arhitekturo omrežja po slojih.

Strojna oprema

To je sloj, kjer so definirani standardni vmesniki in protokoli za dostop do raznih strojnih virov, kot npr. računskih virov, spominskih zmogljivosti, mrežnih virov in senzorjev.

Povezljivost

Ta sloj definira protokole za komunikacijo in avtentikacijo za potrebe omrežnih transakcij. Komunikacijski protokoli omogočajo komunikacijo med viri



Slika 2.3: Arhitektura omrežja po slojih

na strojnem nivoju, protokoli za avtentikacijo pa nudijo kriptografske metode za preverjanje identitete uporabnikov in virov v omrežju.

Viri

Sloj virov definira protokole in vmesnike nad nivojem povezljivosti. Protokoli omogočajo spremljanje (monitoring), nadzor in plačilo storitev na posameznem viru.

Zbirna plast

V nasprotju s slojem virov zbirna plast ni povezana z določenim virom, ampak definira protokole, ki so globalni in skrbijo za zbiranje informacij o virih v omrežju.

2.4.2 Podatkovno omrežje

V prejšnjem razdelku smo predstavili splošen pogled na omrežje, sedaj pa si oglejmo še storitve, ki določajo podatkovno omrežje in jih umestimo v zgornje sloje splošne arhitekture.

Storitve za upravljanje s podatki lahko razdelimo v dve skupini:

1. osnovne podatkovne storitve in
2. podatkovne storitve na višjem nivoju.

Med prve štejemo:

- dostop do podatkov in podatkovna abstrakcija in
- upravljanje z metapodatki.

Med druge pa:

- upravljanje s kopijami,
- izbira kopij in podatkovno filtriranje in
- optimizacija postavitev kopij.

2.5 Definicija ključnih pojmov

Da bi lahko razumeli kontekst, v katerem hočemo izdelati ustrezne modele in zanje načrtovati algoritme podvajanja, si oglejmo opise nekaterih pojmov, ki jih bomo uporabljali v nadaljevanju dela.

- Kopija. V podatkovnem omrežju lahko ustvarimo poljubno število povsem enakih datotek. Vse datoteke iz te množice imenujemo *kopije*. Postavljene so lahko na različnih vozliščih podatkovnega omrežja. Z informacijskim sistemom moramo omogočiti aplikacijam, da lahko najdejo kopije katerekoli datoteke v omrežju.
- Originalna datoteka. Prvotno kopijo določene datoteke v podatkovnem omrežju imenujemo originalna datoteka. Razlog za razlikovanje originalne datoteke od ostalih kopij v omrežju je predvsem v zagotavljanju, da je v sistemu vedno vsaj ena kopija določene datoteke. Metode za podvajanje namreč nimajo dovoljenja za brisanje originalne datoteke, saj bi se lahko zgodilo, da v podatkovnem omrežju ne bi bilo več nobene kopije določene datoteke.
- Ime datoteke. V podatkovnem omrežju imamo dva tipa imen datotek:
  - Logično ime datoteke je ime, ki se nanaša na celotno množico kopij neke datoteke. S pomočjo tega imena lahko aplikacije najdejo fizične lokacije kopij.
  - Fizično ime je ime datoteke, ki vsebuje logično ime datoteke, in njeno trenutno fizično lokacijo. Ponavadi je del fizičnega imena tudi ime vozlišča in trdega diska ali traku, na katerem je datoteka shranjena.
- Metapodatki. Metapodatki so podatki, ki opisujejo lastnosti in kontekst drugih podatkov v omrežju. Nekatere vrste metapodatkov so navedene v nadaljevanju.

  - Metapodatki za uporabnika so specifični podatki za vsako aplikacijo, npr. informacija o množicah datotek, ki spadajo k istemu izračunu.
  - Metapodatki o izvoru so podatki o tem, kje in kako je neka datoteka nastala, ter kdo je njen avtor.
  - Varnostni metapodatki so vsi podatki, ki določajo dostop do določenih podatkov (npr. uporabnike, ki te podatke lahko berejo ali spreminjajo).
  - Knjigovodski metapodatki so podatki za vodenje dnevnika in nadzorovanje opravil.

- Katalog kopij. Katalog kopij je ena izmed osnovnih storitev v podatkovnem omrežju. Zadolžen je za hranjenje preslikav med logičnimi in fizičnimi imeni datotek, ter tudi za hranjenje metapodatkov o datotekah. Obstajajo različne implementacije katalogov kopij, od centralnih katalogov (na enem samem vozlišču omrežja) do povsem porazdeljenih (vsako vozlišče hrani en del kataloga).

- Računski element. Računski element je enotni vmesnik do računskih zmogljivosti vozlišča. Predstavlja lahko en sam procesor ali celo množico procesorjev. Računski element nudi dostop do informacij o stanju računskih zmogljivosti, ki jih predstavlja. Poleg tega omogoča uporabniku oddajanje opravil za izvajanje in nadzor nad njimi (ustavljanje, ponovno zaganjanje, spreminjanje parametrov, ipd.).

- Podatkovni vir. Podatkovni vir je enotni vmesnik do podatkovnih zmogljivosti nekega vozlišča (npr. diskovnega polja, tračnih enot ipd.). Podatkovni vir omogoča dostop do podatkov, ki se nahajajo na njegovih podatkovnih zmogljivostih in ustvarjanje novih datotek.

Tretje poglavje

Razvrstitev pristopov podvajanja

Kot smo že poudarili v uvodu, je podvajanje podatkov zelo široko področje. V tem poglavju se bomo zato osredotočili na bolj natančno definicijo podvajanja, da bi lahko v nadaljevanju disertacije lažje obrazložili naše cilje.

Glavni cilji tega poglavja so:

- izdelati razvrstitev (taksonomijo) pristopov k podvajanju,
- pregledati sorodno delo s tega področja in ga umestiti v izdelano razvrstitev ter
- identificirati slabo raziskane oz. še neraziskane teme pri podvajanju v podatkovnih omrežjih.

V naslovu tega poglavja smo uporabili izraz pristop k podvajanju, v uvodu pa smo govorili tudi o metodah in algoritmih za podvajanje podatkov. Na prvi pogled so si ti pojmi zelo podobni, zato moramo na tem mestu natančneje definirati, kakšen pomen imajo v tej disertaciji.

- Algoritem za podvajanje. Ta pojem bomo uporabili zgolj v formalnem kontekstu. Algoritem za podvajanje je formalen zapis postopka za reševanje optimizacijskega problema, ki temelji na nekem modelu podatkovnega omrežja.
- Metoda za podvajanje. Metoda je dejanska implementacija nekega algoritma. Je umestitev algoritma za podvajanje v arhitekturo podatkovnega omrežja.

- Pristop k podvajanju je sestavljen iz metode za podvajanje in konteksta, v katerem se podvajanje izvaja (lastnosti sistema). Četudi je metoda v nekem kontekstu dobra, lahko postane zelo slaba, če določene lastnosti sistema spremenimo (ali celo spremenimo cilj, ki bi ga s podvajanjem radi dosegli).

3.1 Podvajanje v sorodnih sistemih

Pri izdelavi razvrstitve se bomo osredotočili samo na podatkovna omrežja. Seveda pa se moramo dotakniti tudi podvajanja podatkov v sorodnih sistemih in pokazati ključne razlike med podvajanjem v teh sistemih in podvajanjem v podatkovnih omrežjih. Podvajanje podatkov je namreč prav zaradi teh razlik še vedno zelo aktualna raziskovalna tema.

Najprej si torej poglejmo sorodne sisteme, ki uporabljajo podvajanje podatkov za izboljšanje svojih lastnosti:

- Večprocesorski sistemi [4, 33, 48] uporabljajo podvajanje za pohitritev izvajanja opravil. V večprocesorskih sistemih se ta problem imenuje problem dodeljevanja datotek. Imamo eno samo datoteko, ki jo je potrebno podvojiti na k procesorjev, da minimiziramo čas dostopa do datoteke.

- Mreže za dostavo vsebine [5, 59, 60]. Podrobneje so že bile obdelane v poglavju 2.

- Strežniki v medmrežju [67, 68, 79]. Podvajanje strežnikov v medmrežju je podoben problem kot problem podvajanja v mrežah za dostavo vsebine, s to razliko, da se pri podvajanju strežnikov vedno podvoji celotna vsebina strežnika, v mrežah za dostavo vsebine pa lahko podvajamo tudi manjše dele celotne vsebine.

- Porazdeljene baze podatkov [21]. To je še eno zelo aktualno področje, ki uporablja podvajanje podatkov za izboljšavo lastnosti sistema. Aplikacije so na tem področju zelo specifične, saj moramo za vse transakcije zagotoviti še posebej ostre pogoje glede varnosti in skladnosti podatkov.

- Sistemi enak z enakim [47, 69]. Tudi to področje smo podrobneje obdelali v poglavju 2.

Vsako od teh področij ima svoje posebnosti, na tem mestu pa bomo predstavili samo tiste lastnosti, ki se razlikujejo od lastnosti podatkovnih omrežij.

- Heterogenost in samostojnost virov. V podatkovnih omrežjih je vsako vozlišče povsem samostojna entiteta. Dovoljenja za uporabo virov na vozliščih (npr. procesorska moč, prostor na disku) se določijo z lokalno varnostno politiko. Druga značilnost podatkovnih omrežij, je njihova velika heterogenost - v tem se podatkovna omrežja razlikujejo od ostalih (zgoraj naštetih) sistemov.

- Ogromne količine podatkov. Pri vseh zgoraj naštetih sistemih govorimo o megabajtih podatkov, ki jih lahko naenkrat potrebuje tipična aplikacija, v podatkovnih omrežjih pa imamo opravka z gigabajti ali celo terabajti podatkov. Zaradi tako velike količine podatkov je potrebno odpraviti povsem redundantno podvajanje. V sistemih z manjšo količino podatkov namreč lahko ustvarimo veliko kopij ne glede na to, koliko jih bomo pozneje dejansko potrebovali.

- Veliko večji stroški režije kot v tradicionalnih sistemih. Kot smo že opisali v prejšnjem poglavju, je varnost v podatkovnih omrežjih ključnega pomena. Zagotavljanje večje varnosti pa prinese velike stroške režije.

- Možnost razvrščanja opravil in uravnotežanja obremenitve (angl. *load balancing*). Za razliko od večine zgoraj naštetih sistemov imamo v podatkovnih omrežjih razvrščevalnike, ki dodelijo opravilo vozlišču, na katerem se bo to izvedlo najhitreje. To omogoči tudi boljšo porazdelitev povpraševanja po podatkih.

3.2 Razdelitev področja

Podvajanje podatkov je pojem, ki se pojavlja v številnih, zelo različnih kontekstih. Da bi lahko natančneje definirali področje, ki ga želimo obdelati v disertaciji, si najprej oglejmo grobo razdelitev podvajanja podatkov. V literaturi najdemo izraz podvajanje podatkov v dveh različnih pomenih.

- Kot skupek mehanizmov. Mehanizmi omogočajo enostaven in varen prenos podatkov med vozlišči porazdeljenega sistema, pridobivanje informacij o lokaciji posamezne datoteke ter zagotavljajo skladnost kopij v sistemu. Primeri protokolov in storitev za podvajanje so:

  - prenos podatkov [98],
  - katalog kopij [84],
  - storitve za povpraševanje po kopijah in upravljanje z njimi [49, 88, 89, 96].

  Do sedaj je bila velika večina raziskav usmerjena v snovanje različnih mehanizmov za podvajanje podatkov.

- Kot optimizacijski problem. Če govorimo o podvajanju podatkov kot o optimizacijskem problemu, potem predpostavljamo, da imamo v sistemu vse mehanizme, ki nam omogočajo ustvarjanje kopij in upravljanje z njimi. Cilj optimizacijskega problema pa je postaviti kopije v sistem tako, da izboljšamo določeno lastnost sistema (npr. propustnost sistema, odpornost).

V nadaljevanju doktorskega dela bomo govorili izključno o podvajanju podatkov kot o optimizacijskem problemu.

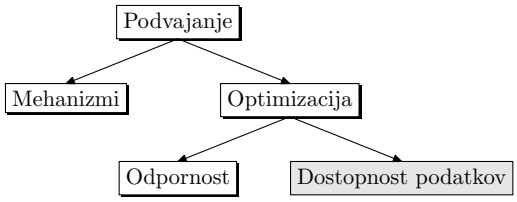
Optimizacijske pristope za podvajanje podatkov lahko nadalje razdelimo glede na cilj, ki bi ga radi dosegli z več kopijami v sistemu. Tipično sta to dva precej različna cilja.

- Odpornost na napake. V porazdeljenih sistemih so vozlišča nezanesljiva in pogosto prihaja do izpadov. Ob izpadu vozlišča postanejo tudi podatki, ki se na njem nahajajo, nedosegljivi. Z več kopijami v sistemu zagotovimo večjo verjetnost, da bo podatek v primeru izpadov ali napak v omrežju dosegljiv na vsaj enem vozlišču.

- Hitrejši dostop do podatkov. Ko imamo v sistemu več kopij, lahko vsak uporabnik dostopa do najbližje kopije. S tem dosežemo dva cilja:

  - uporabnik pride hitreje do podatkov in
  - zmanjšamo obremenjenost računalniških mrež.

Kot smo že omenili v poglavju 2, so vozlišča v omrežnem računanju ponavadi kar celotne organizacije. Verjetnost izpada celotne organizacije iz sistema je majhna. Zato so v podatkovnih omrežjih, za razliko od npr. sistemov enak z enakim [81], raziskave bolj usmerjene v podvajanje podatkov s ciljem izboljšanja dostopnosti podatkov. Tudi v doktorski disertaciji smo se lotili problema podvajanja podatkov, da bi izboljšali dostopnost podatkov v podatkovnih omrežjih.



Slika 3.1: Groba razdelitev pojma podvajanje podatkov

3.3 Razvrstitev optimizacijskih pristopov k podvajanju

V literaturi obstaja veliko različnih pristopov k podvajanju podatkov. V nadaljevanju bomo predstavili razdelitev teh pristopov. Kot smo definirali že v uvodu v to poglavje, lahko pristop k podvajanju definiramo kot delo na področju podvajanja podatkov, ki je sestavljeno iz predpostavk o sistemu, v katerem podvajanje izvajamo, in iz metode za podvajanje.

Lastnosti pristopa k podvajanju podatkov bomo razdelili v dve kategoriji:

- lastnosti sistema, na katerem se podvajanje izvaja, in
- lastnosti metode, ki se uporablja za podvajanje.

V nadaljevanju bolj podrobno razdelamo obe kategoriji lastnosti.

3.3.1 Lastnosti sistema

Oglejmo si najprej, kako lahko razdelimo predpostavke o sistemu, v katerem želimo izvajati podvajanje.

Dostop do podatkov. Omrežne aplikacije uporabljajo podatke na dva načina:

- Samo branje. Če omrežne aplikacije ne spreminjajo podatkov, potem kopij ni potrebno sinhronizirati med seboj. Večina trenutnih aplikacij v omrežju je tega tipa [51].
- Branje in pisanje. V prihodnosti se bo pojavilo verjetno več aplikacij, ki bodo podatke tudi spreminjale. Pri takih aplikacijah je potrebno pri podvajanju podatkov upoštevati tudi stroške (čas), ki so potrebni za sinhronizacijo podatkov [31].

Topologija omrežja. Vozlišča so v omrežjih lahko povezana v različne topologije.

- Hierarhična topologija. To je omrežje, v katerem so vozlišča povezana v drevo. Velikokrat velja tudi, da vsi podatki nastajajo v korenu drevesa. Primer takega sistema je podatkovno omrežje projekta EU Datagrid [111].

- Splošna topologija. Vozlišča so povezana v splošen graf, torej omrežje ne odlikuje nobena posebna struktura. Prav tako se lahko podatki pojavijo na kateremkoli vozlišču.

Tipi aplikacij. V podatkovnih omrežjih se pojavljajo različni tipi podatkovno intenzivnih aplikacij. Za potrebe podvajanja podatkov jih razdelimo v dve kategoriji.

- Odjemalci. To so aplikacije, v katerih se velike količine podatkov prenašajo neposredno do uporabnikov. S podvajanjem podatkov tako ne moremo vplivati na porazdelitev povpraševanja. Ta tip aplikacij je zelo podoben aplikacijam v mrežah za dostavo vsebine [60].

- Opravila. To so aplikacije, ki so sestavljene iz opravil. Razvrščevalniki porazdelijo opravila po vozliščih omrežja v skladu z zahtevami opravil. Ta tip aplikacij je trenutno najbolj aktualen v podatkovnih omrežjih in tudi raziskovalno najbolj zanimiv.

Razvrščanje opravil. V primeru, da imamo v sistemu aplikacije z opravili, je potrebno opravila razvrstiti po vozliščih podatkovnega omrežja. Kakovost podvajanja podatkov je tesno povezana z razvrščanjem opravil. Tako lokacija podatkov, kot tudi vozlišče, na katerem se opravilo izvaja sta namreč ključnega pomena za hitrost izvajanja. Pri izbiri strategije podvajanja je torej vedno potrebno upoštevati tudi algoritem za razvrščanje opravil, ki se bo v danem primeru uporabljal.

Dong [32] podaja podrobnejši pregled pristopov k razvrščanju v podatkovnih omrežjih V nadaljevanju pa bomo razdelali nekatere značilnosti podvajanja podatkov in poudarili tiste, ki so pomembne za podvajanje podatkov. V splošnem je razvrščanje opravil ureditev množice opravil in dodeljevanje teh opravil računskim virom, ki jih nato izvedejo. Najprej si oglejmo, katere vrste opravil so tipično na voljo v podatkovnih omrežjih.

- Neodvisna opravila. Razvrščevalnik sprejme vsako opravilo posebej in se na podlagi informacije o njem odloči, kateremu vozlišču ga bo dodelil.

- Skupina neodvisnih opravil. V tem primeru razvrščevalnik prejme večjo skupino neodvisnih opravil (angl. *bag of tasks*). Ta opravila se lahko izvajajo v poljubnem vrstnem redu, razvrščevalnik pa lahko optimizira medsebojno uporabo podatkov in tako pohitri skupno izvedbo opravil.

- Podatkovno pretokovna opravila. V tem primeru je dana množica opravil, ki so medsebojno odvisna. Opravilo in odvisnosti med podopravili so predstavljene z enostavnim usmerjenim grafom. Vozlišče v grafu predstavlja podopravilo, povezava iz vozlišča A v vozlišče B predstavlja podatkovno odvisnost opravila B od opravila A. Opravilo B mora namreč počakati, da se opravilo A izvede do konca, da lahko v svojih izračunih uporabi njegove rezultate. Razvrščevalnik mora vse te odvisnosti upoštevati pri dodeljevanju opravil vozliščem.

Glede na lokacijo razvrščevalnika ločimo tri pristope k razvrščanju podatkovno zahtevnih opravil.

- Lokalno razvrščanje. Opravila so vedno dodeljena lokalnim virom, torej vozlišču, kjer se nahaja tudi uporabnik, ki je opravilo poslal v izvajanje.

- Centralno razvrščanje. Razvrščevalnik se nahaja le na enem izmed vozlišč. Vsak uporabnik pošlje svoje opravilo temu vozlišču, razvrščevalnik pa se, glede na informacijo, ki mu je na voljo, odloči kam bo to opravilo poslal.

- Porazdeljeno razvrščanje. Vsako vozlišče ima svoj razvrščevalnik. Uporabniki pošljejo opravilo razvrščevalniku na svojem vozišču in ta se nato odloči, na katero vozlišče bo opravilo poslal.

3.3.2 Lastnosti metod

Metode za podvajanje so lahko zelo raznolike, v razdelitvi pa smo upoštevali dejavnike, ki najbolje definirajo metodo podvajanja.

Dinamičnost. Prva lastnost, po kateri ločimo metode podvajanja, je časovni potek izdelave kopij.

- Statične metode. Statične so tiste metode, ki kopije postavijo v omrežje samo enkrat, glede na takratno znanje o omrežju. Ta postavitev se ne prilagaja kasnejšim spremembam v omrežju.

- Dinamične metode. Dinamične metode prilagajajo postavitev kopij spremembam v omrežju. Npr. spremembe povpraševanja po določenih podatkih, spremembe prometa po računalniški mreži ali prostora, namenjenega podatkom, lahko sprožijo izdelavo novih ali brisanje starih kopij.

Porazdeljenost. Druga lastnost, po kateri se razlikujejo metode podvajanja med seboj, je lokacija in soodvisnost komponent, ki izvajajo podvajanje.

- Centralizirano podvajanje. Pri teh metodah predpostavimo, da obstaja centralno vozlišče, ki odloča o postavitvi kopij. Temu vozlišču je na voljo informacija o stanju omrežja in tudi mehanizmi, s katerimi lahko ustvari in izbriše kopije na drugih vozliščih.

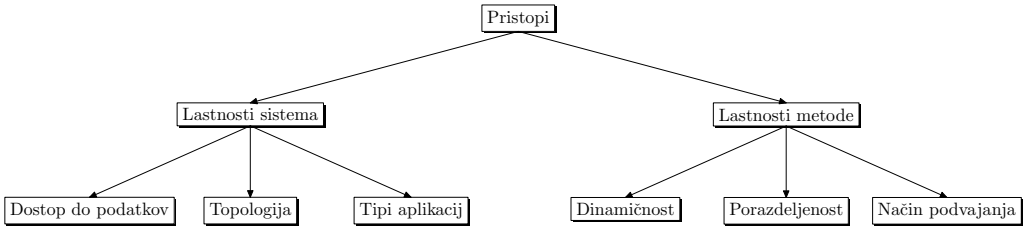
- Porazdeljeno podvajanje. Vsako vozlišče se odloča samostojno, katere kopije bo izdelalo in kam jih bo postavilo.

Način. Metode lahko razdelimo tudi glede na način, kako nastajajo nove kopije.

- Podvajanje na lastno pobudo. Posamezna vozlišča se sama odločijo, katere kopije podatkov bodo naredila na svojih virih in kdaj se bo to zgodilo.

- Podvajanje na pobudo drugih. Pri teh metodah vozlišča ustvarijo kopije datotek na svojih diskih na pobudo drugih vozlišč.

Centralizirane metode uporabljajo podvajanje na pobudo drugih, saj je centralno vozlišče tisto, ki določi kdaj in kje se ustvarijo nove kopije. Porazdeljene metode pa lahko uporabljajo oba načina.



Slika 3.2: Delitev pristopov k podvajanju

Razvrščanje opravil. Kot smo omenili že pri lastnostih sistema, je razvrščanje opravil pri podatkovno intenzivnih aplikacijah tesno povezano s podvajanjem podatkov. Metode lahko podvajajo sočasno z razvrščanjem ali pa povsem ločeno.

- Integrirano z razvrščanjem. Odločitev o podvajanju podatkov se sprejme skupaj z odločitvijo o razvrščanju opravila (ali več opravil) na vozlišča.
- Ločeno od razvrščanja. Metoda podvajanja podatkov je ločena od metode razvrščanja opravil. Razvrščanje opravil še vedno lahko vpliva na odločitve o podvajanju podatkov, vendar samo implicitno, tako da vpliva na parametre, ki se upoštevajo pri izdelavi ali brisanju kopij.

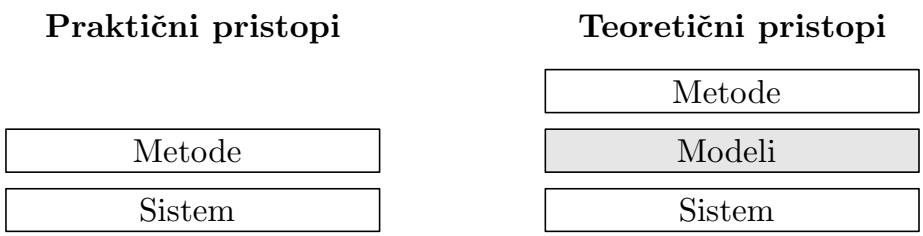
3.4 Opis obstoječih pristopov

V prejšnjem razdelku smo opisali razvrstitev pristopov k podvajanju. Cilj take razvrstitve je boljši vpogled v problem podvajanja. Razvrstitev je izdelana na podlagi obstoječih pristopov, potreb, ki jih imajo aplikacije in arhitekture v podatkovnih omrežjih. V nadaljevanju bomo opisali obstoječe pristope k podvajanju in jih umestili v našo razvrstitev. Obstoječe pristope bomo razdelili na praktične in teoretične. Prvi so *ad hoc* pristopi, saj njihove metode ne izhajajo iz kakega formalnega modela. Za razliko od praktičnih pristopov pa teoretični pristopi svoje metode zasnujejo na formalnem modelu.

3.4.1 Praktični pristopi

EU Data Grid.

Raziskave podvajanja podatkov v omrežnem računanju so postale bolj pomembne z začetkom projekta EU Data Grid [111]. Prav pri tem pro-



Slika 3.3: Praktični in teoretični pristopi

jektu so se najbolj pokazale potrebe za optimizacijo dostopa do podatkov. Načrtovanje fizikalnih eksperimentov v velikem hadronskem trkalniku [29] je namreč pokazalo, da je potrebno postaviti infrastrukturo, ki je sposobna hraniti več petabajtov podatkov. Ti podatki pa morajo biti na voljo za procesiranje tisočim opravil dnevno.V podatkovnem omrežju EU Data Grid imamo opravka z neodvisnimi, podatkovno intenzivnimi opravili. Vsako opravilo potrebuje množico datotek, da se lahko izvede. Datotek opravila nikoli ne spreminjajo (podatke samo berejo). V sistemu se nahaja centralni razvrščevalnik, ki opravilo dodeli vozlišču, na katerem se bo izvajalo. Ko se opravilo začne izvajati na dodeljenem vozlišču, mora najprej prenesti vse podatke, ki trenutno niso na voljo na lokalnih diskih. Iz zgornjega opisa izvajanja podatkovno intenzivnega opravila je razvidno, da so metode podvajanja podatkov v projektu EU Data Grid porazdeljene, povsem ločene od razvrščanja in izvajajo podvajanje na lastno pobudo.

V tem projektu je sodelovala tudi skupina, ki je zasnovala temeljno delo [7, 8, 18, 73] na področju podvajanja podatkov v podatkovnih omrežjih. Bell in ostali [7] so predstavili dve enostavni metodi podvajanja, izvirata iz dobro znanih metod za upravljanje navideznega pomnilnika v operacijskih sistemih. To sta LRU in LFU. Kadar je na vozlišču še dovolj prostora, se vedno ustvari novo kopijo datoteke. Ko pa prostora primanjkuje, se mora določiti datoteke za brisanje. Metoda LRU vedno izbriše tiste datoteke, ki najdalj časa niso bile uporabljene, metoda LFU pa tiste, ki so bile najmanjkrat uporabljene.

V okviru istega projekta so Carman in ostali [8, 18] predstavili ekonomski model za podvajanje podatkov, ki obravnava datoteke kot ekonomske dobrine. Vozlišče izračuna dobiček, ki si ga lahko obeta od določene datoteke, če bi bila ta na vozlišču. Dobiček od neke datoteke je število prenosov te datoteke s tega vozlišča. Na podlagi pričakovanega dobička se nato vozlišče odloči, ali bo datoteko kopiralo svoje lokalne diske ali ne.

Simulacije so pokazale [8], da je najbolj kvalitetna strategija podvajanja

ekonomski model. Nicholson pa pokaže [73], da se na večjih podatkovnih omrežjih veliko bolje izkažeta enostavni metodi LRU in LFU.

Ostali pristopi

Ranganathan [80] je, neodvisno od projekta EU Data Grid in na podlagi enakih predpostavk, raziskoval različne kombinacije algoritmov za razvrščanje in podvajanje podatkov. Za razliko od projekta EU Data Grid je opisal podvajanje v hierarhičnih topologijah, vozlišča pa lahko tudi potisnejo kopije na druga vozlišča. Ranganathan je pokazal, da so algoritmi podvajanja in razvrščanja zelo povezani. Predlaga pa tudi arhitekturo podatkovnega omrežja, kjer se podvajanje in razvrščanje obravnavata ločeno. S tem se poveča prilagodljivost, saj problema podvajanja in razvrščanja lažje analiziramo in rešujemo ločeno.

Takefusa [93] je opisal več različnih metod podvajanja in razvrščanja opravil. Metode podvajanja so centralizirane in tip aplikacije so neodvisna opravila. Poseben poudarek je namenjen primerjavi dveh različnih arhitektur podatkovnega omrežja. Pri prvi arhitekturi so vsi podatki shranjeni na centralnem vozlišču in tudi vsa opravila se izvajajo na tem vozlišču. Druga arhitektura ima hierarhično topologijo.

Park [75] v svojem delu predstavi nekaj kombinacij dinamičnih porazdeljenih metod podvajanja. Preizkušene so različne kombinacije metod podvajanja in razvrščanja. Topologija omrežja je hierarhična. Predlagana strategija, poimenovana BHR (angl. *bandwidth hierarchy replication*), izkorišča znanje o hierarhiji medmrežja. Če namreč zmanjka prostora za kopije na nekem vozlišču, strategija potisne kopije na vozlišče v bližini. Tako se ustvarijo kopije trenutno priljubljenih datotek v določeni regiji. Po primerjavi s strategijami LRU, LFU in ekonomskim modelom se ta strategija podvajanja izkaže za boljšo.

Lamehamedi [63] predlaga novo porazdeljeno metodo za aplikacije, v katerih nimamo opravil, ampak samo odjemalce. Predstavi tudi zanimivo hibridno topologijo podatkovnega omrežja, ki je kombinacija hierarhične in ploske topologije (hierarhična topologija, ki ima še dodatne povezave med vozlišči na istem nivoju).

Tako kot Lamehamedi tudi Tang [95] obravnava aplikacije, kjer samo odjemalci dostopajo do podatkov. Za razliko od zgornjih pristopov pa je v tem delu podano centralizirano podvajanje. Odločitve o podvajanju torej sprejme centralno vozlišče, ki kopije potisne na druga vozlišča. Poleg tega je

		EU Data Grid [7, 18, 73]	Ranganathan [80]	Lamehamedi [63]	Tang [95]	Park [75]
Lastnosti metode	Porazdeljenost	porazdeljena	porazdeljena	porazdeljena	centralizirana	porazdeljena
	Način	lastna pobuda	pobuda drugih	lastna pobuda	pobuda drugih	pobuda drugih
	Dinamičnost	dinamična	dinamična	dinamična	dinamična	dinamična
Lastnosti sistema	Tip apl.	opravila	opravila	odjemalci	odjemalci	opravila
	Dostop	branje	branje	branje	branje	branje
	Topologija	splošna	hierarhična	hierarhična	hierarhična	hierarhična

Tabela 3.1: Razvrstitev nekaterih praktičnih pristopov k podvajanju

topologija omrežja hierarhična. Tang predstavi dve novi dinamični metodi, ki ju preizkusi v simulacijah.

Tabela 3.1 podaja bolj natančno razvrstitev praktičnih pristopov k podvajanju.

3.4.2 Teoretični pristop

Povsem drugačnega pristopa kot v zgoraj opisanih primerih so se lotili Desprez [27], Chakrabarti [19] in Phan [77]. Problem najprej predstavijo kot matematični model, na podlagi tega izberejo algoritem za optimizacijo, ki ga ustrezno prilagodijo za podvajanje v podatkovnih omrežjih.

Desprez [27] je predstavil modeliranje aplikacije za bioinformatiko. V optimizacijskem problemu pa je vključil tudi razvrščanje opravil, kar je značilnost vseh tukaj predstavljenih teoretičnih pristopov. Model je predstavljen kot celoštevilski program, algoritem za reševanje optimizacijskega problema pa najprej rešuje celoštevilskemu enak linearni program, nato pa rešitve linearnega programa zaokrožuje. Avtor predstavi tudi nekatere teoretične rezultate (problem je NP-težek), model pa ovrednoti s simulacijami aplikacij iz biologije.

Chakrabarti in ostali [19] v svojem delu podajo zelo podoben rezultat. Bistvena razlika je le v reševanju problema, saj ga avtorji ne rešujejo z zaokroževanjem rešitev linearnega programa, ampak s posebej zasnovano hevristično metodo.

Phan in ostali [77] obravnavajo podvajanje podatkov skupaj z razvrščanjem podatkovno zahtevnih opravil. Predpostavijo, da vnaprej poznajo

		Desprez [27]	Chakrabarti [19]	Phan [77]
Lastnosti metode	Porazdeljenost	centralizirana	centralizirana	centralizirana
	Način	pobuda drugih	pobuda drugih	pobuda drugih
	Dinamičnost	statična	statična	statična
Lastnosti sistema	Tip apl.	opravila	opravila	opravila
	Dostop	branje	branje	branje
	Topologija	splošna	splošna	splošna

Tabela 3.2: Razvrstitev nekaterih teoretičnih pristopov k podvajanju

vsa opravila in njim potrebne podatke. Z enostavnim modelom definirajo kakovostno zaporedje izvajanja opravil, ki ga nato poiščejo z genetskim algoritmom. Hkrati izberejo tudi podatke, ki se podvojijo ob izvedbi opravila na določenem vozlišču.

Tabela 3.2 podaja bolj natančno razvrstitev teoretičnih pristopov k podvajanju.

3.4.3 Obetavna področja

Ogledali smo si dosedanje delo na področju podvajanja podatkov v podatkovnih omrežjih. Na tem mestu bomo nakazali pristope, ki v raziskavah še niso bili dovolj dobro obdelani.

Najprej lahko poudarimo, da se vsi dosedanji pristopi ukvarjajo samo z aplikacijami, ki podatkov ne spreminjajo. Če v aktualnih aplikacijah spreminjanje podatkov lahko zanemarimo, pa v bodočih tega ne bomo več mogli. Zaradi tega je potrebno postaviti temelje za podvajanje podatkov tudi pri aplikacijah, ki podatke spreminjajo.

Če izluščimo rdečo nit praktičnih pristopov, lahko to opišemo kot razvoj *ad hoc* metod na podlagi predpostavk o lastnostih sistema. Vse metode v praktičnih pristopih so dinamične in v večinoma tudi porazdeljene. Zasnovane metode so v vseh delih preizkušene na zelo malo testnih scenarijih, kar se je že izkazalo za slabo. Kot primer lahko navedemo ekonomsko metodo podvajanje [18], ki se je v začetnih preizkusih [8] izkazala za najbolj kvalitetno. Novejši preizkusi [73] pa so razkrili, da sta enostavni metodi LRU in LFU veliko boljši na večjih podatkovnih omrežjih.

Za razliko od praktičnih pristopov teoretični pristopi za podlago uporabljajo model sistema, v katerem se izvaja podvajanje. Model daje boljši vpogled v problem in omogoča uporabo številnih teoretičnih orodij, ki so že razvita za podobne probleme. Ker so modeli tipično zasnovani kot klasični optimizacijski problemi, so tudi uporabljene metode za reševanje klasični hevristični algoritmi. Teoretični pristopi imajo zato razvite zgolj statične metode podvajanja. Na podlagi matematičnih modelov pa bi bilo najbolj zanimivo in potrebno zasnovati porazdeljene metode. Porazdeljene metode so namreč v podatkovnih omrežjih bolj uporabne, predvsem zato, ker olajšajo razširljivost sistema. Vidimo lahko, da je teoretičnih pristopov k podvajanju manj kot praktičnih. Četudi so si teoretični pristopi podobni, pa se zasnovani modeli med seboj zelo razlikujejo. Vsak model je namreč zasnovan na svoj način, voden predvsem z mislijo na ciljno aplikacijo.

Zgoraj zapisano lahko strnemo v zaključke, ki bodo vodili naše nadaljnje delo v disertaciji.

1. Formalni modeli nudijo boljšo podlago za nadaljnji razvoj metod.

2. Zaenkrat ne obstaja dovolj širok nabor modelov, na podlagi katerega bi lahko zadovoljivo načrtovali boljše metode v različnih sistemih in za različne tipe aplikacij.

3. Statični modeli niso osnova zgolj statičnim metodam; zasnovati je mogoče tudi dinamične metode, ki temeljijo na statičnih modelih.

4. V podatkovnih omrežjih je podvajanje podatkov za aplikacije, ki tudi spreminjajo podatke, še povsem neraziskano.

Četrto poglavje

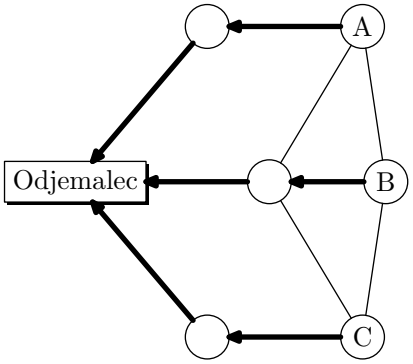
Modeliranje podvajanja podatkov

Predstavitev podatkovnih omrežij z matematičnimi modeli omogoča boljše razumevanje teh kompleksnih porazdeljenih sistemov. S pomočjo modelov lahko sestavimo veliko bolj učinkovite algoritme za številne optimizacijske probleme, ki se pojavljajo pri načrtovanju in uporabi podatkovnih omrežij. V tem poglavju bomo predstavili nabor matematičnih modelov, ki vključujejo najpomembnejše parametre. Osredotočili se bomo na tiste parametre, ki so ključni za ovrednotenje kakovosti postavitev kopij. Te modele bomo formalno analizirali z vidika njihove časovne zahtevnosti, kar bo tudi vodilo nadaljnje načrtovanje algoritmov za probleme podvajanja.

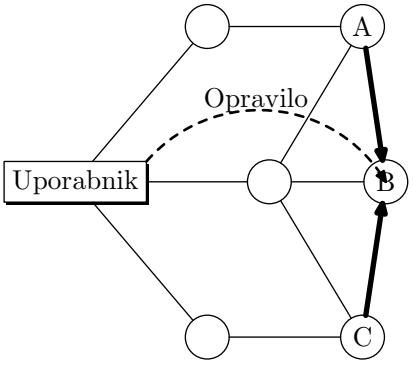
Obstaja veliko načinov za modeliranje problemov podvajanja, v tem poglavju pa bomo predstavili tiste, ki temeljijo na modelih iz *teorije razmeščanja* . Zato bomo najprej predstavili probleme razmeščanja, in sicer dva tipa takih problemov, ki sta najbližje problemom podvajanja podatkov.

Že v prejšnjem poglavju smo omenili, da ločimo pristope k podvajanju tudi glede na tip aplikacije, ki se izvaja v omrežju. Tudi modele bomo zato razdelili na dva razreda.

- Prvi razred modelov opisuje aplikacije, ki nimajo možnosti razvrščanja opravil na druga vozlišča. Delujejo torej kot množica odjemalcev, ki zahtevajo določene podatke. Slika 4.1 prikazuje primer izvajanja take aplikacije. Odebeljene povezave predstavljajo prenos datotek A, B in C do uporabnika. Pomembna značilnost takih aplikacij je, da s postavitvijo podatkov ne moremo vplivati na porazdelitev povpraševanja po določenih podatkih.



Slika 4.1: Primer izvajanja podatkovno zahtevnega odjemalca



Slika 4.2: Primer izvajanja podatkovno zahtevnega opravila

- Drugi razred modelov opisuje aplikacije, ki so sestavljene iz opravil. Ta opravila lahko razvrščevalnik dodeli vozliščem v podatkovnem omrežju. Slika 4.2 prikazuje primer take aplikacije. Prekinjena črta predstavlja dodelitev opravila vozlišču, odebeljene črte pa zopet predstavljajo prenos datotek. Vidimo, da je v tem primeru potreben prenos veliko manjše količine podatkov, čeprav opravilo potrebuje iste podatke kot odjemalec na sliki 4.1.

Poleg osnovnih modelov bomo predlagali tudi razširitve (dodatne parametre), ki jih lahko dodamo osnovnim modelom, če je v danem sistemu to potrebno. Zadnji del tega poglavja je analiza časovne zahtevnosti predstavljenih modelov oz. bolj natančno, predstavljenih optimizacijskih problemov. Pojma *problem* in *model* v disertaciji uporabljamo v praktično enakem pomenu. Ko govorimo o modelu, dajemo večji poudarek opisovanju realnih sistemov, ko pa govorimo o problemu, dajemo večji poudarek reševanju.

Z analizo časovne zahtevnosti bomo pokazali, da probleme podvajanja

zelo težko rešujemo natančno. Posledica tega je, da v nadaljevanju za te probleme ne iščemo več natančnih algoritmov, ampak se osredotočimo zgolj na hevristične algoritme.

4.1 Modeliranje v teoriji razmeščanja

Teorija razmeščanja (angl. *location theory*) [26] spada v operacijske raziskave . Problemi teorije razmeščanja sprašujejo po taki postavitvi ponudnikov storitev v določenem prostoru (zveznem ali diskretnem), ki minimizira neko ciljno funkcijo. Oglejmo si definiciji dveh osnovnih problemov teorije razmeščanja. Prvi problem je t. i. *problem mediane*, drugi problem pa je *problem k-center*.

Na prvi pogled se ti problemi precej razlikujejo od problemov podvajanja podatkov. Vendar je tudi podvajanje podatkov neke vrste razmeščanje ponudnikov (podatki) v prostoru (omrežje). Zato zasnovani modeli podvajanja podatkov črpajo navdih iz teorije razmeščanja, kar bo razvidno v nadaljevanju tega poglavja.

Opis parametrov. Obstaja zelo veliko definicij problemov razmeščanja, predstavljena različica pa vsebuje veliko elementov, ki jih bomo uporabili tudi pri definiciji problemov podvajanja podatkov v nadaljevanju. Dana je množica *porabnikov* neke storitve, $N = \{1, 2, ..., n\}$, in množica $F \subseteq N$, ki predstavlja množico lokacij, kamor lahko postavimo *ponudnike* storitve. Funkcija

$$c : F \times N \longrightarrow \mathbb{Z}^+$$

predstavlja razdaljo med morebitnim ponudnikom in porabnikom.

Problem k-mediane. Poiskati moramo tako množico ponudnikov $S \subseteq F$, kjer je $|S| = k$, da minimiziramo ciljno funkcijo:

$$\sum_{n \in N} najblizjiPonudnik(n),$$

kjer je $najblizjiPonudnik$ funkcija, ki vrne najkrajšo razdaljo do ponudnika storitev (vozlišča iz množice S). Problem mediana se največkrat pojavlja v zasebnem gospodarskem sektorju, kjer je cilj minimiziranje povprečne dostopnosti neke storitve.

Problem k-center. Osnovni problem k-center se razlikuje od problema k-mediana zgolj v definiciji ciljne funkcije. Zopet moramo torej poiskati množico ponudnikov $S \subseteq F$, kjer je $|S| = k$, da minimiziramo ciljno funkcijo:

$$\max_{n \in N} najblizjiPonudnik(n).$$

Ta ciljna funkcija predstavlja največjo razdaljo od porabnika do ponudnika. Problemi k-center se, za razliko od problema mediane, pojavljajo predvsem v javnem sektorju, kjer nas zanima "pravična" porazdeljenost storitev v nekem okolju. Pri problemu k-mediana se namreč lahko zgodi, da so tudi pri zelo dobri rešitvi nekateri porabniki zelo oddaljeni od storitev, pri rešitvi problema k-center pa se to ne more zgoditi.

4.2 Osnovni model podvajanja podatkov

V podatkovnih omrežjih lahko za podvajanje podatkov zasnujemo podobne modele kot so zgoraj opisani modeli v teoriji razmeščanja. Datoteka namreč predstavlja ponudnika podatkov, aplikacije (bodisi odjemalci ali pa opravila) pa predstavljajo porabnike teh podatkov. Na podlagi teh analogij želimo postaviti dovolj enostaven model, ki bo tudi enostavno razširljiv, obenem pa bo model že opisoval osnovni problem podvajanja, ki ga bomo potem lahko analizirali in reševali. Poleg najbolj osnovnih parametrov bomo podali tudi definicije pomožnih funkcij, ki jih bomo potrebovali v nadaljevanju.

4.2.1 Osnovni parametri

Osnovni gradnik podatkovnega omrežja so računalniška vozlišča, ki so med seboj povezana z računalniško mrežo. Vse to lahko predstavimo z enostavnim neusmerjenim grafom.

$$G = \langle V, E \rangle ,$$

kjer je V množica vozlišč, $E \subseteq V \times V$ pa množica povezav med vozlišči. Vsako vozlišče podatkovnega omrežja lahko hrani končno količino podatkov. Pomnilno kapaciteto vozlišča predstavimo s preslikavo

$$c : V \longrightarrow \mathbb{Z}^+.$$

Vsaka povezava v računalniški mreži ima neko pasovno širino t.j. sposobnost prenašanja podatkov. V nadaljevanju pa bomo namesto pasovne širine

uporabljali razdaljo med vozlišči, ki pa bo povezana s pasovno širino povezav. V podatkovnih omrežjih lahko za "razdaljo" med vozliščema vzamemo kar čas, ki je potreben za prenos enote podatkov (npr. megabajta) med njima. Vidimo, da sta pasovna širina in tako definirana razdalja obratno sorazmerni. Vozlišči, med katerima je mogoče hitro prenašati podatke, sta si blizu.

Predpostavili bomo tudi, da komunikacija lahko poteka med vsemi pari vozlišč, zato bomo razdaljo definirali nad celotno množico $V \times V$:

$$d : V \times V \longrightarrow \mathbb{Z}^+.$$

V podatkovnem omrežju je dana tudi množica podatkov, ki so na voljo uporabnikom. Predstavimo jih z množico A datotek,

$$A = \{a_1, a_2, \ldots, a_k\},$$

velikost vsake od teh datotek pa dobimo s preslikavo

$$s : A \longrightarrow \mathbb{Z}^+.$$

4.2.2 Pomožne definicije

V nadaljevanju bomo potrebovali tudi nekatere pomožne definicije in funkcije.

Iskana preslikava. Najprej definirajmo preslikavo, ki predstavlja postavitev podatkov v podatkovnem omrežju:

$$rep : A \longrightarrow 2^V.$$

Funkcija rep dodeli vsaki datoteki množico vozlišč, kjer so njene kopije. Množica 2^V namreč predstavlja potenčno množico množice V. Vsaka taka preslikava pa ni vedno tudi dopustna.

Dopustne preslikave. Preslikava $rep : A \longrightarrow 2^V$ je dopustna, če spoštuje kapacitete vozlišč, t.j.

$$\forall v \in V : \left(\sum_{a \in \{a' | v \in rep(a')\}} s(a) \right) \leq c(v). \tag{4.1}$$

Omejitev (4.1) zagotavlja, da funkcija rep nikoli ne postavi na vozlišče več datotek, kot jih lahko to vozlišče sprejme.

Da bi lahko opisali ustrezne ciljne funkcije, s katerimi bomo ovrednotili posamezno postavitev podatkov, moramo najprej definirati ceno prenosa podatkov med vozlišči.

Cena prenosa. Ceno prenosa datoteke a z vozlišča v' na vozlišče v označimo kot $cost_a(v', v)$. Cena je odvisna od velikosti $s(a)$ datoteke in razdalje $d(v', v)$ med vozliščema. Če datoteke a ni v vozlišču v', predpišemo, da je cena prenosa enaka ∞. V nasprotnem primeru je cena enaka $s(a)d(v', v)$. Torej:

$$cost_a(v', v) = \begin{cases} s(a)d(v', v) & \text{ko } v' \in r(a) \\ \infty & \text{sicer.} \end{cases}$$

Označimo s $cost(v, a)$ ceno prenosa datoteke a do vozlišča v s kateregakoli vozlišča. Datoteka bo vedno prenesena z vozlišča, ki bo minimiziralo ceno prenosa, zato lahko definiramo

$$cost(v, a) = \min_{v' \in V} cost_a(v', v).$$

Skupna notacija za ciljne funkcije. V naših modelih podvajanja uporabljamo podobne ciljne funkcije kot v teoriji razmeščanja, torej vsoto ali maksimum (po vseh uporabnikih). Toda naše ciljne funkcije se bodo vendarle razlikovale od funkcij v teoriji razmeščanja, saj v prostor ne postavljamo samo ene vrste ponudnika, ampak več različnih ponudnikov (različnih datotek).

Ciljno funkcijo lahko sestavimo na več načinov.

Vzemimo primer, množica A naj predstavlja množico ponudnikov, množica B pa množico porabnikov vseh storitev. Funkcija $f : A \times B \longrightarrow \mathbb{Z}$ naj predstavlja neko razdaljo med elementi množice A in B. Z vsoto in maksimumom lahko sestavimo naslednje funkcije.

$$\sum_{a \in A} \sum_{b \in B} f(a, b)$$

$$\sum_{a \in A} \max_{b \in B} f(a, b)$$

$$\max_{a \in A} \sum_{b \in B} f(a, b)$$

$$\max_{a \in A} \max_{b \in B} f(a, b)$$

V nadaljevanju bomo za definicijo ciljnih funkcij uporabljali vse štiri možnosti. Da ne bi pri vseh ciljnih funkcijah pisali vseh možnih kombinacij, vpeljemo nov zapis:

$$\bigoplus_{a \in A, b \in B} f(a, b)$$

predstavlja katerokoli izmed štirih zgoraj definiranih ciljnih funkcij nad množicama A in B. S takim zapisom poenostavimo definicije ciljnih funkcij v nadaljevanju, Kadar bomo hoteli eksplicitno poudariti določeno ciljno funkcijo, bomo obe operaciji v izbrani ciljni funkciji zapisali nad oznako $\bigoplus$. Zgornje štiri ciljne funkcije bi torej s to oznako lahko zapisali tudi kot

$$\bigoplus_{a\in A,b\in B}^{\Sigma,\Sigma}, \quad \bigoplus_{a\in A,b\in B}^{\Sigma,max}, \quad \bigoplus_{a\in A,b\in B}^{max,\Sigma} \text{ ali } \bigoplus_{a\in A,b\in B}^{max,max}$$

4.3 Modeli za odjemalce

Kot smo že omenili na začetku poglavja, bomo najprej obdelali modele, v katerih predpostavljamo, da aplikacije ne vsebujejo opravil. To pomeni, da s postavitvijo podatkov ne moremo vplivati na porazdelitev povpraševanja po določenih podatkih.

Modele bomo ločili tudi glede na način dostopanja do podatkov (samo branje, branje in pisanje). V primeru, ko aplikacije lahko spreminjajo podatke, moramo namreč upoštevati tudi stroške (čas) za posodobitev vseh kopij v sistemu.

4.3.1 Samo branje

Če aplikacije podatke zgolj berejo, v model vpeljemo še preslikavo, ki opisuje pogostost povpraševanja vozlišč po podatkih. Definiramo ga kot

$$f_r : V \times A \longrightarrow \mathbb{Z}^+,$$

torej je $f_r(v, a)$ pogostost povpraševanja vozlišča v po podatku (datoteki) a. V realnih sistemih lahko ta parameter določimo npr. statistično, redkeje pa tudi z analizo aplikacije, ki se v omrežju izvaja.

Ciljne funkcije. Po analogiji s teorijo razmeščanja lahko na podatke gledamo kot na ponudnike storitev, vozlišča, ki si te podatke želijo, pa kot na porabnike. Kot najenostavnejši primer vzemimo, da je množica podatkov kar $A = \{a\}$; po analogiji s problemi razmeščanja imamo torej opravka s ponudniki ene vrste storitev. Ciljna funkcija, ki je podobna tisti iz problema mediane, je

$$\sum_{v\in V} f_r(v, a)cost(v, a).$$

Če pa hočemo doseči bolj pravično porazdelitev kopij (v pomenu, kot je bil razložen v razdelku 4.1), vzamemo ciljno funkcijo, ki je podobna tisti iz problema k-center:

$$\max_{v \in V} f_r(v, a) cost(v, a).$$

Vendar pa pri podvajanju podatkov zelo redko podvajamo samo eno datoteko. Zato moramo definirati ciljne funkcije po zgornji analogiji tudi za primere, ko $|A| > 1$. Tudi v tem primeru se bomo poslužili dveh operacij - vsote in maksimuma po vseh elementih množice A. Če upoštevamo še definiciji iz prvega in drugega primera zgoraj, lahko sestavimo štiri ciljne funkcije. Vse štiri zajamemo s pomočjo predhodno definirane oznake $\bigoplus$:

$$\bigoplus_{v \in V, a \in A} f_r(v, a) cost(v, a).$$

4.3.2 Branje in pisanje

Če lahko aplikacije tudi spreminjajo podatke, moramo definirati novo funkcijo, ki to spreminjanje opisuje. Poleg pogostosti branja f_r definiramo še pogostost pisanja

$$f_w : V \times A \longrightarrow \mathbb{Z}.$$

$f_w(v, a)$ pomeni pogostost spreminjanja datoteke $a \in A$ z vozlišča $v \in V$. Ta model je bolj splošen od prej opisanih. Če namreč postavimo $f_w(v, a) = 0$ za $\forall v \in V$ in $\forall a \in A$, potem postane ta model enak tistemu, kjer aplikacije podatke samo berejo.

Ciljne funkcije. Pri konstrukciji ciljne funkcije moramo upoštevati tudi ceno posodobitve vseh kopij neke datoteke v sistemu. Končno ciljno funkcijo zapišemo kot vsoto cene branja podatkov in cene posodobitev:

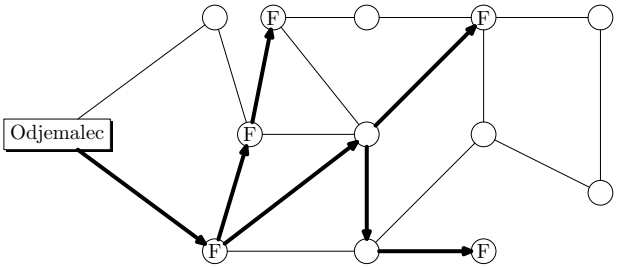
$$CenaBranja + CenaPosodobitve$$

Poglejmo si najprej kako poteka posodabljanje datotek in kolikšna je cena za posodobitev kopij ene datoteke.

Posodabljanje kopij. Obstaja zelo veliko različnih protokolov za posodabljanje kopij v porazdeljenih sistemih [23]. Za definicijo cene posodabljanja pa ne bomo uporabili nobenega konkretnega protokola, ampak bomo predpostavili, da lahko naredimo posodobitev v optimalnem času. Realni protokoli za posodabljanje seveda niso optimalni. Vsak protokol pa lahko

ovrednotimo in določimo faktor, za katerega odstopa od optimalnega. S tem faktorjem utežimo ceno optimalnega posodabljanja v ciljnih funkcijah.

Na posodobitev lahko gledamo kot na drevo, katerega koren je odjemalec, druga vozlišča tega drevesa pa so kopije, ki jih posodabljamo. Primer posodobitve kopij ene datoteke kaže slika 4.3.



Slika 4.3: Primer posodobitve množice kopij

Ker še vedno govorimo o zelo velikih količinah prenesenih podatkov, so zakasnitve med vozlišči zanemarljive v primerjavi s časom, ki je potreben za prenos vseh podatkov. V aplikacijah, kjer pa zakasnitev ni zanemarljiva, se za določanje cene komunikacije med vsemi vozlišči (multicast) uporablja cena minimalnega vpetega drevesa [2]. Če je zakasnitev zanemarljiva pa ceno posodobitve določa povezava z najmanjšo pasovno širino, oz. največjo razdaljo; to povezavo imenujemo ozko grlo. Bolj enostavno povedano, iščemo tako drevo posodobitve, da nobena povezava v njem ni dolga.

Za razliko od komunikacije med vsemi vozlišči (multicast), v posodabljanje ne vključujemo vseh vozlišč v mreži, ampak samo tista vozlišča, ki imajo kopijo spremenjene datoteke. Zato ne iščemo več vpetega drevesa ampak Steinerjevo drevo. Za razliko od vpetih dreves, so Steinerjeva drevesa taka, da ne vsebujejo nujno vseh vozlišč v danem grafu, ampak samo določeno podmnožico. Za optimalno (najhitrejše) posodabljanje datotek moramo v grafu poiskati tako Steinerjevo drevo, ki ima najmanjše ozko grlo. V literaturi je zgoraj opisani problem poznan pod imenom Steinerjevo drevo z minimalnim ozkim grlom (*angl.* bottleneck Steiner tree). Kot večina problemov s Steinerjevimi drevesi je tudi ta NP-težek, vendar zanj obstaja tudi nekaj aproksimacijskih algoritmov [10]. V realnih sistemih bi zato lahko ceno optimalne posodobitve ocenili s približnim reševanjem tega problema.

Ceno posodobitve ene datoteke z določenega vozlišča lahko tedaj defini-

ramo takole:

$$costUpdate(v, a) = botSteinerTree(rep(a) \cup v)s(a),$$

kjer je $botSteinerTree(S)$ cena minimimalnega ozkega grla Steinerjevega drevesa nad množico S (Steinerjevo drevo vključuje množico vozlišč $S \subseteq V$). Družino ciljnih funkcij, ki opisuje ceno posodabljanja kopij, lahko zapišemo kot

$$\bigoplus_{v \in V, a \in A} costUpdate(v, a).$$

4.4 Modeli z opravili

Kot smo že nekajkrat omenili, sta podvajanje podatkov in razvrščanje opravil v podatkovnih omrežjih tesno povezani dejavnosti. Prvo vprašanje, ki se nam zastavlja, je, kako v modelu obravnavati opravila in predvsem kako vanj vključiti razvrščanje, ki je pomemben dejavnik pri optimizaciji izvajanja opravil. Vprašanja se bomo lotili na dva različna načina. Predstavili bomo dva tipa modelov, in sicer:

- modele, ki opravil ne vsebujejo, in
- modele, ki opravila vsebujejo.

Drugo vprašanje, ki se nam zastavlja, pa je, kaj bi s podvajanjem v omrežju radi dosegli. Izpostavimo lahko tri cilje, ki se med seboj ne izključujejo:

- zagotoviti čim večjo propustnost sistema (čim več opravljenih opravil v časovni enoti),
- zmanjšati obremenjenost mreže (čim manj prenašanja podatkov po mreži) in
- omogočiti enakomerno računsko obremenjenost računalnikov v omrežju.

Zadnji cilj se na prvi pogled zdi odveč, saj je zagotavljanje enakomerne računske obremenjenosti računalnikov ponavadi zgolj sredstvo za doseganje večje propustnosti sistema. Vendar je enakomerna obremenjenost pomembna tudi zato, ker se računalniki na posameznih vozliščih lahko uporabljajo še za druge namene (npr. lokalno procesiranje). Z enakomerno porazdeljenostjo računskega bremena zagotovimo, da vsako vozlišče omrežju prispeva (približno) enako računsko moč in obenem posameznih vozlišč ne obremenimo preveč.

4.4.1 Implicitno vključena opravila

Za dosego vseh treh ciljev se bomo pri modelih, ki eksplicitno ne vsebujejo opravil, osredotočili na zagotavljanje enakomerne obremenjenosti omrežja. V porazdeljenih sistemih se običajno zagotavlja enakomerno porazdeljenost opravil z ustreznim algoritmom za razvrščanje.

Pri modelih, ki opravil ne vsebujejo, bomo predpostavili, da se v podatkovnem omrežju uporablja tako razvrščanje opravil, ki dodeli opravilo blizu vsem potrebovanim podatkom. Boljšo propustnost sistema pa bomo skušali doseči s tem, da bomo omogočili razvrščevalcu, da bo imel na voljo več kakovostnih vozlišč, kamor bo lahko dodelil opravilo. Tako lahko dosežemo boljšo porazdelitev opravil in s tem večjo propustnost sistema. V model eksplicitno ne vključimo samih opravil, prav tako zaenkrat ne vpeljemo kakih predpostavk o priljubljenosti podatkov. V realnih sistemih namreč zelo težko *a priori* določimo verjetnost uporabe podatkov. Zato bomo v tem modelu zaenkrat predpostavili, da imajo vsi podatki enako verjetnost uporabe.V razdelku 4.5 pa bomo predstavili tudi dodaten parameter, s katerim lahko predstavimo različne verjetnosti uporabe podatkov.

Samo branje. Sistem ima načeloma največjo propustnost, če so vsa vozlišča dobro in enakomerno obremenjena. Pri podatkovno zahtevnih opravilih to lahko dosežemo takrat, ko imajo vsa vozlišča omogočen hiter dostop do vseh podatkov. Razvrščevalec ima tako večjo izbiro, kam bo postavil določeno opravilo.

Ciljne funkcij, ki jih bomo uporabili za dosego tega cilja, so zelo podobne ciljnim funkcijam pri modelih z odjemalci. Vsako vozlišče obravnavamo kot morebitnega uporabnika vseh podatkov. Ustrezne ciljne funkcije zajema

$$\bigoplus_{a \in A, v \in V} cost(v, a).$$

Branje in pisanje. Tudi pri opravilih lahko uvedemo podobno funkcijo cene posodabljanja podatkov kot v primeru odjemalcev. Ker sta pogostosti branja in pisanja podatkov običajno zelo različni, uvedemo tudi ustrezni uteži k ceni branja in ceni pisanja. Parameter f_r predstavlja povprečno pogostost branja in f_w povprečno pogostost pisanja podatkov. Ciljne funkcije opisuje

$$f_r \bigoplus_{a \in A, v \in V} cost(v, a) + f_w \bigoplus_{a \in A, v \in V} costUpdate(v, a), \tag{4.2}$$

pri čemer je *costUpdate* enaka funkcija kot v modelih z odjemalci (razdelek 4.3.2).

4.4.2 Eksplicitno vključena opravila

V tem razdelku bomo sestavili modele, v katere bomo eksplicitno vključili tudi opravila, ki se izvajajo v omrežju.

Kot smo že omenili v poglavju 3, je v omrežju več različnih vrst opravil. Modele bomo razdelili na:

1. modele z neodvisnimi opravili in
2. modele s podatkovno pretokovnimi opravili.

Neodvisna opravila. V modelih z neodvisnimi opravili predpostavimo, da opravila poznamo že vnaprej. Definirajmo množico opravil, ki jih je potrebno izvesti na podatkovnem omrežju:

$$J = \{j_1, j_2, \ldots, j_k\}.$$

Vsako opravilo potrebuje neko množico podatkov, ki jo določa funkcija

$$D_r : J \longrightarrow 2^A.$$

Pri tem problemu ne bomo iskali le funkcije podvajanja *rep*, ampak tudi funkcijo razvrščanja opravil,

$$sch : J \longrightarrow V.$$

Rešitev problema bo torej sočasno vključevala podvajanje podatkov in razvrščanje opravil.

Samo branje. S ciljno funkcijo bi radi dosegli, da ima vsako opravilo vse potrebne podatke čim bližje. Družina funkcij, s katero ovrednotimo sočasno podvajanje in razvrščanje pa je

$$\bigoplus_{j \in J, a \in D_r(j)} cost(sch(j), a). \tag{4.3}$$

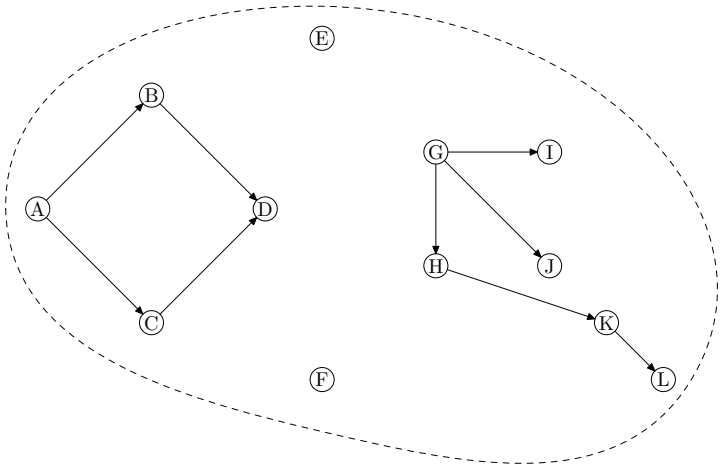
Branje in pisanje. Če lahko ocenimo povprečno pogostost branja in pisanja f_r in f_w, potem je cena posodabljanja kopij zelo podobna ceni 4.2 definirani pri implicitno vključenih opravilih (razdelek 4.4.1).

$$f_r \bigoplus_{j \in J, a \in D_r(j)} cost(sch(j), a) + f_w \bigoplus_{j \in J, a \in D_r(j)} costUpdate(sch(j), a)$$

PODATKOVNO PRETOKOVNA OPRAVILA. Poleg neodvisnih opravil, se v podatkovnih omrežjih pogosto pojavljajo tudi podatkovno pretokovna opravila. Še vedno velja enaka definicija opravil kot pri neodvisnih opravilih, s to razliko, da je sedaj množica J množica podopravil (osnovnih opravil, ki sestavljajo večja podatkovno pretokovna opravila). Poleg množice podopravil podamo še podatkovne odvisnosti med njimi, vse skupaj pa z enostavnim usmerjenim grafom,

$$T = (J, Conn),$$

kjer je $Conn \subseteq J \times J$.



Slika 4.4: Primer množice podatkovno pretokovnih opravil

Tu lahko ločimo dva tipa podatkovno pretokovnih opravil.

1. Količina podatkov, ki se prenašajo med posameznimi podopravili, je zanemarljiva v primerjavi s količino podatkov, ki so potrebni podopravilom.

2. Količina podatkov, ki se pretakajo med opravili, ni zanemarljiva.

Najprej si oglejmo prvi tip podatkovno pretokovnih opravil. Problem je na prvi pogled videti bolj zapleten kot problem neodvisnih opravil, vendar lahko ciljne funkcije definiramo ravno tako kot ciljne funkcije pri neodvisnih opravilih (funkcija 4.3). Razlog za tako poenostavitev je v tem, da smo tudi pri neodvisnih opravilih skušali povečati souporabo podatkov med posameznimi opravili.

V drugem primeru pa pretakanja podatkov med opravili ne moremo zanemariti. Zato uvedemo nov parameter, ki opisuje količino podatkov, ki

se pretakajo med podopravili. Natančneje, vsaki povezavi dodelimo količino podatkov, ki se pretočijo med njenima krajiščema:

$$flow : Conn \longrightarrow \mathbb{Z}^+.$$

Pri definiciji ciljne funkcije pa se upošteva prenašanje teh podatkov podobno kot se upošteva prenos datotek. Ciljni funkciji 4.3 v primeru neodvisnih opravil zato prištejemo še funkcijo

$$\bigoplus_{j \in J, \langle sch(j), v' \rangle \in Conn} flow(\langle sch(j), v' \rangle).$$

4.5 Razširitve

Poleg vseh parametrov, ki smo jih opisali zgoraj, lahko modelom dodamo tudi številne druge razširitve, ki se lahko pojavijo v različnih podatkovnih omrežjih in aplikacijah. V nadaljevanju bomo opisali nekaj predlogov, kako podati razširitve modelov in obrazložili v kakšnih primerih se lahko take razširitve pojavijo.

1. Cena hrambe podatkov (v odvisnosti od zasedenosti prostora)

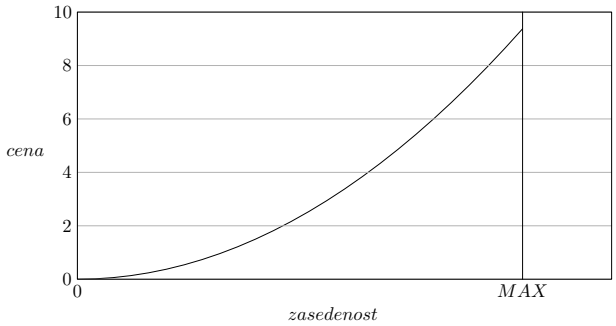
   $$storageCost : V \times \mathbb{Z} \longrightarrow \mathbb{Z}.$$

   V vseh dosedanjih modelih smo upoštevali prostor na vozliščih kot danost, ki jo lahko poljubno izkoristimo. Pogosto pa to ni tako. V podatkovnih omrežjih, kjer si iste vire deli veliko aplikacij, je zaželeno, da podvajanje uporabi čim manj prostora na vozliščih. To lahko uravnavamo z uvedbo cene hrambe podatkov. Zgoraj definirana funkcija definira ceno hranjena podatkov za vsako vozlišče. Cena pa je odvisna od količine že zasedenega prostora. Primer funkcije, ki bi uravnavala pretirano porabo prostora na nekem vozlišču, kaže slika 4.5. Več kot je zasedenega prostora, večja je cena za shranitev novih podatkov.

2. Računska zmogljivost vozlišč

   $$power : V \longrightarrow \mathbb{Z}^+.$$

   Druga predpostavka, ki smo jo tiho upoštevali pri vseh zgoraj zasnovanih modelih, je enaka računska zmogljivost vozlišč. Podatkovno zahtevna opravila pa so velikokrat tudi računsko zelo zahtevna, zaradi tega heterogenosti vozlišč s stališča računske moči ne moremo zanemariti. Zato vpeljemo funkcijo, ki vsakemu vozlišču priredi njegovo



Slika 4.5: Primer cene hrambe podatkov

računsko moč. V ciljnih funkcijah lahko utežimo prispevek vsakega vozlišča z njegovo računsko močjo. Kot posledica bodo v dobrih podvajanjih vsi podatki bližje računsko močnejšim vozliščem.

3. Priljubljenost datotek

$$popularity : A \longrightarrow \mathbb{Z}^{+}.$$

Čeprav smo v modelih s podatkovno zahtevnimi opravili vse podatke obravnavali kot enakovredne, lahko v nekaterih primerih vseeno predpostavimo določeno priljubljenost datotek. V ciljnih funkcijah velikosti datotek utežimo z njihovo priljubljenostjo, kar povzroči, da so v dobrih podvajanjih bolj priljubljene datoteke večkrat podvojene po vozliščih omrežja.

4. Gruče podatkov

$$C = \{c_1, c_2, c_3, \ldots, c_k\}.$$

Vsaka gruča c_i je sestavljena iz podmnožice datotek, ki jih določa funkcija

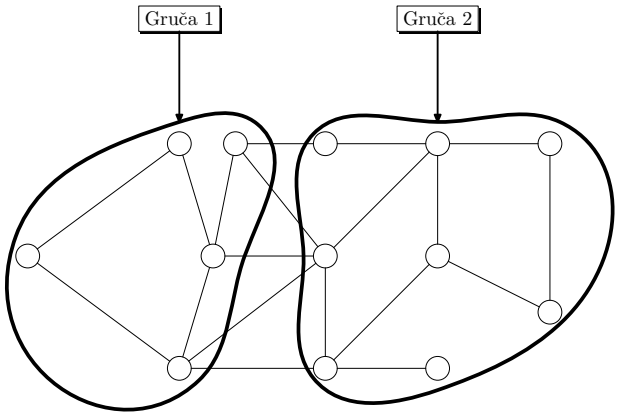
$$dataCluster : C \longrightarrow 2^{A}.$$

Četudi ne poznamo vnaprej vseh opravil, ki naj bi se izvajala v omrežju, vseeno lahko ocenimo, kakšne tipe opravil imamo. Velikokrat lahko opravila razdelimo na več različnih tipov, vsak tip opravila pa potrebuje podatke iz določene gruče podatkov. Kot primer lahko vzamemo analizo fizikalnih eksperimentov: rezultati enega eksperimenta pripadajo eni gruči, rezultati drugega eksperimenta pa drugi gruči podatkov.

Podobno kot smo pri neodvisnih opravilih poleg funkcije podvajanja *rep* iskali tudi funkcijo razvrščanja *sch*, v primeru gruč iščemo funkcijo

$$vClust : C \longrightarrow 2^V.$$

Ta funkcija dodeli gruči podatkov c_i gručo vozlišč, na kateri se lahko



Slika 4.6: Primer dodelitve podatkovnih gruč množici vozlišč

izvajajo opravila, ki potrebujejo podatke iz c_i. Slika 4.6 prikazuje primer take preslikave.

Ciljne funkcije so zelo podobne ciljnim funkcijam pri implicitno vključenih opravilih. Vsako gručo podatkov in gručo vozlišč, ki ji pripada, obravnavamo kot ločen problem, minimiziramo pa zopet povprečje ali pa maksimum ciljnih funkcij teh problemov. Tako nastane družina funkcij

$$\bigoplus_{c \in C, a \in dataCluster(c)} \quad \bigoplus_{v \in vClust(c), a} cost(v, a).$$

Samo preslikava *vClust* za definicijo še ni dovolj, saj sedaj lahko dobimo optimalno vrednost ciljne funkcije, če vsaki podatkovni gruči dodelimo prazno množico vozlišč. Med podatkovne gruče pa bi radi razdelili vsa vozlišča omrežja. Zato moramo postaviti dodaten pogoj

$$\bigcup_{c \in C} vClust(c) = V.$$

4.6 Enotno označevanje modelov

Da bi lahko v nadaljevanju enostavno opisali model, ki ga uporabljamo v danem trenutku, bomo uvedli enoten zapis za modele. Model opišemo z

n-terko

$$\langle Ap, D, G, \ldots (dodatni\ parametri) \rangle ,$$

kjer pomeni

- Ap ... tip aplikacije:
  - $client$ ... odjemalci,
  - $explJobs$ ... eksplicitno vključena opravila,
  - $implJobs$ ... implicitno vključena opravila.
- D ... način dostopa do podatkov:
  - R ... samo branje,
  - R/W ... branje in pisanje.
- G ... ciljna funkcija: $\bigoplus^{\Sigma,\Sigma}, \bigoplus^{\Sigma,max}, \bigoplus^{max,\Sigma}, \bigoplus^{max,max}$.
- Seznam dodatnih parametrov, ki so vpeljani v model
  - $storageCost$ ... cena hranjenja podatkov,
  - $power$ ... računska zmogljivost vozlišč,
  - $popularity$ ... priljubljenost podatkov,
  - $clusters$ ... gruče podatkov.

Seznam dodatnih parametrov seveda ni dokončen, saj je mogoče glede na različne zahteve v sistemu vpeljati še številne druge parametre.

Primeri. Prvi primer oznake modela je

$$\left\langle client, R/W, \overset{\Sigma,\Sigma}{\bigoplus}, storageCost \right\rangle .$$

Ta oznaka opisuje model z odjemalci, kjer se podatki tako berejo kot tudi pišejo, ciljna funkcija je $\bigoplus^{\Sigma,\Sigma}$, dodatno pa je vpeljan en parameter - cena prostora na vozlišču.

Drugi primer je

$$\left\langle explJobs, R, \overset{\Sigma,max}{\bigoplus} \right\rangle ,$$

ki predstavlja model za podatkovno intenzivna opravila, ki so eksplicitno vključena v model, podatke samo berejo, ciljna funkcija pa je $\bigoplus^{\Sigma,max}$.

Tretji primer (v nadaljevanju velikokrat uporabljen) pa je

$$\left\langle implJobs, R, \bigoplus^{\Sigma,\Sigma} \right\rangle.$$

Ta model opravil eksplicitno ne vključuje, opravila podatke samo berejo, ciljna funkcija pa je $\bigoplus^{\Sigma,\Sigma}$

4.7 Analiza časovne zahtevnosti

Pri analizi časovne zahtevnosti se bomo omejili na model $\left\langle implJobs, R, \bigoplus^{\Sigma,\Sigma} \right\rangle$, ker z dokazom NP-težkosti zelo enostavnega modela pokažemo tudi enako časovno zahtevnost bolj zapletenih modelov, saj iz vsakega od bolj zapletenih lahko dobimo enostavnega tako, da postavimo določene omejitve. Drugi razlog za to izbiro je, da bomo v nadaljnjih poglavjih uporabljali predvsem ta model. Da bi dokazali NP-težkost optimizacijskega problema $\left\langle implJobs, R, \bigoplus^{\Sigma,\Sigma} \right\rangle$, bomo najprej definirali odločitveno obliko problema. Za to obliko bomo pokazali, da je NP-polna. Iz tega dokaza bo neposredno sledilo, da je optimizacijska oblika problema NP-težka.

Definicija 1 (Podvajanje)**.**
PROBLEM: *Enostaven neusmerjen graf (omrežje)* $G = \langle V, E \rangle$*, kapaciteta vozlišč* $c : V \longrightarrow \mathbb{Z}^+$*, razdalja med vozlišči* $d : V \times V \longrightarrow \mathbb{Z}^+$*, končna množica datotek* A *, velikost datotek* $s : A \longrightarrow \mathbb{Z}^+$*, celo število* $D \in \mathbb{Z}^+$*.*
VPRAŠANJE: *Ali obstaja funkcija* $f : A \longrightarrow 2^V$*, ki zadošča omejitvi (4.1 - razdelek 4.2.2), da velja*

$$\sum_{v \in V} \sum_{a \in A} cost(v, a) \leq D?$$

NP-polnost problema Podvajanje bomo dokazali s polinomsko prevedbo dobro znanega NP-polnega problema Razdelitev [44].

Definicija 2 (Razdelitev)**.**
PROBLEM: *Končna množica* B*, velikost elementov* $w(b) \in \mathbb{Z}^+$ *za vsak* $b \in B$*.*
VPRAŠANJE: *Ali obstaja podmnožica* $B' \subseteq B$*, da velja*

$$\sum_{b \in B'} w(b) = \sum_{b \in B \setminus B'} w(b)?$$

Oznaka $B \setminus B'$ predstavlja razliko množic B in B'.

Izrek 1. *Problem* Razdelitev *je NP-poln.*

Dokaz. Glej [44] □

Izrek 2. *Problem* Podvajanje *je NP-poln.*

Dokaz. Najprej lahko dokažemo, da je problem Podvajanje v razredu NP. To lahko enostavno vidimo, saj lahko vrednost ciljne funkcije izračunamo v polinomskem času.

V drugem delu dokaza moramo pokazati, da bi bil problem Razdelitev rešljiv v polinomskem času, če bi bil problem Podvajanja rešljiv v polinomskem času. To lahko naredimo tako, da sestavimo polinomsko prevedbo problema Razdelitev na problem Podvajanja.

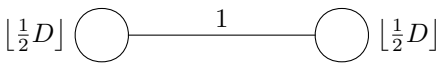
Prevedba. Naj bo dana poljubna naloga (B, w) problema Razdelitev. Enakovredna naloga problema Podvajanje je sestavljena na naslednji način:

- $D = \sum_{b \in B} w(b)$,
- $A = B$ in $s = w$,
- $G = \langle V, E \rangle$ kjer je $V = \{v_1, v_2\}$ in $E = \{\langle v_1, v_2 \rangle, \langle v_2, v_1 \rangle\}$,
- $c(v) = \left\lfloor \frac{1}{2} D \right\rfloor$ za $\forall v \in V$,
- $d(e) = 1$ za $\forall e \in E$.

Slika 4.7 prikazuje prevedbo.
Pokazati moramo, da ima tako sestavljena naloga problema Podvajanje rešitev (funkcijo *rep* z vrednostjo ciljne funkcije $\leq D$) natanko takrat ko ima rešitev tudi pripadajoča naloga Razdelitev.

- ($\Rightarrow$) Denimo, da obstaja rešitev problema Podvajanje. Tedaj obstaja funkcija *rep*, ki razdeli množico A v dve disjunktni množici $A' = \{a \in A : rep(a) = \{v_1\}\}$ in $A \setminus A' = \{a \in A : rep(a) = \{v_2\}\}$. Rešitev naloge Razdelitev lahko enostavno dobimo, če postavimo $B' = A'$.

- ($\Leftarrow$) Denimo, da obstaja rešitev problema Razdelitev. Tedaj obstaja množica B', za katero velja $\sum_{b \in B'} s(b) = \frac{D}{2}$. Rešitev naloge Podvajanje lahko dobimo, če postavimo $rep(a) = \{v_1\}$ za $\forall a \in B'$ in $rep(a) = \{v_2\}$ za $\forall a \in B \setminus B'$.



Slika 4.7: Prevedba problema RAZDELITEV na problem PODVAJANJE

□

Iz dokaza lahko izluščimo še en zanimiv rezultat. Pokažemo lahko namreč, da je problem $\left\langle implJobs, R, \bigoplus^{\Sigma,\Sigma} \right\rangle$(torej optimizacijska različica problema PODVAJANJE) neaproksimabilen. To pomeni, da če velja $P \neq NP$, potem ne obstaja polinomski ε-aproksimacijski algoritem, ki bi lahko vrnil približno rešitev vseh nalog problema $\left\langle implJobs, R, \bigoplus^{\Sigma,\Sigma} \right\rangle$.

Izrek 3. *Problem* $\left\langle implJobs, R, \bigoplus^{\Sigma,\Sigma} \right\rangle$ *je neaproksimabilen.*

Dokaz. Dokaz temelji na dejstvu, da ima ciljna funkcija pri zgornji prevedbi samo dve možni vrednosti. Ti vrednosti sta: D v primeru, ko lahko množico B razdelimo na dve množici, in ∞ ko taka razdelitev ne obstaja. Predpostavimo, da obstaja polinomski ε-aproksimacijski algoritem P za $\left\langle implJobs, R, \bigoplus^{\Sigma,\Sigma} \right\rangle$. Tedaj bi lahko rešili tudi problem RAZDELITEV v polinomskem času s sledečim algoritmom.

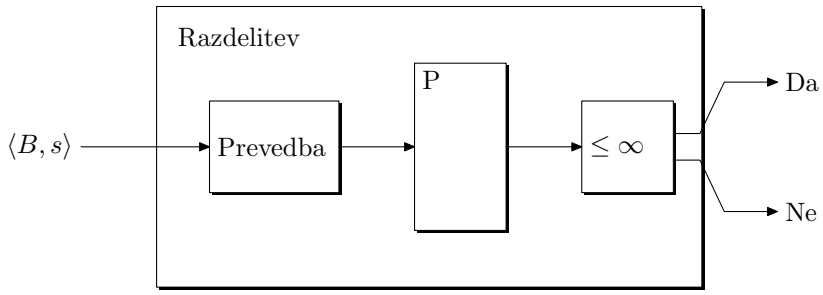
1. Najprej naredimo prevedbo naloge RAZDELITEV na problem PODVAJANJE. Nato uporabimo algoritem P in z njim dobimo rešitev ekvivalentnega problema $\left\langle implJobs, R, \bigoplus^{\Sigma,\Sigma} \right\rangle$.

2. Če je vrednost ciljne funkcije pri približni rešitvi končna, potem obstaja rešitev problema PODVAJANJE. Torej obstaja tudi rešitev originalnega problema RAZDELITEV.

3. V nasprotnem primeru, torej če je vrednost ciljne funkcije enaka ∞, pa je tudi vrednost vsake dopustne rešitve naloge $\left\langle implJobs, R, \bigoplus^{\Sigma,\Sigma} \right\rangle$ enaka ∞. V tem primeru tako naloga PODVAJANJE kot tudi naloga RAZDELITEV nimata rešitve.

□

Slika 4.8 prikazuje zgornji dokaz. Z drugimi besedami, če bi obstajal polinomski aproksimacijski algoritem za problem $\left\langle implJobs, R, \bigoplus^{\Sigma,\Sigma} \right\rangle$, bi lahko reševali tudi problem RAZDELITEV v polinomskem času.



Slika 4.8: Neaproksimabilnost problema $\left\langle implJobs, R, \bigoplus^{\Sigma,\Sigma} \right\rangle$

Peto poglavje

Preizkus modela v simulacijah

V prejšnjem poglavju smo zasnovali množico modelov, ki opisuje dobro postavitev kopij datotek v podatkovnem omrežju. Širok nabor modelov je bil definiran z namenom, da omogočimo dovolj veliko fleksibilnost in uporabnost modelov v zelo različnih sistemih in aplikacijah. V različnih okoliščinah, ki se pojavljajo v podatkovnih omrežjih, so namreč pomembni različni parametri. Načrtovalcem algoritmov pa je prepuščeno, da izberejo model, ki v danih okoliščinah najbolje opisuje sistem. Množica predstavljenih modelov je bila zasnovana na podlagi trenutnega znanja o podatkovnih omrežjih in na podlagi podobnosti problema podvajanja s problemi iz teorije razmeščanja.

V tem poglavju pa bomo predstavili način, kako pokazati, da teoretični model dobro opisuje realna omrežja. Enostavno povedano, model dobro opisuje problem podvajanja, ko ima neka postavitev kopij, ki je v modelu ovrednotena kot kakovostna, tudi v realnih sistemih za posledico izboljšanje lastnosti podatkovnega omrežja.

Model, ki ga bomo ovrednotili v tem poglavju, je $\left\langle implJobs, R, \bigoplus^{\Sigma,\Sigma} \right\rangle$. V tem modelu so podatkovno zahtevna opravila, ki so implicitno vključena v model, podatke samo berejo, ciljna funkcija za ovrednotenje postavitev podatkov pa je povprečje dostopnosti podatkov v omrežju. Izbira modela je posledica lastnosti in zahtev aplikacij, ki so trenutno najbolj aktualne. Razloge lahko strnemo v naslednjih točkah.

- Model je zasnovan za podatkovno zahtevna opravila, kar je trenutno zelo aktualna aplikacija v podatkovnih omrežjih. Model je tudi bolj uporaben, ker eksplicitno ne vključuje opravil. V realnih podatkovnih

omrežjih so namreč aplikacije, v katerih vnaprej poznamo vsa opravila, zelo redke.

- Model je enostaven. To je zelo zaželena lastnost, saj je tak problem lažje analizirati in reševati, kot bolj kompleksne modele.

- Velika večina današnjih aplikacij podatkov ne spreminja. Zato se bomo zaenkrat osredotočili le na modele, ki podatke samo berejo.

- Ciljna funkcija $\bigoplus^{\Sigma,\Sigma}$ je bila izbrana zaradi lastnosti podatkovnega omrežja, ki bi jo radi s podvajanjem dosegli. Namreč, radi bi dosegli večjo prepustnost sistema za podatkovno zahtevna opravila. Z drugimi besedami, zanima nas izboljšava povprečnega časa izvajanja opravil. Zaradi tega je tudi funkcija povprečne dostopnosti podatkov v tem primeru najboljša.

Najboljši preizkus kakovosti modela bi bila izvedli v dejanskem podatkovnem omrežju. Zaradi številnih tehnoloških omejitev pa je to zelo težko ali celo nemogoče. Razlogi so v težki dostopnosti dejanskih omrežij, predvsem pa v dejstvu, da v trenutnih omrežjih storitve, ki bi omogočale podvajanje podatkov, še niso na voljo.

Zato bomo model preizkusili v simulacijah. Simulacije pa niso vedno slabša alternativa preizkusu v realnih sistemih, ampak imajo tudi številne prednosti:

- preizkusi v simulatorjih so enostavno ponovljivi, kar omogoča boljšo preverljivost rezultatov,

- v simulatorjih imamo veliko večji nadzor nad vsemi parametri,

- v simulatorjih lahko preizkusimo več različnih sistemov, med drugim tudi sisteme, ki v realnosti še ne obstajajo.

Oglejmo si, kako bomo model preizkusili.

1. Najprej bomo izbrali nabor testnih primerov, ki najbolje predstavljajo trenutna podatkovna omrežja in aplikacije v njih.

2. Vsak testni primer bomo simulirali večkrat, vsakokrat z drugačno postavitvijo podatkov.

3. Vsako postavitev podatkov bomo ocenili s ciljno funkcijo $\bigoplus^{\Sigma,\Sigma}$.

4. V simulacijah pa bomo merili čas, ki je potreben za izvedbo vseh opravil v enem testnem primeru.

Pokazali bi radi, da boljša (manjša) vrednost ciljne funkcije pomeni večjo prepustnost sistema.

V nadaljevanju tega poglavja bomo najprej predstavili lastnosti obstoječih omrežnih simulatorjev in razloge za izbiro simulatorja OptorSim, ki smo ga uporabili pri preizkusu modela. Opisali bomo tudi izbrane realne testne primere in testne primere, ki smo jih generirali umetno, vendar z lastnostmi dejanskih velikih sistemov. Nazadnje bomo predstavili rezultate simulacij in zaključke, ki iz teh rezultatov sledijo.

5.1 Izbira simulatorja

Za preizkus modela v simulacijah smo se odločili uporabiti simulator OptorSim [117]. Ker to ni edini simulator, ki je na voljo za simulacijo podatkovnih omrežij, si najprej oglejmo še nekatere druge simulatorje in njihove lastnosti. Poleg simulatorja OptorSim, so trenutno na voljo še GridSim [15, 92], Monarc [30], ChicSim [126], SimGrid [64] in MicroGrid [85]. Tabela 5.1 povzema lastnosti teh simulatorjev.

Funkcionalnost	OptorSim	GridSim	Monarc	ChicSim	SimGrid	MicroGrid
podvajanje podatkov	✓	✓	✓	✓	✗	✗
razvrščanje opravil	✓	✓	✓	✗	✗	✓
topologija	splošna	splošna	hierarhična	hierarhična	splošna	splošna
enostavna razširljivost	✓	✓	✗	✗	✗	✗
enostavna uporaba	✓	✗	✗	✗	✗	✗
optimizacija podvajanja	✓	✗	✗	✗	✗	✗

Tabela 5.1: Seznam funkcionalnosti različnih omrežnih simulatorjev

- Podvajanje podatkov. Simulator omogoča simulacijo podvajanja podatkov in informacijskega sistema za dostop do podatkov.
- Optimizacija podvajanja. Simulator nudi mehanizme, ki omogočajo implementacijo metod za optimizacijo podvajanja podatkov.
- Razvrščanje opravil. Simulator omogoča razvrstitev opravil na druga vozlišča.
- Tip topologije. Topologija, ki jo lahko simuliramo. Lahko simuliramo hierarhično topologijo (topologija v obliki drevesa) ali pa splošno topologijo (topologija je splošen graf).

- Razširljivost. Simulator je zasnovan tako, da omogoča enostavno razširljivost. Svoje metode razvrščanja in podvajanja lahko dodajamo brez velikih sprememb celotne arhitekture sistema.

- Enostavnost uporabe. Kako hitro lahko spremenimo simulator, zaženemo simulacijo, napišemo novo metodo ipd.

Lastnosti, podane v tabeli 5.1, so tiste, ki so bile za nas najbolj pomembne pri izbiri simulatorja. Vidimo lahko, da sta po teh lastnostih simulatorja GridSim in OptorSim zelo podobna. Zelo pomembna razlika med njima pa je hitrost delovanja. Ta lastnost je tudi prevladala, saj smo morali za ovrednotenje modelov izvesti veliko število simulacij. Kot primer lahko navedemo, da v povprečju simulacija v OptorSimu traja nekaj ur, primerljiva simulacija v GridSimu pa tudi več dni.

OptorSim je bil razvit v projektu EU DataGrid [111]. Cilj tega simulatorja je študija različnih metod podvajanja. GridSim pa je bil najprej simulator računskih omrežij, šele pozneje je bila izdelana razširitev, ki omogoča simulacijo podatkovnih omrežij. Trenutno je GridSim najbolj primeren za študij mehanizmov (razdelek 3.1) za podvajanje podatkov, saj je lažje razširljiv kot OptorSim. Ker pa se v tem delu ne posvečamo mehanizmom za podvajanje, je za nas najbolj primeren OptorSim.

5.2 Opis testnih primerov

Omenili smo že, da bomo izbrali nabor testnih primerov, ki jih bomo simulirali. S pojmom testni primer mislimo vse parametre, ki natančno določajo celoten sistem in aplikacijo, ki se v njem izvaja. Testni primer je sestavljen iz:

- topologije podatkovnega omrežja,

- datotek, ki jih podvajamo,

- lastnosti vozlišč (predvsem kapacitet za hrambo podatkov) in

- podatkovno zahtevnih opravil (porazdelitev uporabe podatkov, prihod opravil).

Testne primere razdelimo v dve skupini. V prvi so testni primeri, ki temeljijo na realnih podatkovnih omrežjih in aplikacijah, ki se na njih izvajajo. Vsi realni testni primeri so povezani z velikim hadronskim trkalnikom LHC [29].

Simulirali bomo namreč podatkovna omrežja, ki so bila postavljena za analizo rezultatov eksperimentov tega pospeševalnika delcev v CERN-u. Drugo skupino pa sestavljajo testni primeri, ki smo jih generirali umetno. Zaradi pomanjkanja širšega nabora testnih primerov smo razvili generator umetnih testnih primerov. Na podlagi novejših raziskav velikih sistemov lahko veliko parametrov v podatkovnem omrežju generiramo z določenimi verjetnostnimi porazdelitvami. Tako dobimo umetne primere, ki pa imajo lastnosti realnih sistemov.

Najprej si oglejmo testne primere, ki temeljijo na realnih podatkovnih omrežjih in aplikacijah.

5.2.1 Realni testni primeri

Kot smo že prej omenili, vsi realni testni primeri temeljijo na podatkovnih omrežjih za analizo eksperimentov v velikem hadronskem trkalniku. Razlogi za tako izbiro testnih primerov so predvsem pomanjkanje velikih podatkovnih omrežij na drugih področjih. Fizika osnovnih delcev je namreč gonilo razvoja podatkovnih omrežij, aplikacije na ostalih področjih pa sledijo temu razvoju. V nadaljevanju bomo na kratko opisali podatkovna omrežja in aplikacije, ki jih bomo simulirali.

- *European Data Grid (EDG)* je prvo veliko podatkovno omrežje, ki je bilo postavljeno za analizo eksperimentov velikega hadronskega trkalnika. Vključuje 11 organizacij, predvsem univerz in inštitutov v Evropi. V omrežju se nahaja 100 različnih datotek, vsaka ima velikost 10 GB. Aplikacija, ki se izvaja na tem omrežju, je sestavljena iz šestih različnih tipov opravil. Opravila med izvajanjem potrebujejo od 2 do 57 datotek.

- CMS Data Challenge 2002. *Compact Muon Solenoid* je eden izmed eksperimentov, ki bodo potekali v velikem hadronskem trkalniku. Leta 2002 so postavili testno podatkovno omrežje, na katerem so preizkušali aplikacije za analizo rezultatov tega eksperimenta. V tem testnem omrežju je 20 vozlišč, ki se nahajajo v Evropi in v ZDA. V omrežju se nahaja 95 različnih datotek, vsaka datoteka ima velikost 10 GB. Ker gre za zelo podobno aplikacijo, kot je bila testirana v omrežju EDG, je tudi aplikacija CMS sestavljena iz 6 tipov opravil, opravila pa potrebujejo od 5 do 50 datotekami.

- *LHC Computing Grid - LCG* je podatkovno omrežje, ki vključuje 65 ustanov (vozlišč) po celem svetu. Simulirano podatkovno omrežje je

stanje tega sistema iz leta 2004. Datoteke in vrste opravil so zelo podobne kot v zgornjih dveh primerih.

- *GridPP* je podatkovno omrežje, ki je bilo postavljeno v Veliki Britaniji, prav tako z namenom, da bi analizirali eksperimente v pospeševalnikih delcev. Omrežje združuje 19 univerz v Veliki Britaniji, poleg njih pa še CERN, iz katerega prejemajo vse podatke. V tem omrežju se nahaja 600 datotek, vsaka ima velikost 10 GB. Izvaja se 15 različnih tipov opravil, vsako opravilo pa za izvajanje potrebuje od 10 do 50 datotekami.

Opisani testni primeri se med seboj razlikujejo po številu vozlišč, mrežni topologiji in velikosti prostora za hrambo podatkov. Vendar so med seboj tudi zelo podobni, predvsem zaradi tipov opravil in velikosti datotek. Razlog za to podobnost je ciljna aplikacija, ki je v vseh primerih enaka, tj. analiza rezultatov pospeševalnika delcev.

5.2.2 Generiranje umetnih testnih primerov

Ker želimo model problema podvajanja podatkov preizkusiti na širšem naboru testnih primerov, smo implementirali generator podatkovnih omrežij in aplikacij, ki se na njih izvajajo. Novejše raziskave so pokazale, da parametri velikih porazdeljenih sistemov sledijo določenim zakonitostim. Raziskave teh parametrov so pomembne predvsem za pridobivanje znanja o teh sistemih in tudi za generiranje umetnih primerkov teh sistemov.

Ker bi radi v umetno generirane testne primere podatkovnih omrežij in aplikacij vnesli čim več lastnosti realnih sistemov, smo raziskave velikih sistemov upoštevali pri načrtovanju generatorja umetnih testnih primerov.

Lastnosti, ki nas pri generiranju najbolj zanimajo, so:

- topologija omrežja,
- velikost prostora za hrambo podatkov,
- velikost in število datotek v sistemu in
- tipi opravil in število datotek, ki jih opravilo potrebuje.

Generiranje mrežnih topologij

Pri generiranju topologij, ki imajo najbolj kritične lastnosti realnih topologij, se lahko poslužimo velikega nabora modelov, ki so bili izdelani za

generiranje internetnih topologij. Za generator topologij smo uporabili BRITE [70]. BRITE je univerzalni generator topologij. Uporabljen je bil v številnih raziskavah računalniških mrež in analizah njihovih lastnosti. V generatorju BRITE lahko generiramo ploske ali hierarhične topologije.

- Ploske topologije. Za generiranje ploskih topologij so na voljo trije različni modeli:
  - Barabasi-Albert,
  - Waxmann,
  - Mali svet (angl. small-world).

  Pri topologijah realnih omrežij se najbolj pogosto uporablja model Barabasi-Albert. Dobljena topologija je t. i. *brezlestvični graf* [36] (angl. scale-free graph).

- Hierarhične topologije. Drugi tip so hierarhične topologije. Pri teh sta možna dva nivoja. Na najvišjem nivoju je ena topologija, vsako njeno vozlišče pa predstavlja npr. državo, regijo ali pa veliko organizacijo. Za vsako vozlišče se nato generira še ena topologija.

Generiranje datotek in njihovih velikosti

Veliko dela je že bilo opravljenega na področju modeliranja verjetnostnih porazdelitev velikosti datotek v različnih sistemih. V delih se pojavljata predvsem dva modela. Prvi predstavlja porazdelitev Pareto [24], drugi model pa log-naravno porazdelitev [34] . Pri porazdelitvi Pareto je verjetnost, da je velikost datoteke večja od vrednosti x, enaka

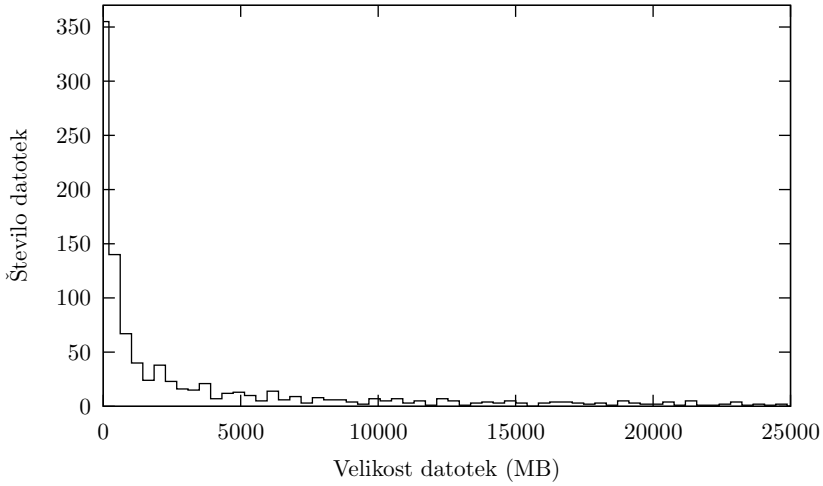
$$Pr(X >) = \left(\frac{x}{x_m}\right)^{-k},$$

za vsak $x \geq x_m$, kjer je x_m najmanjša možna velikost datoteke in k neko pozitivno število.

Druga verjetnostna porazdelitev velikosti datotek je log-naravna porazdelitev. Verjetnostna spremenljivka X je porazdeljena po tej porazdelitvi, če je spremenljivka $\log X$ porazdeljena po naravni porazdelitvi.

Slika 5.1 prikazuje primer porazdelitve velikosti datotek, ki smo jo generirali z našim generatorjem z verjetnostno porazdelitvijo Pareto. Vidimo

lahko, da je daleč največ majhnih datotek, vendar v sistemu obstajajo tudi zelo velike datoteke. Zaradi te lastnosti takim porazdelitvam pravimo tudi dolgorepe porazdelitve (angl. *long-tailed distributions*).



Slika 5.1: Porazdelitev velikosti datotek

Generiranje velikosti prostora za hrambo podatkov

Velikosti prostora za hrambo podatkov so v realnih omrežjih ponavadi odvisne od položaja posameznega vozlišča v določeni topologiji. Vozlišča, ki imajo večjo stopnjo, imajo ponavadi več diskovnega prostora. Včasih pa obstajajo tudi vozlišča, ki močno odstopajo od povprečnih kapacitet, zato bomo generirali velikosti po verjetnostni porazdelitvi Pareto, podobno kot pri velikosti datotek.

Generiranje opravil in njihovih podatkov

Za izvajanje v podatkovnih omrežjih moramo generirati množico opravil. Za vsako opravilo najprej izberemo število datotek, ki jih potrebuje, v naslednjem koraku pa določimo, katere so te datoteke. Število datotek, ki jih opravila potrebujejo, je bilo izbrano s porazdelitvijo Pareto, tako da vrednosti ne odstopajo preveč od vrednosti iz realnih testnih primerov.

Generirana testna primera

Ker je topologija zelo pomemben parameter, smo se odločili, da izberemo dva testna primera, ki sta dovolj različna, obenem pa prestavljata dva do-

volj realna primera. Prvi testni primer ima plosko, drugi pa hierarhično topologijo.

5.3 Testiranje

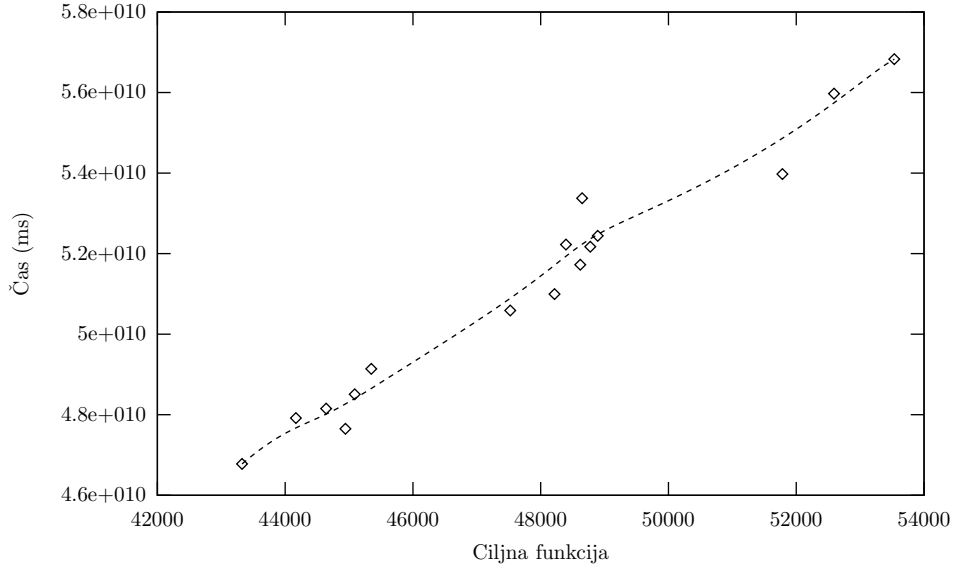
Vsak izmed zgoraj opisanih testnih primerov ima 5000 opravil, ki jih mora izvesti na podatkovnem omrežju. Za vsako datoteko smo najprej naključno postavili originalne datoteke. S tem smo zagotovili, da je vsaka datoteka vsaj enkrat postavljena na podatkovno omrežje. Nato smo vsako vozlišče zapolnili s kopijami naključno izbrane podmnožice datotek. Ko so vsa vozlišča zapolnjena z datotekami, smo izračunali vrednost ciljne funkcije pri taki postavitvi kopij. Vsak testni primer smo simulirali 20 krat, vsakokrat z drugo naključno postavitvijo datotek. Pri simulaciji smo merili celoten čas izvajanja vseh opravil. V simulatorju je malo naključnosti, zato so tudi odstopanja časa izvajanja majhna (manj kot 1 odstotek). To je tudi eden izmed razlogov, zakaj je za posamezno postavitev kopij dovolj ena simulacija. Drugi pomemben razlog za zgolj eno simulacijo pa je trajanje ene simulacije, saj lahko ena simulacija traja od nekaj ur do celega dneva.

Na slikah 5.2, 5.3, 5.4, 5.5, 5.6 in 5.7 so prikazani rezultati simulacij za posamezne testne primere. Na osi x je prikazana vrednost ciljne funkcije pri določeni postavitvi kopij, na osi y pa čas izvajanja testnega primera.

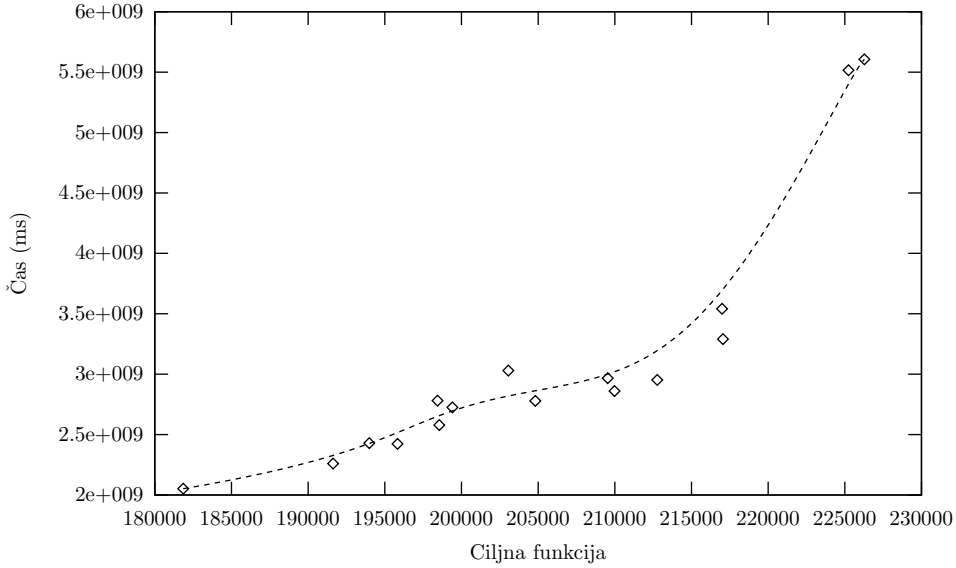
Na teh grafih lahko vidimo, da je čas izvajanja množice opravil daljši, če so podatki postavljeni manj kakovostno. Kakovost pa je opisana z vrednostjo ciljne funkcije modela $\left\langle implJobs, R, \bigoplus^{\Sigma,\Sigma} \right\rangle$.

5.4 Rezultati in zaključki

V tem poglavju smo hoteli prikazati dejansko kakovost enega izmed razvitih modelov. Model $\left\langle implJobs, R, \bigoplus^{\Sigma,\Sigma} \right\rangle$ smo izbrali zaradi njegove enostavnosti in predvsem zaradi njegove aktualnosti. Opisuje namreč sisteme in aplikacije, ki so trenutno najbolj zanimivi v podatkovnih omrežjih. Kakovost modela smo preizkusili v simulatorju OptorSim. Izbrali smo nabor testnih primerov, ki temeljijo na realnih testnih omrežjih. Zaradi pomanjkanja širšega nabora realnih testnih primerov, pa smo razvili tudi generator umetnih testnih primerov. Na podlagi novejših odkritij o lastnostih velikih računalniških sistemov smo v generator vključili tudi lastnosti, ki jih imajo taki veliki sistemi. Tako lahko rečemo, da smo se z umetno generiranimi testnimi primeri kar najbolj približali realnosti.

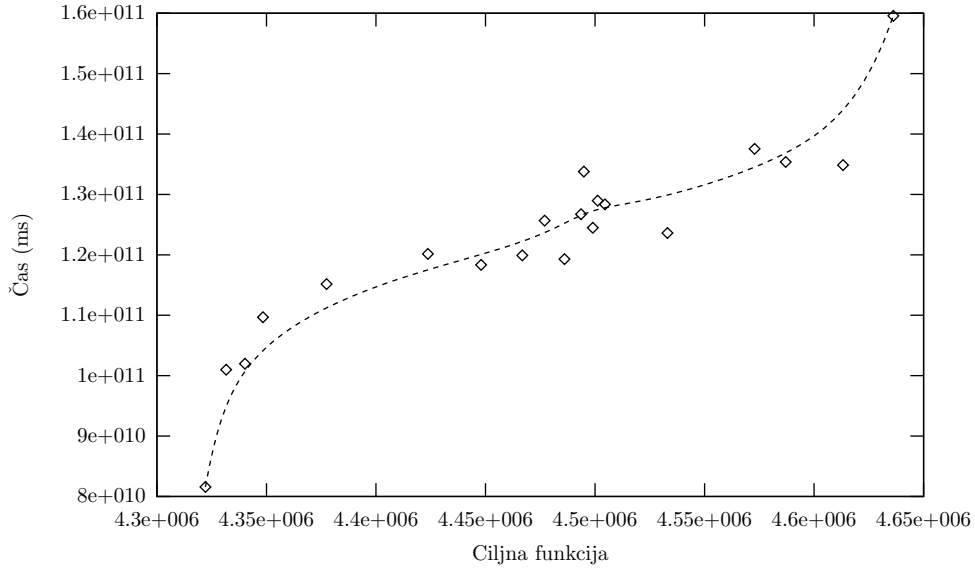


Slika 5.2: Rezultat za EDG

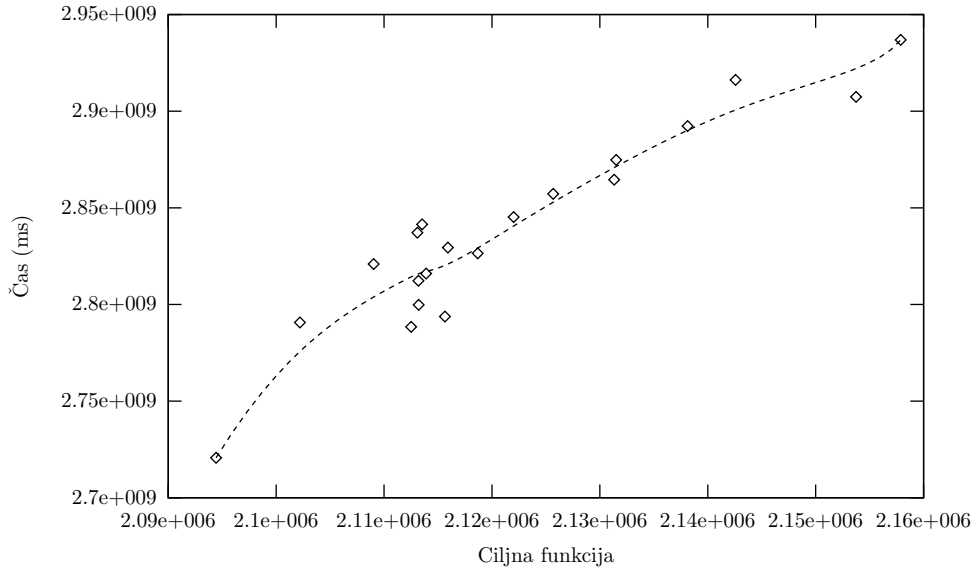


Slika 5.3: Rezultat za CMS

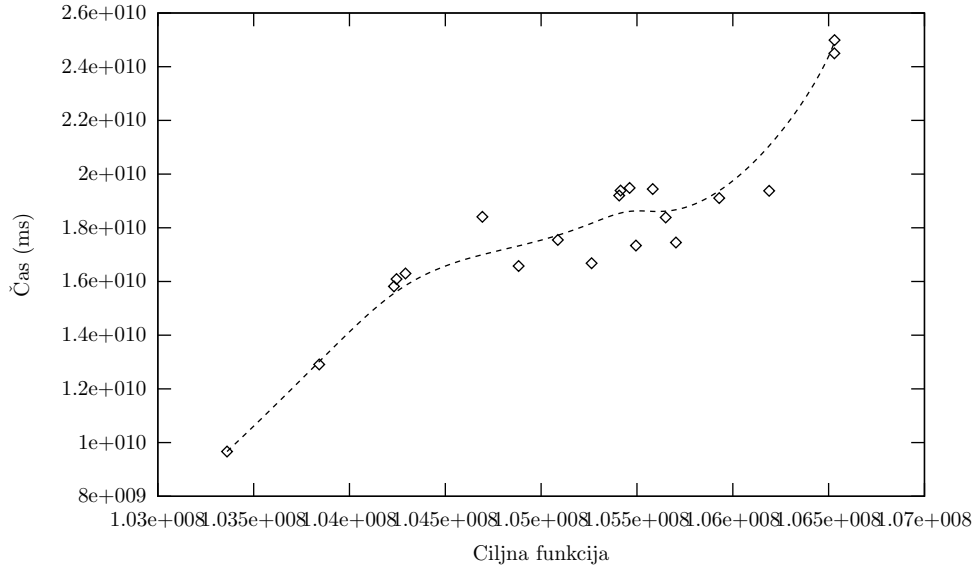
Vsak testni primer smo simulirali 20-krat, vsakokrat z drugačno postavitvijo podatkov. Simulacije so pokazale, da model dobro opisuje kakovostno postavitev podatkov. V primeru boljše (manjše) vrednosti ciljne funkcije se tudi opravila v preizkušenih testnih podatkovnih omrežjih izvajajo hitreje.



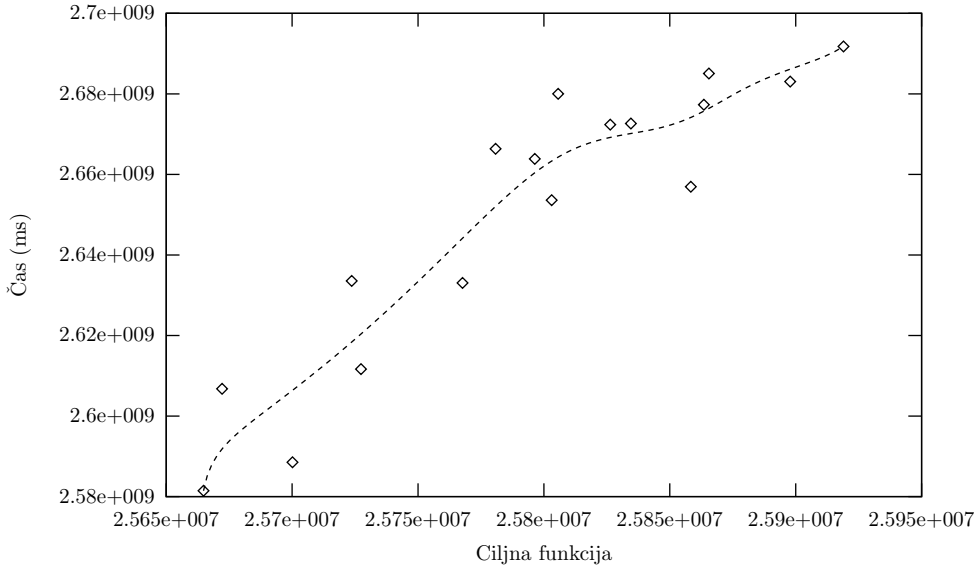
Slika 5.4: Rezultat za LCG



Slika 5.5: Rezultat za GRIDPP



Slika 5.6: Rezultat za umetni testni primer s plosko topologijo



Slika 5.7: Rezultat za umetni testni primer s hierarhično topologijo

Šesto poglavje

Načrtovanje algoritmov

Prejšnje poglavje je bilo namenjeno preizkusu modela $\langle implJobs, R, \bigoplus^{\Sigma,\Sigma} \rangle$ v simulacijah. Pokazali smo, da model dobro opisuje kakovostno postavitev podatkov v trenutno aktualnih podatkovnih omrežjih in aplikacijah, ki se tam izvajajo. Postavitve podatkov, ki smo jih preizkušali v simulacijah, smo dobili zgolj z naključnim postavljanjem kopij. Zato se bomo v tem poglavju posvetili snovanju algoritmov za iskanje čim boljših postavitev. Poiskali bi radi algoritme, ki hitro najdejo kakovostno postavitev, obenem pa so dovolj preprosti za implementacijo v porazdeljenem sistemu. Ker smo že dokazali veliko časovno zahtevnost tega modela (glej razdelek 4.7), se bomo pri načrtovanju algoritmov osredotočili na hevristične algoritme. Ogledali si bomo tri zelo različne vrste hevrističnih algoritmov, in sicer:

- požrešne algoritme,
- lokalno optimizacijo in
- genetske algoritme.

Predstavili bomo dva požrešna algoritma, od katerih je prvi zelo preprost požrešni algoritem, drugi pa temelji na znanju iz teorije razmeščanja. Za uporabo lokalne optimizacije moramo določiti relacijo sosednosti v danem problemu. V tem poglavju bomo definirali dve vrsti sosednosti, od katerih lahko vsako uporabimo za klasično lokalno optimizacijo. Obe sosednosti pa bomo tudi združili v bolj napredni metodi, imenovani *lokalna optimizacija z menjavo sosednosti* (angl. *variable neighbourhood search - VNS*). Genetski algoritmi so tretja vrsta algoritmov, ki jih bomo uporabili za reševanje

našega problema. Njihova lastnost je, da imajo veliko parametrov. Predstavili bomo nabor vrednosti parametrov, ki smo jih določili empirično, da smo dobili čim boljše rešitve.

Predstavljene algoritme bomo med seboj primerjali. Primerjavo algoritmov bi lahko izvedli z uporabo simulacij tako, da bi postavitev, ki jo vrne algoritem, simulirali in merili prepustnost podatkovnega omrežja. Ker pa smo že dokazali odvisnost med prepustnostjo podatkovnega omrežja in vrednostjo ciljne funkcije, algoritmov ni potrebno primerjati v simulacijah; zadoščala bo primerjava vrednosti ciljnih funkcij njihovih postavitev. Prednost takega testiranja je veliko večja enostavnost in hitrost. Simulacije so namreč časovno zelo zahtevne. Za primerjanje algoritmov smo generirali 39 testnih primerov. Da bi v testnih primerih ohranili ključne lastnosti realnih sistemov, smo jih generirali z enakimi verjetnostnimi porazdelitvami kot umetna testna primera za preizkus modela v poglavju 5.

Na koncu poglavja bomo predstavili še pristope k natančnemu reševanju problema podvajanja podatkov. Bolj natančno predstavimo celoštevilski program, ki opisuje model $\left\langle implJobs, R, \bigoplus^{\Sigma,\Sigma} \right\rangle$, in omejitve, ki lahko dovolj poenostavijo problem, da postane rešljiv v polinomskem času.

6.1 Požrešni algoritmi

Požrešni algoritmi so v praksi velikokrat uporabljeni zaradi nekaterih dobrih lastnosti. Prva dobra lastnost je njihova enostavnost, druga pa hitrost. Ne preiskujejo namreč velikega prostora rešitev, kot je to pogosto pri večini hevurističnih algoritmov, ampak z enostavnim kriterijem težijo k dobri rešitvi.

6.1.1 Enostaven požrešni algoritem

Glavna zamisel enostavnega požrešnega algoritma je, da morajo imeti večje datoteke več kopij v sistemu. Prenos večjih datotek po omrežju traja namreč več časa, zato te datoteke prispevajo največ h končni vrednosti ciljne funkcije.

Z enostavnim požrešnim algoritmom bomo vozlišča zapolnili eno za drugim. Ali bo neka datoteka postavljena na določeno vozlišče pa je odvisno od njene velikosti in od števila njenih kopij, ki se že nahajajo v omrežju. Vsaki datoteki dodamo še en koeficient, ki bo določal kolikokrat je datoteka že bila postavljena na omrežje v primerjavi z drugimi datotekami. Koeficient določimo s funkcijo

$$\text{koef} : A \longrightarrow \mathbb{R}^{+}.$$

Vsakokrat, ko je datoteka dodana na vozlišče v omrežju, se koeficient razpolovi. Če pa datoteka v trenutnem zapolnjevanju vozlišča ni bila dodana (vozlišče smo že zapolnili z drugimi datotekami), se koeficient podvoji. Pred vsakim zapolnjevanjem vozlišča vse datoteke uredimo po padajoči vrednosti $koef(a)s(a) \quad (a \in A)$.

Algoritem 1: Enostaven požrešni algoritem

Vhod: primerek problema $\left\langle implJobs, R, \bigoplus^{\Sigma,\Sigma} \right\rangle$.
Izhod: funkcija podvajanja rep
`Init()`;
$koef(a) := 1.0 \qquad \forall a \in A$;
forall $v \in V$ **do**
  $Urejeni[] :=$ `UrediDatoteke(`A`)`; /* padajoče po velikosti */
  $i := 0$;
  while $\left(\sum_{\{a|v \in rep(a)\}} s(a)\right) \leq c(v)$ **do** /* dokler je prostor */
    $rep(Urejeni[i]) := rep(Urejeni[i]) \cup \{v\}$; /* dodajamo */
    $koef(Urejeni[i]) := \frac{koef(Urejeni[i])}{2}$;
    $i++$;
  end
  while $i < |A|$ **do** /* datoteke, ki niso bile dodane */
    $koef(Urejeni[i]) := 2 * koef(Urejeni[i])$;
    $i++$;
  end
end

Pomanjkljivost tega pristopa je, da lahko vrne postavitev, ki nekaterih datotek sploh ne postavijo na podatkovno omrežje. Da bi se temu izognili, izvedemo dodatno proceduro za inicializacijo, ki naključno postavi eno kopijo vsake datoteke na omrežje. Procedura za postavljanje naključne kopije vsakega objekta je prikazana kot Algoritem 2.

Slabost enostavnega požrešnega algoritma je, da pri postavljanju kopij upošteva samo velikost datotek in število kopij, ki so že postavljene v sistemu, ne upošteva pa razdalj med vozlišči, kamor postavlja kopije. Da bi popravili to pomanjkljivost, smo razvili izboljšano različico Algoritma 1.

6.1.2 Izboljšan požrešni algoritem

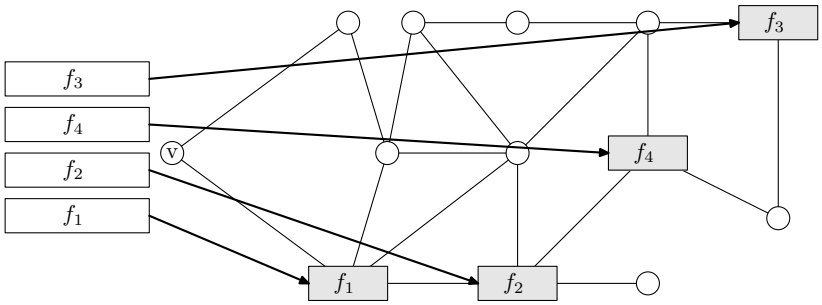
Za izboljšanje enostavnega požrešnega algoritma uporabimo zamisel algoritma iz teorije razmeščanja. Problem k-center se zelo pogosto rešuje z

Algoritem 2: Init(): dodatna funkcija za inicializacijo

Vhod: primerek problema $\left\langle implJobs, R, \bigoplus^{\Sigma,\Sigma} \right\rangle$.
Izhod: funkcija podvajanja rep
$rep(a) = \{\} \quad \forall a \in A;$
$i := 0;$
forall $a \in A$ **do**
  $v :=$ `NakljucnoVozlisce();` /* nakljucno vozlisce z dovolj veliko kapaciteto */
  $rep(a) := rep(a) \cup \{v\}$
end

uporabo t. i. Gonzalezovega algoritma [46]. Ideja tega algoritma je naslednja: dobro vozlišče za postavitev ponudnika je tisto vozlišče, ki je trenutno najbolj oddaljeno od že postavljenih ponudnikov. Začnemo torej z enim naključno postavljenim ponudnikom, nato pa požrešno dodajamo $k-1$ ponudnikov glede na prej omenjeni kriterij.

Podobno idejo uporabimo tudi pri požrešnem algoritmu za podvajanje. Enostaven požrešni algoritem spremenimo tako, da namesto urejanja datotek po velikosti, uporabimo urejanje po razdalji od trenutnega vozlišča do najbližje že postavljene kopije. Datoteke, ki so najbolj oddaljene od svojih kopij, so podvojene na trenutno vozlišče. Slika 6.1 kaže primer takega urejanja. Kopije datotek f_1, f_2, f_3 in f_4 so že postavljene v omrežju. Na vozlišču v pa uredimo datoteke glede na razdaljo do najbližje kopije. Poudariti moramo, da je razdalja do kopij utežena z velikostjo datoteke. S tem dosežemo podoben učinek kot pri enostavnem požrešnem algoritmu, vendar je pri tem upoštevana tudi topologija omrežja. Večje datoteke so še vedno postavljene na več vozlišč, vendar ne na poljubna vozlišča, pač pa na vozlišča, ki so med seboj bolj oddaljena.



Slika 6.1: Urejanje glede na razdaljo do najbližje kopije

Postopek je podrobneje opisan v algoritmu 3.

Algoritem 3: Izboljšan požrešni algoritem

Vhod: primerek problema $\left\langle implJobs, R, \bigoplus^{\Sigma,\Sigma} \right\rangle$.
Izhod: funkcija podvajanja rep
`Init();`
$i := 0$;
forall $v \in V$ **do**
  $Urejeni[] :=$ `UrediPoRazdalji(`A`);` /* padajoče */
  while $\left(\sum_{\{a|v \in rep(a)\}} s(a)\right) \leq c(v)$ **do**
    $rep(Urejeni[i]) := rep(Urejeni[i]) \cup \{v\}$;
    $i++$;
  end
end

6.2 Lokalna optimizacija z menjavo sosednosti

Poleg požrešnih algoritmov je pogost pristop k načrtovanju hevrističnih algoritmov lokalna optimizacija. Za enolično definicijo algoritma lokalne optimizacije je potrebno opisati:

- relacijo sosednosti in
- način pregledovanja soseščine.

Z definicijo relacije sosednosti opišemo, katere rešitve optimizacijskega problema so sosedne. Najpogosteje je sosednost opisana s postopkom, ki iz ene rešitve zgradi drugo (sosedno). Za pregledovanje sosednosti sta na voljo dve možnosti. Prva je, da algoritem kot boljšo rešitev iz soseščine vzame kar prvo rešitev, ki ima manjšo vrednost ciljne funkcije. Druga možnost pa je, da preišče celotno soseščino in vzame najboljšo rešitev. Pri reševanju našega problema bomo uporabili drugi način, saj se je na izbranem naboru izkazal za najboljšega.

Za problem podvajanja podatkov smo definirali dve relaciji sosednosti, ki bosta predstavljeni v nadaljevanju. Vsako lahko uporabimo za navadno lokalno optimizacijo. Obe sosednosti pa združimo v eni metodi, poimenovani lokalna optimizacija z menjavo sosednosti [71]. Ta algoritem je izboljšava

lokalne optimizacije, ki uporablja več sosednosti. Algoritem deluje na naslednji način: ko algoritem izvaja lokalno optimizacijo z eno sosednostjo in se ujame v lokalni minimum, zamenja sosednost in nadaljuje lokalno optimizacijo z novo sosednostjo. Sosednosti se menjata, dokler tako menjanje še izboljšuje trenutno rešitev. Obstaja veliko različic lokalne optimizacije z menjavo sosednosti, v splošnem pa metoda ni omejena le na 2 sosednosti, temveč jih lahko uporabi poljubno mnogo. Algoritem 4 podaja podrobnejši opis lokalne optimizacije z menjavo sosednosti.

Algoritem 4: Lokalna optimizacija z menjavo sosednosti

Vhod: primerek problema $\left\langle implJobs, R, \bigoplus^{\Sigma,\Sigma} \right\rangle$.
Izhod: funkcija podvajanja rep
$tmpR$:=`NakljucnaPostavitev()`; /* zacetna resitev */
repeat
  $rep := tmpR$;
  $tmpR$:= `PrvaSosednost(`$tmpR$`)`; /* lokalna opt. s prvo sosednostjo */
  $tmpR$:= `DrugaSosednost(`$tmpR$`)`; /* lokalna opt. z drugo sosednostjo */
until $rep = tmpR$;

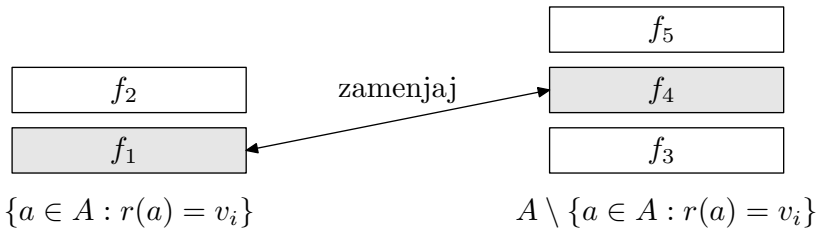
6.2.1 Definicija sosednosti

Relacijo sosednosti bomo opisali kot postopek, ki iz dane rešitve zgradi sosedno rešitev.

Prva sosednost. Dani rešitvi (funkciji podvajanja rep) so sosedne tiste rešitve, ki jih dobimo na naslednji način:

- izberemo vozlišče v,
- izberemo datoteko a_1, ki je postavljena na vozlišče v ($v \in rep(a_1)$),
- izberemo datoteko $a_2 \in A \setminus \{a | v \in rep(a)\}$,
- namesto datoteke a_1 na vozlišče v postavimo kopijo datoteke a_2 (če kapaciteta vozlišča to dopušča).

Slika 6.2 prikazuje primer zamenjave datotek. Pri lokalni optimizaciji je pomembna informacija velikost soseščine, saj je to eden izmed pomembnih parametrov, ki določa časovno zahtevnost algoritma. Zgoraj predstavljena soseščina je velika $O(|V||A|^2)$.

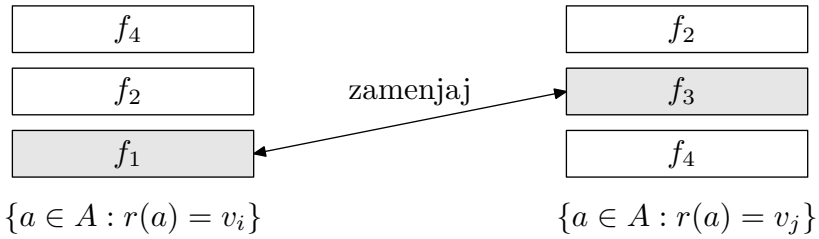


Slika 6.2: Prva sosednost

DRUGA SOSEDNOST Sosedne rešitve glede na drugo relacijo sosednosti dobimo na naslednji način:

- izberemo vozlišči v_i in v_j,
- izberemo datoteko a, katere kopija se nahaja na vozlišču v_i,
- izberemo datoteko b, katere kopija se nahaja na vozlišču v_j,
- kopiji teh dveh datotek zamenjamo (če kapaciteti vozlišč to dopuščata).

Velikost druge soseščine je $O(|V|^2|A|^2)$. Primer zamenjave datotek med dvema vozliščema prikazuje slika 6.3.



Slika 6.3: Druga sosednost

6.3 GENETSKI ALGORITMI

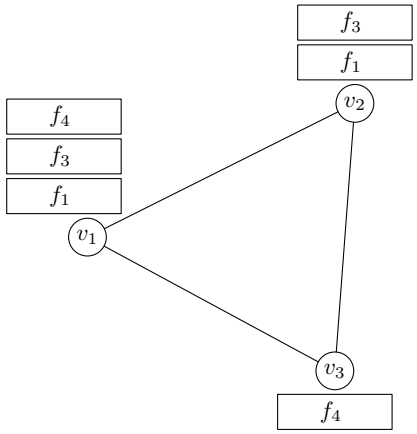
V zadnjem desetletju so se med hevristikami zelo uveljavili genetski algoritmi. Zato smo problem podvajanja reševali tudi na ta način. Genetski algoritmi so prilagodljive metode, ki jih uporabljamo za reševanje iskalnih in optimizacijskih problemov. Temeljijo na genetskem procesu bioloških organizmov, saj se zgledujejo po evoluciji v naravi, kjer se populacija neke vrste skozi generacije razvija po načelu naravnega izbora in preživetja uspešnejšega. Za definicijo genetskega algoritma moramo najprej sestaviti kromosom, ki predstavlja problem, nato pa ustrezno izbrati genetske operatorje in ostale parametre.

6.3.1 Definicija kromosoma

V genetskih algoritmih je rešitev predstavljena z dvojiškim nizom. Zato moramo pri uporabi teh algoritmov rešitve najprej zakodirati v takšen niz.

Postavitev kopij v podatkovnem omrežju smo zakodirali na naslednji način: kromosom ima $|V||A|$ dvojiških genov. Vrednost gena na k-ti poziciji pove, ali se datoteka ($k \mod |A| + 1$) nahaja na vozlišču (k div $|A| + 1$). Primer kromosoma je na sliki 6.4, predstavlja pa rešitev problema, ki je na sliki 6.5.

Slika 6.4: Primer kromosoma



Slika 6.5: Rešitev, predstavljena s kromosomom iz slike 6.4

6.3.2 Ostali parametri

Algoritem 5 je tipičen genetski algoritem, ki pa smo ga ustrezno prilagodili za problem podvajanja podatkov. Za vsako funkcijo, ki je omenjena v tem algoritmu, obstaja veliko različnih možnosti. V nadaljevanju bomo opisali, katere možnosti smo izbrali. Izbrani nabor parametrov je bil določen empirično. Postavljen je tako, da vrača čim boljše rešitve.

- Velikost populacije smo izbrali kot kompromis med hitrostjo konvergiranja in hitrostjo delovanja. Vzeli smo populacijo z 200 osebki.

Algoritem 5: Izvajanje genetskega algoritma

Vhod: primerek problema $\left\langle implJobs, R, \bigoplus^{\Sigma,\Sigma} \right\rangle$.
Izhod: funkcija podvajanja *rep*
populacija :=`Generiraj populacijo()`;
`OceniPopulacijo(`*populacija*`)`;
$i := 0$;
while $i \leq 2000$ **do**
  novaPopulacija :=`Selekcija(`*populacija*`)`;
  potomci := `Krizanje(`*novaPopulacija*`)`;
  potomci := `Mutacija(`*potomci*`)`;
  populacija := `IzberiNajboljse(`*potomci*, *novaPopulacija*`)`;
  $i++$;
end
rep :=`NajboljsaResitev(`*populacija*`)`;

- Verjetnost mutacije. Verjetnost mutacije je 0.01.

- Križanje. Izbrali smo najbolj običajen tip križanja, tj. enomestno oz. enostavno križanje, kjer vzamemo dva kromosoma, izberemo mesto križanja in vse gene od tega mesta dalje zamenjamo z geni drugega kromosoma.

- Selekcija. Za selekcijo smo izbrali turnirsko selekcijo. Ta izvede "tekmovanje" oziroma "turnir", na katerem med q osebki deterministično izberemo najboljšega. Izvedba turnirske selekcije je zelo preprosta, saj ne zahteva nobenega urejanja. Izbor q osebkov, ki vstopijo v turnir, je odvisen od implementacije. Najpogostejša je naključna izbira vseh q osebkov, ki smo jo tudi mi uporabili. Za ta tip selekcije smo se odločili, ker zagotavlja najhitrejšo konvergenco, ko smo že blizu rešitve. Selekcija z ruleto se v tem primeru izkaže precej slabše.

6.4 Primerjava algoritmov

Ker smo v poglavju 5 pokazali, da model dobro opisuje kakovostno postavitev podatkov, lahko namesto v simulatorju algoritme primerjamo z reševanjem primerkov optimizacijskih problemov. Za tako primerjavo smo generirali nabor različnih primerkov, ki lahko služijo tudi kot referenčna množica problemov za razvoj algoritmov podvajanja.

Podobno kot smo to storili v primeru generiranja umetnih testnih primerov pri ovrednotenju modelov, je tudi knjižnica primerkov optimizacijskih problemov generirana s ciljem približati se realnim lastnostim sistema. Generirali smo množico različnih testnih primerov. Vseh testnih primerov je 39, najmanjši med njimi ima 20 vozlišč, na katera postavljamo po 5 datotek, največji pa 500 vozlišč, na katera postavljamo 100 datotek.

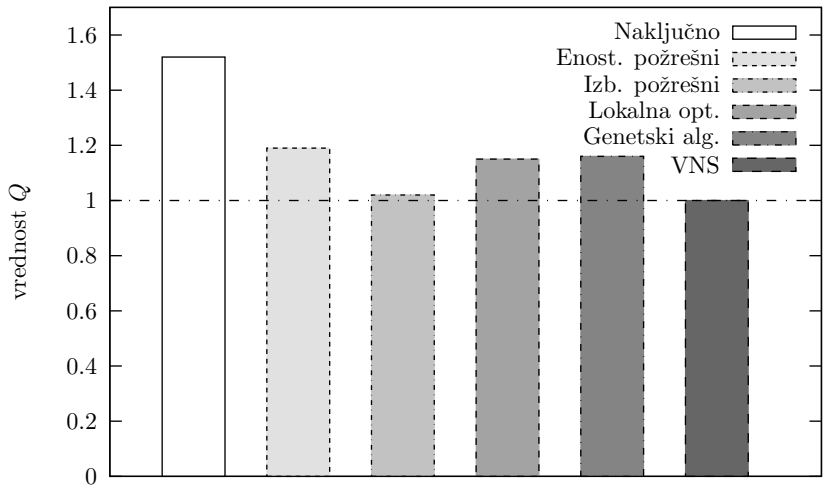
Testirali smo zgoraj opisane algoritme in naključno generirane rešitve. Algoritmi, ki smo jih preizkusili so sledeči:

- naključna rešitev,
- enostaven požrešni algoritem,
- izboljšan požrešni algoritem,
- enostavna lokalna optimizacija (prva sosednost),
- lokalna optimizacija z menjavo sosednosti (VNS) in
- genetski algoritem.

Na sliki 6.6 so rezultati primerjave naštetih algoritmov. Prikazano je povprečno razmerje med rešitvijo, dobljeno z nekim algoritmom, in najboljšo rešitvijo testnega primera. To metriko lahko bolj formalno opišemo na naslednji način. Naj bo T množica testnih primerov, $Alg(t)$ naj bo vrednost ciljne funkcije rešitve, ki jo vrne algoritem Alg pri testnem primeru $t \in T$. Vrednost $M(t)$ naj bo najboljša vrednost ciljne funkcije, dobljena s katerimkoli algoritmom. Vrednost, prikazana na sliki 6.6 za algoritem Alg, je tedaj

$$Q = \frac{1}{|T|} \sum_{t \in T} \frac{Alg(t)}{M(t)}.$$

Vidimo lahko, da so naključne rešitve v povprečju za 50 % slabše od najboljših rešitev. Izmed požrešnih algoritmov je pričakovano najboljši izboljšan požrešni algoritem. Lokalna optimizacija s prvo sosednostjo je slabša, prav tako se je genetski algoritem pokazal za slabšega od izboljšanega požrešnega algoritma in tudi od lokalne optimizacije z menjavo sosednosti. Slednja se izkaže za najboljšo, saj v vseh testnih primerih vrne najboljšo rešitev. Slabost tega algoritma pa je njegov čas izvajanja. Izboljšan požrešni algoritem se je zelo približal rešitvam, ki jih vrača lokalna optimizacija z menjavo sosednosti. Poleg tega pa je požrešni algoritem, zaradi svoje enostavnosti, zelo hiter v primerjavi z drugimi algoritmi. V posameznih primerih



Slika 6.6: Primerjava rezultatov algoritmov

rešitve odstopajo do 10 odstotkov, vendar je v vseh primerih vrstni red algoritmov povsem enak, razen v nekaj primerih genetski algoritem in lokalna optimizacija zamenjata mesti.

6.5 Natančno reševanje

V poglavju 4 smo pokazali, da je optimizacijski problem podvajanja podatkov NP-težek. Iz tega rezultata sledi, da je iskanje optimalne rešitve tega problema v splošnem zelo težka naloga. Poleg tega je splošna oblika problema tudi neaproksimabilna (če $P \neq NP$). Zato lahko trdimo, da za problem podvajanja v splošnem ne moremo najti zagotovljeno kakovostne rešitve v razumnem času.

6.5.1 Celoštevilsko programiranje

Celoštevilsko programiranje pa je kljub temu zelo pogost pristop k natančnemu reševanju podobno težkih problemov. Razvitih je bilo zelo veliko metod, ki omogočajo obvladovanje velikih primerkov NP-težkih optimizacijskih problemov [99]. V nadaljevanju podajamo prevedbo problema podvajanja na problem celoštevilskega programiranja.

Uporabljene bodo sledeče odločitvene spremenljivke:

- x_{ik} ima vrednost 1, če je kopija datoteke k postavljena na vozlišče i, in 0 v nasprotnem primeru.

- y_{ijk} ima vrednost 1, če vozlišče j prenese kopijo datoteke k z vozlišča i, in 0 v nasprotnem primeru.

Model opišemo na sledeči način.
PROBLEM: s_k (velikost datoteke k), d_{ij} (razdalja med vozliščema i in j) in c_i (kapaciteta vozlišča i).
CILJ: Minimiziraj vsoto

$$\sum_i \sum_j \sum_k s_k d_{ij} y_{ijk},$$

z naslednjimi omejitvami:

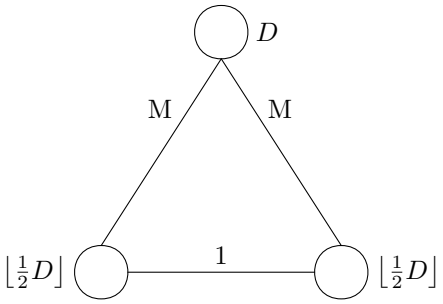
$$\begin{aligned}
\sum_k x_{ik} s_k &\leq c_i, & \forall i & \quad (6.1)\\
\sum_i y_{ijk} &= 1, & \forall j,k & \quad (6.2)\\
x_{ik} - y_{ijk} &\geq 0, & \forall i,j,k & \quad (6.3)\\
x_{ik} &\in \{0,1\}, & \forall i,k & \quad (6.4)\\
y_{ijk} &\in \{0,1\}, & \forall i,j,k. & \quad (6.5)
\end{aligned}$$

Neenakost (6.1) zahteva spoštovanje omejene kapacitete vozlišč. Enakost (6.2) določa, da za vsako vozlišče obstaja natanko eno vozlišče, s katerega prenesemo določen podatek. Neenakost (6.3) predpisuje, da je lahko datoteka prenesena samo z vozlišč, na katerih se nahajajo njene kopije. Zadnji dve omejitvi sta enostavni celoštevilski omejitvi za odločitvene spremenljivke.

Model, opisan s takim celoštevilskim programom, pa ni povsem ekvivalenten modelu $\left\langle implJobs, R, \bigoplus^{\Sigma,\Sigma} \right\rangle$. Pri prvotnem modelu so v množico dopustnih rešitev vključene tudi take rešitve, ki kopij nekaterih datotek ne postavijo na nobeno vozlišče. Take rešitve imajo neskončno vrednost ciljne funkcije.

Zaradi poenostavitve celoštevilskega programa, take rešitve niso več dopustne. Ta omejitev velja zaradi enakosti 6.2, ki zahteva, da ima vsako vozlišče dostop do natanko ene kopije vsake datoteke. Modela bi lahko naredili povsem enakovredna, če bi celoštevilskemu programu dodali novo vozlišče z dovolj veliko kapaciteto. Vendar bi to po nepotrebnem naredilo model bolj nepregleden. Rešitve, ki kopij nekaterih datotek sploh ne postavijo na omrežje, so namreč v praksi povsem neuporabne, zato celoštevilski program opisuje vse postavitve, ki nas pri tem problemu zanimajo.

Zaradi dodatne omejitve pa se postavlja vprašanje časovne zahtevnosti takega modela, saj bi lahko manjša množica dopustnih rešitev omogočila



Slika 6.7: Prevedba spremenjene oblike modela

polinomsko reševanje problema. Izkaže se, da je problem kljub dodatni omejitvi še vedno enako časovno zahteven kot prvotni model. NP-težkost omejenega modela dokažemo tako, da prevedbi iz izreka 2 dodamo novo vozlišče s kapaciteto D in razdaljo M do obeh obstoječih vozlišč (kjer je M dovolj veliko število). Slika 6.7 prikazuje novo prevedbo. V tem primeru je vedno mogoče postaviti na omrežje vsaj eno kopijo vsake datoteke. Preostanek dokaza je zelo podoben dokazu izreka 2.

6.5.2 Poenostavitve

Poleg natančnega reševanja s splošnimi metodami lahko večje znanje o problemu pridobimo tudi s poenostavljanjem problema. S poenostavitvami pogosto zmanjšamo časovno zahtevnost problema in včasih najdemo celo natančne polinomske algoritme. Take algoritme za poenostavljene probleme lahko nato uporabljamo tudi za hevristično reševanje splošnega problema.

V splošnem problemu so vse količine neomejene. V veliko problemih je ravno to vzrok za veliko časovno zahtevnost. V našem modelu so količine, ki jih lahko omejimo, sledeče:

- število datotek $|A|$ in njihova velikost $s(a)$,
- razdalje $d(v_1, v_2)$ med vozlišči,
- kapacitete $c(v)$ vozlišča.

Primer 1. Največjo poenostavitev imamo, ko je $A = \{a_1\}$. V tem primeru lahko natančno rešitev dobimo v linearnem času. Datoteka a_1 je podvojena na vsa vozlišča, ki imajo dovolj veliko kapaciteto. Ta rešitev je seveda optimalna.

PRIMER 2. Če je $A = \{a_1, a_2\}$, $s(a_1) = s(a_2) = 1$ in so vse razdalje in kapacitete enake 1, torej $\forall v \in V\ c(v) = 1$ in $\forall e \in E\ d(e) = 1$, potem je problem rešljiv v polinomskem času. Spodnji algoritem razdeli množico $|V|$ v dve disjunktni množici S_1 in S_2. Kopije datoteke a_1 so postavljene na vozlišča v množici $S_1 \subseteq V$, datoteka a_2 pa na vozlišča v množici S_2. Množica

Algoritem 6: Podvajanje dveh datotek

Vhod: primerek problema $\left\langle implJobs, R, \bigoplus^{\Sigma,\Sigma} \right\rangle$.
Izhod: funkcija podvajanja *rep*
$S_1 := \emptyset$; $S_2 := \emptyset$;
$S_1 :=$ `MaksimalnaNeodvisnaMnozica(`G`)`;
$S_2 := V \setminus S_1$;
$rep(a_1) := S_1$;
$rep(a_2) := S_2$;

S_1 postane preprosto maksimalna neodvisna množica grafa G, množica S_2 pa je preostanek množice V.

Izrek 4. *Algoritem 6 vrne optimalno rešitev v polinomskem času.*

Dokaz. Enostavno je pokazati, da je algoritem polinomski. Enostaven požrešni algoritem namreč vrne maksimalno neodvisno množico v polinomskem času. Spodnja meja vrednosti ciljne funkcije je V. Spodnjo mejo dosežemo, ko ima vsako vozlišče, kamor je postavljena datoteka a_1, sosedno vozlišče, na katerem je postavljena datoteka a_2. Enako velja tudi za vozlišča, kjer je datoteka a_2. Za vsak $v \in V$ se ciljna funkcija poveča za 1. Algoritem 6 vrne rešitev, ki doseže to spodnjo mejo. Zato je rešitev optimalna. □

Sedmo poglavje

Porazdeljena metoda podvajanja

V prejšnjem poglavju smo predstavili klasične algoritme za problem podvajanja podatkov (natančneje modela $\left\langle implJobs, R, \bigoplus^{\Sigma,\Sigma} \right\rangle$). Zasnovane algoritme smo primerjali med seboj na umetno generirani množici problemov.

Naslednji korak je uporaba zasnovanih algoritmov v podatkovnih omrežjih. V poglavju 3 smo poimenovali implementacijo algoritma v omrežju metoda za podvajanje. To poimenovanje bomo uporabili tudi v nadaljevanju, saj iščemo porazdeljeno metodo, ki je prilagojena za izvajanje v dejanskem omrežju. Prav tako smo v tretjem poglavju predstavili razvrstitev metod podvajanja. Med drugim smo metode razdelili glede na porazdeljenost na

- centralno podvajanje in
- porazdeljeno podvajanje,

glede na dinamičnost pa na

- statične in
- dinamične metode podvajanja.

Algoritme, predstavljene v prejšnjem poglavju, najbolj enostavno preslikamo v centralne in statične metode podvajanja. Pri centralnih metodah je za postavitev kopij odgovorno eno samo vozlišče. To vozlišče zbira podatke o stanju sistema. Ko pridobi dovolj podatkov, izvede algoritem za podvajanje (npr. enega od algoritmov iz prejšnjega poglavja) in glede na dobljeno postavitev ustvari kopije na ustreznih vozliščih. Centralno statično metodo

lahko enostavno spremenimo v dinamično tako, da definiramo pravila, ki določajo, kdaj se bo sprožil algoritem. Ta pravila so lahko povezana s stanjem povezav med vozlišči, povpraševanjem, spremembo procesorske moči, itd. Na ta način dobimo nove postavitve kopij, ki se prilagodijo spremembam v sistemu.

Centralno podvajanje pa ima dve bistveni pomanjkljivosti.

- Prva slabost je slaba odpornost proti napakam. Če izpade vozlišče, ki je zadolženo za podvajanje, celotno omrežje ne dobiva več novejših informacij o kakovostni postavitvi kopij.

- Druga slabost, ki je v podatkovnih omrežjih še pomembnejša, pa je zelo slaba razširljivost. V majhnih omrežjih je centralni pristop dober, v večjih omrežjih pa postane vsak centralni nadzor ozko grlo sistema.

Porazdeljene metode podvajanja so zaradi boljše odpornosti proti napakam in lažje razširljivosti boljša rešitev za podvajanje podatkov. Na žalost pa se klasični zaporedni algoritmi težko neposredno preslikajo v porazdeljene metode podvajanja.

Algoritmi v prejšnjem poglavju so bili izbrani tudi zaradi možnosti porazdeljene implementacije. Lokalno optimizacijo lahko implementiramo v porazdeljenem sistemu, prav tako obstajajo načini za porazdelitev genetskih algoritmov. V primerjavi na umetnih primerkih problema pa se je zelo dobro izkazal tudi izboljšan požrešni algoritem. Njegovi prednosti pred lokalno optimizacijo z menjavo sosednosti sta enostavnost in hitrost delovanja.

V tem poglavju bomo zato predstavili porazdeljeno dinamično metodo podvajanja, ki temelji na izboljšanem požrešnem algoritmu. Najprej si bomo podrobneje ogledali arhitekturo in metode, ki so bile na tej arhitekturi že preizkušene. Nato bomo predstavili novo metodo podvajanja, ki temelji na enostavnem pravilu izboljšanega požrešnega algoritma. Že znane metode bomo primerjali z novo metodo v simulacijah testnih primerov, ki so bili že uporabljeni v poglavju 5 za ovrednotenje kakovosti modela. Za konec bomo izpostavili slabosti trenutne arhitekture podatkovnega omrežja in predlagali nekatere možne izboljšave.

7.1 Opis arhitekture

Najprej moramo opisati arhitekturo, v katero bomo postavili porazdeljeno metodo. Oris arhitekture podatkovnega omrežja smo predstavili že v poglavju 2. Na tem mestu pa bomo podrobneje opisali komponente, ki so najbolj pomembne za podvajanje podatkov in izvajanje podatkovnih opravil. Opisana arhitektura je nastala kot del projekta EU DataGrid, trenutno pa je implementirana le v simulatorju OptorSim. Opis arhitekture zato temelji na implementaciji tega simulatorja.

Komponente. Na vsakem vozlišču se nahaja komponenta za upravljanje s podatki, imenovana upravnik kopij. Na vozlišču sta tudi dve komponenti, ki predstavljata vmesnik do fizičnih naprav, imenovani *računski element* (dostop do računske moči vozlišča) in *podatkovni vir* (dostop do diskov na vozlišču). V omrežju obstaja centralna komponenta, ki skrbi za razvrščanje opravil, imenovana *upravnik virov*.

Razvrščanje opravil. Podatkovno zahtevna opravila so sestavljena iz seznama datotek, ki jih mora vozlišče imeti na voljo lokalno, da se lahko določeno opravilo izvede. Uporabnik opravilo pošlje upravniku virov, in ta se na podlagi trenutnega stanja sistema odloči, kateremu vozlišču bo dodelil opravilo. V OptorSimu so implementirana štiri pravila za razvrščanje:

1. Naključno razvrščanje. Upravnik virov opravilo dodeli naključnemu vozlišču.

2. Najkrajša vrsta. Opravilo je dodeljeno vozlišču, ki ima v čakalni vrsti najmanj opravil.

3. Najbližje podatkom. Opravilo je dodeljeno vozlišču, za katerega upravnik virov oceni, da lahko najhitreje prenese vse potrebne podatke.

4. Najkrajša vrsta in najbližje podatkom. To pravilo je kombinacija zgornjih dveh. Upravnik opravilo dodeli vozlišču, ki lahko najhitreje prenese vse podatke opravil, ki so še v čakalni vrsti, in vse podatke opravila, ki ga trenutno razvršča.

V vseh do sedaj opravljenih preizkusih se je zadnje pravilo za razvrščanje obneslo najbolje, zato bomo tudi v naših simulacijah uporabili prav to razvrščanje.

Izvajanje opravil. Ko je pravilo dodeljeno vozlišču, se postavi v vrsto. Ko pride opravilo na vrsto za izvajanje, upravnik kopij najprej prenese vse potrebne datoteke na lokalno vozlišče. Izmed vseh kopij posamezne datoteke

vedno izbere najbližjo, torej tisto kopijo, ki jo lahko najhitreje prenese. Ko je datoteka prenesena na vozlišče, se upravnik kopij odloči:

- ali bo datoteko obdržal na lokalnem vozlišču,
- če na vozlišču ni dovolj prostora, katere datoteke naj izbriše.

Definicija teh dveh pravil določa metodo podvajanja v opisani arhitekturi.

Metode podvajanja. V simulatorju OptorSim so bile preizkušene naslednje metode:

1. LRU. Pri tej metodi je nova datoteka vedno podvojena na lokalno vozlišče. Izbriše se datoteke, ki najdlje niso bile uporabljene.

2. LFU. Tudi pri tej metodi je nova datoteka vedno podvojena na lokalno vozlišče. Metoda pa izbriše tiste datoteke, ki so bile najmanjkrat uporabljene v zadnjem časovnem intervalu.

3. Ekonomski pristop. Podvajanje z ekonomskim modelom je metoda, ki je podrobneje predstavljena v [18]. Ekonomski model vključuje različne akterje (avtonomne entitete, ki želijo doseči neki cilj) in vire v podatkovnem omrežju. Z interakcijo med akterji in viri dosežemo kakovostno postavitev kopij. Cilj akterjev je maksimizirati dobiček in minimizirati ceno upravljanja s podatkovnimi viri. Datoteke predstavljajo dobrine na trgu. Kupujejo jih tako računski viri, kot tudi podatkovni viri. Prvi za opravila, ki se na njih izvajajo, drugi pa kot investicijo za izboljšanje prihodnjih prihodkov. Podatkovni viri so namreč tisti, ki računskim virom datoteke prodajajo. Ekonomski model je uporabljen za odločitev omenjenih problemov, kadar je potrebno podvojiti datoteko na vozlišče in katere datoteke izbrisati, če na vozlišču ni dovolj prostora. Ko vozlišču dodeljeno opravilo zahteva določeno datoteko, upravnik kopij poskuša najti najcenejšo kopijo v podatkovnem omrežju tako, da začne dražbo. Podatkovni viri se udeležijo dražbe, kjer ponudijo ceno za datoteko, ki je enaka ocenjenemu času prenosa na ustrezen računski vir.

   Obenem lahko podatkovni viri, ki nimajo te datoteke, začnejo svojo dražbo, da ugotovijo, če si lahko zagotovijo boljši zaslužek v prihodnosti.

7.2 Podvajanje glede na razdaljo

V prejšnjem poglavju smo preizkusili več različnih algoritmov. Pokazali smo, da je izboljšan požrešni algoritem našel zelo dobre rešitve, primerljive z bolj kompleksno, predvsem pa časovno bolj zahtevno lokalno optimizacijo z menjavo sosednosti.

Zato smo zamisel, ki jo uporabljamo pri požrešnem algoritmu, uporabili tudi za zasnovo porazdeljene metode za podvajanje. Metodo bomo umestili v zgoraj opisano arhitekturo. Pri definiciji metode moramo vpeljati dve pravili: kdaj postaviti kopijo nove datoteke na lokalno vozlišče in katere datoteke izbrisati, da naredimo prostor za novo, če na vozlišču ni dovolj prostora.

Pravili, ki definirata novo podvajanje, sta:

1. metoda vedno poskuša postaviti novo datoteko na lokalno vozlišče,

2. da bi naredila prostor za novo datoteko, izbriše datoteke na vozlišču, ki so "oddaljene" od svojih kopij manj kot novo prispela datoteka.

Bolj formalno je metoda podana v spodnjem opisu.

Algoritem 7: Porazdeljena metoda podvajanja na vozlišču v_i

Vhod: nova datoteka a_i, stanje sistema in trenutna funkcija podvajanja rep

Izhod: nova funkcija podvajanja rep

```
if prostor na vozliscu then
    rep(a_i) := rep(a_i) ∪ {v_i};                    /* datoteko dodamo */
else
    /* uredi datoteke na lokalnem glede na razdaljo do svoje najbližje kopije */
    Urejeni[] := UrediPoRazdalji();
    if Σ_{a∈A:d(a)<d(a_i)} s(a) ≥ s(a_i) then
        k := 0;
        while ni dovolj prostora na vozliscu do
            /* brišemo datoteke, ki so najbližje svoji kopiji */
            rep(Urejeni[k]) := rep(Urejeni[k]) \ {v_i};
            k++;
        end
        rep(a_i) := rep(a_i) ∪ {v_i};
    end
end
```

Poudariti moramo, da je razdalja v tem kontekstu ocenjen čas za prenos datoteke na neko vozlišče. Zaradi pravila za podvajanje poimenujmo novo metodo *DistRep* (angl. *distance based replication*).

7.3 Primerjava z drugimi dinamičnimi metodami

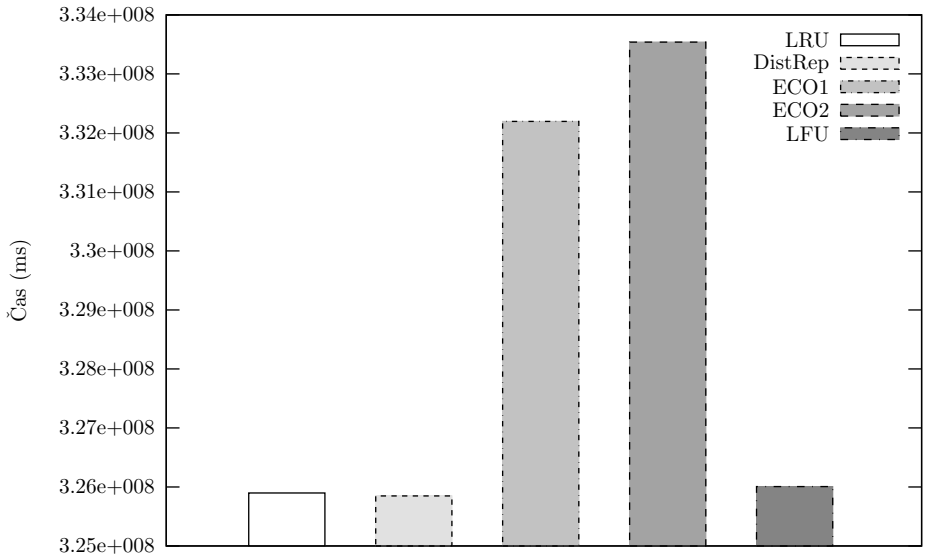
Za primerjavo z ostalimi metodami bomo metode preizkusili na realnih testnih primerih, ki so bili uporabljeni že za ovrednotenje modela v poglavju 5. Preizkus je potekal na štirih realnih testnih primerih: EDG, LCG, CMS in GRIDPP. Primerjali smo metodo DistRep s štirimi že implementiranimi v OptorSimu: LRU, LFU in dvema različicama ekonomske metode. Prva ekonomska metoda za napovedovanje vrednosti datoteke v prihodnosti uporablja binomsko porazdelitev, druga ekonomska metoda pa uporablja Zipfovo porazdelitev . Bolj podroben opis teh metod je podan v [8].

Slike 7.1, 7.2, 7.3 in 7.4 prikazujejo rezultate simulacij štirih testnih primerov. Na grafu je prikazan čas izvajanja množice opravil. Vidimo lahko, da je metoda DistRep boljša na vseh štirih testnih primerih. Na manjšem testnem primeru EDG je čas izvajanja pri uporabi metod LRU in LFU zelo podoben kot pri metodi DistRep. To si lahko razlagamo s tem, da pri manjših testnih primerih topologija še nima tako velikega pomena. Pri večjih testnih primerih pa sta metodi LRU in LFU, pa tudi obe ekonomski metodi, občutno slabši.

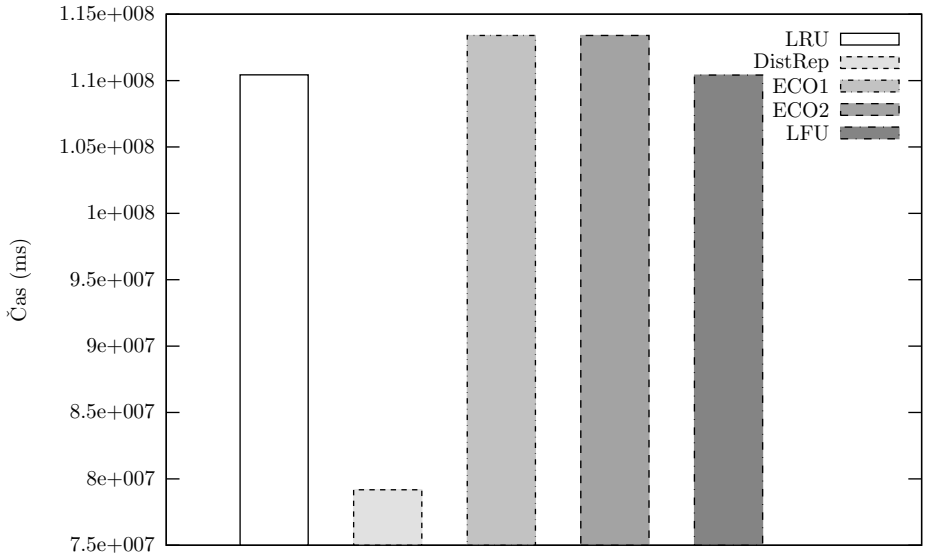
7.3.1 Slabosti trenutne arhitekture

Če podrobneje analiziramo trenutno arhitekturo podatkovnega omrežja, lahko vidimo, da je podvajanje vodeno s povpraševanjem. Podvajanje se namreč zgodi samo, ko je opravilo razvrščeno na neko vozlišče in zahteva določene datoteke.

Arhitektura, kjer povpraševanje usmerja podvajanje podatkov, se je izkazala v sistemih, kjer so aplikacije odjemalci podatkov. V tej disertaciji pa smo že večkrat pokazali, da se podatkovna omrežja razlikujejo od takih sistemov, saj imamo možnost razvrščanja opravil, s katerim lahko vplivamo na porazdelitev povpraševanja. V tem poglavju smo pokazali, da se metoda podvajanja, ki upošteva strukturo podatkovnega omrežja, izkaže bolje kot metode, ki se zgledujejo po sorodnih sistemov.

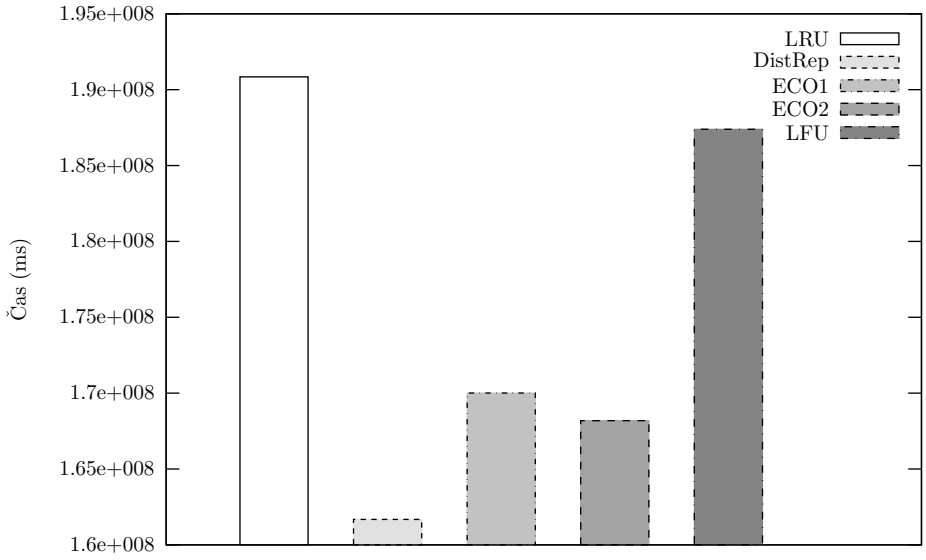


Slika 7.1: Rezultat za EDG

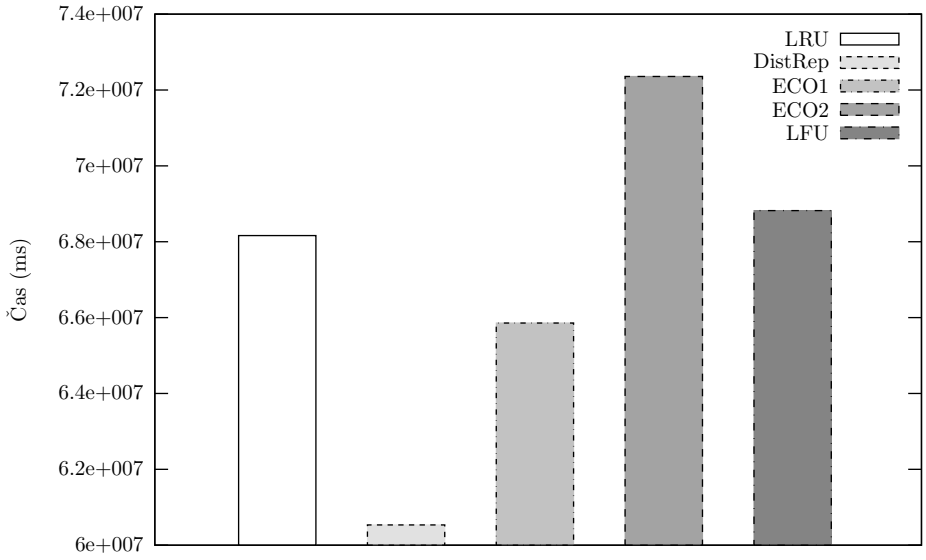


Slika 7.2: Rezultat za LCG

Posledično bi morala biti tudi arhitektura podatkovnega omrežja izdelana za drugačne vrste aplikacij. Če za trenutno arhitekturo lahko rečemo, da je razvrščanje tisto, ki usmerja podvajanje podatkov, bi morala nova arhitektura omogočiti, da podvajanje usmerja razvrščanje opravil.



Slika 7.3: Rezultat za CMS



Slika 7.4: Rezultat za GRIDPP

Osmo poglavje

Zaključek

V tem doktorskem delu smo obdelali podvajanje podatkov v podatkovnih omrežjih. Cilj doktorske disertacije je bil podrobno proučiti podvajanje podatkov v podatkovnih omrežjih, predstaviti teoretično podlago za načrtovanje novih metod podvajanja in predlagati nekatere algoritme in metode. Nekateri rezultati, opisani v tej disertaciji, so bili že objavljeni v mednarodnih revijah [56] in konferencah [52, 53, 54, 55]. V zaključku bomo podali povzetek prispevkov znanosti in nekaj zamisli za prihodnje delo.

8.1 Doseženi rezultati

- Razvrstitev načinov podvajanja podatkov. Izdelali smo razvrstitev pristopov k podvajanju podatkov. V izdelano razvrstitev smo umestili dosedanje delo na področju podvajanja v podatkovnih omrežjih. Razvrstitev predstavlja temelj za nadaljnje delo v disertaciji. Predstavlja namreč sistematičen pogled na podvajanje, ki smo uporabljali tudi v nadaljevanju.

- Izdelava nabora modelov podvajanja podatkov. Zasnovali smo nabor modelov podvajanja podatkov, v katere smo vključili parametre izpostavljene pri razvrstitvi pristopov k podvajanju. Ti modeli predstavljajo formalne temelje za načrtovanje učinkovitih metod podvajanja glede na različne lastnosti sistema in potreb aplikacij, ki se v podatkovnih omrežjih izvajajo.

- Analiza zahtevnosti problema podvajanja podatkov. Izmed zasnovanih modelov smo izbrali enega izmed enostavnih modelov in pokazali,

da je NP-težek in neaproksimabilen. Ti rezultati pokažejo, da za problem najverjetneje ne obstajajo natančni polinomski algoritmi. Zato se moramo pri načrtovanju algoritmov osredotočiti predvsem na hevristike.

- Preizkus modela v simulacijah. Predstavljene modele smo zasnovali na podlagi praktičnega poznavanja podatkovnih omrežij in izkušenj pri modeliranju v teoriji razmeščanja. Da bi pokazali, da model predstavlja kakovostno postavitev podatkov v realnih sistemih, smo ga preizkusili v simulacijah. Prikazali smo postopek, s katerim lahko demonstriramo tudi realno kakovost modela.

- Generator umetnih testnih primerov. Kvaliteto modelov in algoritmov moramo pogosto preizkušati v simulacijah, za kar potrebujemo nabor testnih primerov. Zaradi pomanjkanja realnih testnih primerov smo razvili generator za testne primere. Generator je zasnovan na izsledkih o lastnostih velikih sistemov, zato parametri umetnih testnih primerov posnemajo lastnosti realnih sistemov.

- Zasnova hevristik za reševanje problema podvajanja. Za reševanje izbranega modela $\left\langle implJobs, R, \bigoplus^{\Sigma,\Sigma} \right\rangle$ smo zasnovali nabor hevristik. Primerjali smo jih med seboj na množici testnih problemov. Razvite hevristike lahko enostavno vključimo v podatkovna omrežja kot centralne metode podvajanja.

- Zasnova porazdeljene metode za podvajanje podatkov. Za implementacijo v realnih sistemih pa so, predvsem zaradi boljše razširljivosti, zaželene porazdeljene metode podvajanja. Zato smo zasnovali porazdeljeno metodo, ki temelji na ideji našega požrešnega algoritma. Porazdeljeno metodo smo preizkusili v simulacijah. Pokazali smo, da je za trenutne aplikacije boljša kot obstoječe metode podvajanja.

8.2 Nadaljnje delo

- Implementacija v realnem sistemu. Trenutno implementacija vseh mehanizmov za podvajanje v realnih sistemih še ni na voljo. Ko bodo vsi mehanizmi na voljo, bomo porazdeljeno metodo implementirali v realnem sistemu in jo preizkusili tudi na realnih aplikacijah.

- Branje in pisanje. Aplikacije bodo v prihodnje podatke tudi pisale, zato je potrebno na podlagi predstavljenih modelov zasnovati metode podvajanja, ki upoštevajo tudi ceno sinhronizacije podatkov.

- Podatkovno pretokovna opravila. V podatkovnih omrežjih postajajo zelo aktualna tudi podatkovno pretokovna opravila. V 4. poglavju smo že predstavili modele, ki ta tip opravil vključujejo. Model lahko služi za snovanje tako metod podvajanja podatkov, kot tudi razvrščanja podatkovno pretokovnih opravil. Razvrščanje podatkovno pretokovnih opravil je namreč še vedno zelo aktualen raziskovalni problem.

- Zasnova nove arhitekture. V disertaciji smo izpostavili, da je trenutna arhitektura podatkovnega omrežja zasnovana predvsem za aplikacije, ki vključuje zgolj odjemalce. Podvajanje je namreč vodeno s strani povpraševanja. Zasnovati bi bilo potrebno arhitekturo, v kateri bi podvajanje podatkov vodilo razvrščanje opravil.

Literatura

[1] J. H. Abawajy. File replacement algorithm for storage resource managers in data grids. V zborniku *International Conference on Computational Science*, 2004.

[2] H. Abdel-Wahab, I. Stoica, F. Sultan, and K. Wilson. A simple algorithm for computing minimum spanning trees in the internet. *Information Sciences*, september 1997.

[3] W. Allcock, J. Bester, J. Bresnahan, A. Chervenak, I. Foster, C. Kesselman, S. Meder, V. Nefedova, D. Quesnel, and S. Tuecke. Data management and transfer in highperformance computational grid environments. *Parallel Computing*, 2001.

[4] B. Awerbuch, Y. Bartal, and A. Fiat. Competitive distributed file allocation. V zborniku *Annual ACM Symposium on Theory of Computing*, volume 25, 1993.

[5] S. Bakiras and T. Loukopoulos. Increasing the performance of cdns using replication and caching: A hybrid approach. V zborniku *IEEE International Parallel and Distributed Processing Symposium*, 2005.

[6] Y. Bartal, M. Charikar, and P. Indyk. On page migration and other relaxed task systems. *Theoretical Computer Science*, 268(1):43–66, 2001.

[7] W. H. Bell, D. G. Cameron, L. Capozza, A. P. Millar, K. Stockinger, and F. Zini. Simulation of dynamic grid replication strategies in optorsim. V zborniku *IEEE Workshop on Grid Computing*. Springer Verlag, Lecture Notes in Computer Science, november 2002.

[8] W. H. Bell, D. G. Cameron, R. Carvajal-Schiaffino, A. P. Millar, K. Stockinger, and F. Zini. Evaluation of an economy-based file replication strategy for a data grid. V zborniku *International Workshop on Agent based Cluster and Grid Computing*. IEEE Computer Society Press, maj 2003.

[9] F. Berman, G. Fox, and T. Hey, editors. *Grid Computing: making the global infrastructure a reality*. John Wiley and Sons, 2003.

[10] P. Berman and A. Z.Zelikovsky. On approximation of the power-p and bottleneck Steiner trees. Tehnično poročilo, UCLA Department of Computer Science, Los Angeles, California, 1997.

[11] C. W. Bo. Peer-to-peer overlay networks: A survey. Tehnično poročilo, The Hong Kong University of science and technology, 2003.

[12] V. Boudet, F. Desprez, and F. Suter. One-step algorithm for mixed data and task parallel scheduling without data replication. V zborniku *International Parallel and Distributed Processing Symposium*, 2003.

[13] R. J. Brunner, J. Gray, P. Kunszt, D. Slutz, A. S. Szalay, and A. Thakar. Designing and mining multi-terabyte astronomy archives: The sloan digital sky survey. Tehnično poročilo, Microsoft research, 2000.

[14] R. Buyya. *High Performance Cluster Computing: Architectures and Systems, Vol. 1.* Prentice Hall, 1999.

[15] R. Buyya and M. Murshed. Gridsim: A toolkit for the modeling and simulation of distributed resource management and scheduling for grid computing. *The Journal of Concurrency and Computation: Practice and Experience (CCPE)*, 2002.

[16] R. Buyya, H. Stockinger, J. Giddy, and D. Abrams. Economic models for management of resources in grid computing. V zborniku *International Symposium on The Convergence of Information Technologies and Communications*, 2001.

[17] D. Cameron. Replica management and optimisation for data grids. Tehnično poročilo, University of Glasgow, UK, september 2002.

[18] M. Carman, F. Zini, L. Serafini, and K. Stockinger. Towards an economy-based optimisation of file access and replication on a data grid. V zborniku *International Symposium on Cluster Computing and the Grid*, maj 2002.

[19] A. Chakrabarti, R.A. Dheepak, and S. Sengupta. Integration of scheduling and replication in data grids. *Lecture Notes in Computer Science*, december 2004.

[20] A. Chervenak, I. Foster, C. Kesselman, C. Salisbury, and S. Tuecke. The data grid: Towards an architecture for the distributed management and analysis of large scientific datasets. *Journal of Network and Computer Applications*, 1999.

[21] B. Ciciani, D. M. Dias, and P. S. Yu. Analysis of replication in distributed database systems. *IEEE Transactions on Knowledge and Data Engineering*, 1990.

[22] I. Clarke, O. Sandberg, B. Wiley, and T. W. Hong. Freenet: A distributed anonymous information storage and retrieval system. *Lecture Notes in Computer Science*, 2001.

[23] G. Couloris, J. Dollimore, and T. Kindberg. *Distributed Systems, Concepts and Design*. Addison Wesley, 4 edition, 2005.

[24] M. E. Crovella, M. S. Taqqu, and A. Bestavros. *A Practical Guide To Heavy Tails*, chapter Heavytailed probability distributions in the World Wide Web, page 3–26. Chapman & Hall, 1998.

[25] F. Dabek, E. Brunskill, M. F. Kaashoek, D. Karger, R. Morris, I. Stoica, and H. Balakrishnan. Building peer-to-peer systems with chord, a distributed lookup service. V zborniku *Workshop on Hot Topics in Operating Systems*, 2001.

[26] M. S. Daskin. *Network and Discrete Location: Models, Algorithms and Applications*. John Wiley and Sons, Inc., New York, 1995.

[27] F. Desprez and A. Vernois. Simultaneous scheduling of replication and computation for bioinformatic applications on the grid. V zborniku *Challenges of Large Applications in Distributed Environments*, julij 2005. IEEE Computer Society Press.

[28] R.A. Dheepak, S. Ali, S. Sengupta, and A. Chakrabarti. Study of scheduling strategies in a dynamic data grid environment. *Lecture Notes in Computer Science*, december 2004.

[29] S. Dimopoulos and G. Landsberg. Black holes at the large hadron collider. *Physical Review Letters*, (16), september 2001.

[30] C. Mihai Dobre and C. Stratan. Monarc simulation framework. V zborniku *RoEduNet International Conference*, 2004.

[31] A. Domenici, F. Donno, G. Pucciani, H. Stockinger, and K. Stockinger. Replica consistency in a data grid. V zborniku *International Workshop on Advanced Computing and Analysis Techniques in Physics Research*, 2004.

[32] F. Dong and S. G. Akl. Scheduling algorithms for grid computing: State of the art and open problems. Tehnično poročilo, School of Computing, Queen's University, Kingston, Ontaio, 2006.

[33] L. W. Dowdy and D. V. Foster. Comparative models of the file assignment problem. *ACM Computing Surveys*, 1982.

[34] A. B. Downey. The structural cause of file size distributions. V zborniku *ACM SIGMETRICS international conference on Measurement and modeling of computer systems*, 2001. ACM Press.

[35] T. Erl. *Service-Oriented Architecture (SOA): Concepts, Technology, and Design*. Prentice Hall, 2005.

[36] M. Faloutsos, P. Faloutsos, and C. Faloutsos. On power-law relationships of the internet topology. V zborniku *ACM Special Interest Group on Data Communications*, 1999.

[37] R. Fleischer and S. Seiden. Page replication—variations on a theme. skripta, 1999.

[38] I. Foster, J. Geisler, B. Nickless, W. Smith, and S. Tuecke. Software infrastructure for the i-way metacomputing experiment. *Concurrency: Practice and Experience*, 1998.

[39] I. Foster and A. Iamnitchi. On death, taxes and the convergence of peer-to-peer and grid computing. V zborniku *International Workshop on Peer-to-Peer Systems*, 2003.

[40] I. Foster and C. Kesselman. *The Grid: Blueprint for a new computing infrastructure*. Morgan Kaufmann Publishers, 1999.

[41] I. Foster, C. Kesselman, J. Nick, and S. Tuecke. The physiology of the grid: An open grid services architecture for distributed systems integration. V zborniku *Global Grid Forum*, 2002.

[42] I. Foster, C. Kesselmann, and S. Tuecke. The anatomy of the grid: Enabling scalable virtual organizations. *International Journal of Supercomputer Applications*, 2001.

[43] J. L. Ganley and J. S. Salowe. Optimal and approximate bottleneck Steiner trees. *Operations Research Letters*, 1996.

[44] R. M. Garey and D. S. Johnson. *Computers and Intractability*. W. H. Freeman and Company, 19 izdaja, 1997.

[45] G.Gambosi, V. Kann, A. Marchetti-Spaccamela, and M Protasi. *Complexity and Aproximation: Combinatorial Optimization Problems and Their Approximability Properties*. Springer, 2003.

[46] T. Gonzalez. Clustering to minimize the maximum intercluster distance. *Theoretical Computer Science*, 38:293–306, 1985. posredna referenca.

[47] V. Gopalakrishnan, B. Silaghi, B. Bhattacharjee, and P. Keleher. Adaptive replication in peer-to-peer systems. Technical report, University of Maryland, College Park, 2003.

[48] E. Grapa and G. G. Belford. Some theorems to aid in solving the file allocation problem. *Communications of the ACM*, 1977.

[49] L. Guy, P. Kunszt, E.Laure, H. Stockinger, and K. Stockinger. Replica management in data grids. V zborniku *Global Grid Forum*, junij 2002.

[50] X. He, X. Sun, and G. von Laszewski. "QoS guided min-min heuristic for grid task scheduling. *Journal of Computer Science and Technology, Special Issue on Grid Computing*, 4, 2003.

[51] F. Hui, C. Kang, and I. Yamazaki. Distributed data replication in grid computing. Tehnično poročilo, University of California, 2003.

[52] U. Čibej and B. Robič. A location science approach to data replication in grid computing. V zborniku *European working group on locational analysis*. Krf, Grčija, 2003.

[53] U. Čibej and B. Robič. Locating copies of data in a grid computing environment : a formal model and heuristics. V zborniku *European working group on locational analysis*. Saarbrucken, Nemčija, 2004.

[54] U. Čibej and B. Slivnik. NP-completeness of the incremental data replication problem in data grids. V zborniku *Informacijska družba IS*. Ljubljana, Slovenija, 2004.

[55] U. Čibej, B. Slivnik, and Robič B. Modeling of data replication in data grids. V zborniku *Cracow '04 Grid Workshop*. Krakov, Poljska, 2004.

[56] U. Čibej, B. Slivnik, and B. Robič. The complexity of static data replication in data grids. *Parallel Computing*, 31:900–912, avgust-september 2005.

[57] D. B. Jackson, Q. Snell, and M. J. Clement. Core algorithms of the Maui scheduler. V zborniku *International Workshop on Job Scheduling Strategies for Parallel Processing*, London, UK, 2001. Springer-Verlag.

[58] W. Just. Complexity issues in bioinformatics. Dostopno na `http://www.math.ohiou.edu/~just/TALKS/Complexitytalk/`.

[59] J. Kangasharju, J. Roberts, and K. Ross. Object replication strategies in content distribution networks. *Computer Communications*, 4(24):367–383, 2002.

[60] M. Karlsson, C. Karamanolis, and M. Mahalingam. A framework for evaluating replica placement algorithms. Tehnično poročilo, HP Labs, 2002.

[61] N. T. Karonis, B. Toonen, and I. Foster. Mpich-g2: A grid-enabled implementation of the message passing interface. *Journal of Parallel and Distributed Computing*, 63:551–563, 2003.

[62] S. Kim and J. B. Weissman. A genetic algorithm based approach for scheduling decomposable data grid applications. V borniku *International Conference on Parallel Processing*, 2004.

[63] H. Lamehamedi, Z. Shentu, B. Szymanski, and E. Deelman. Simulation of dynamic data replication strategies in data grids. V zborniku *International Symposium on Parallel and Distributed Processing*, 2003.

[64] A. Legrand, L. Marchal, and H. Casanova. Scheduling distributed applications: The simgrid simulation framework. V zborniku *International Symposium on Cluster Computing and the Grid*, 2003.

[65] M. Li and M. Baker. *The Grid core technologies*. John Wiley and Sons, 2005.

[66] H. Liu, M. Beck, and J. Huang. Dynamic co-scheduling of distributed computation and replication. V zborniku *International Symposium on Cluster Computing and the Grid*, 2006, IEEE Computer Society.

[67] T. Loukopoulos and I. Ahmad. Static and adaptive distributed data replication using genetic algorithms. *Journal of Parallel and Distributed Computing*, november 2004.

[68] T. Loukopoulos, D. Papadias, and I. Ahmad. An overview of data replication on the internet. V zborniku *International Symposium on Parallel Architectures, Algorithms and Networks*, 2002.

[69] Q. Lv, P. Cao, E. Cohen, K. Li, and S. Shenker. Search and replication in unstructured peer-to-peer networks. V zborniku*International conference on Supercomputing*, pages 84–95, New York, NY, USA, 2002. ACM Press.

[70] A. Medina, A. Lakhina, I. Matta, and J. Byers. BRITE: An approach to universal topology generation. V zborniku *International Workshop on Modeling, Analysis and Simulations of Computer and Telecommunications Systems*, avgust 2001.

[71] N. Mladenovič and P. Hansen. Variable neighborhood search. *Computers & Operations Researchs.*, 1997.

[72] G. E. Moore. Cramming more components onto integrated circuits. *Electronics*, 38, 1965.

[73] C. Nicholson, D. G. Cameron, A. T. Doyle, A. P. Millar, and K. Stockinger. Dynamic data replication in LCG 2008. V zborniku *UK e-Science All Hands Meeting*, september 2006.

[74] M. Novak. Porazdeljene imeniške storitve v grid sistemih na osnovi tehnologij enak z enakim. Diplomsko delo, Univerza v Ljubljani, Fakulteta za računalništvo in informatiko, 2006.

[75] S. Park, J. Kim, Y. Ko, and W. Yoon. Dynamic data grid replication strategy based on internet hierarchy. V *Lecture Notes in Computer Science*, 2004.

[76] G. Peng. CDN: Content Distribution Network, 2004.

[77] T. Phan, K. Ranganathan, and R. Sion. Evolving toward the perfect schedule: Co-scheduling job assignments and data replication in wide-area systems using a genetic algorithm. V zborniku *Workshop on Job Scheduling Strategies for Parallel Processing* , 2005.

[78] X. Qin and H. Jiang. Data grid: Supporting data-intensive applications in wide-area networks. Tehnično poročilo, Department of Computer Science and Engineering University of Nebraska-Lincoln, Lincoln, 2003.

[79] L. Qiu, V. N. Padmanabhan, and G. M. Voelker. On the placement of web server replicas. V zborniku *Annual Joint Conference of the IEEE Computer and Communications Societies.*, 2001.

[80] K. Ranganathan and I. Foster. Decoupling computation and data scheduling in distributed data-intensive applications. V zborniku *International Symposium on High Performance Distributed Computing*, 2002.

[81] K. Ranganathan, A. Iamnitchi, and I. Foster. Improving data availability through dynamic model-driven replication in large peer-to-peer communities. V zborniku *Workshop on Global and Peer-to-Peer Computing on Large Scale Distributed Systems*, maj 2002.

[82] L.G. Roberts. Beyond moore's law: Internet growth trends. *Computer*, januar 2000.

[83] D. De Roure, N.R. Jennings, and N.R. Shadbolt. The semantic grid: Past, present, and future. V *Proceedings of the IEEE*, marec 2005.

[84] G. Singh, S. Bharathi, A. Chervenak, E. Deelman, C. Kesselman, M. Manohar, S. Patil, and L. Pearlman. A metadata catalog service for data intensive applications. V zborniku *ACM/IEEE conference on Supercomputing*, 2003. IEEE Computer Society.

[85] H. J. Song, D X. Liu, Jakobsen, R. Bhagwan, X. Zhang, K. Taura, and A. Chien. The microgrid: A scientific tool for modelling computational grids. V zborniku *IEEE Supercomputing Conference*, 2000.

[86] M. Di Stefano. *Distributed data management for grid computing.* Wiley Interscience, 2005.

[87] H. Stockinger. *Database Replication in World-wide Distributed Data Grids.* Doktorska disertacija, Fakultät für Wirtschaftswissenschaften und Informatik, Universität Wien, november 2001.

[88] H. Stockinger, F. Donno, E. Laure, S. Muzaffar, P. Kunszt, G. Andronico, and P. Millar. Grid data management in action: Experience in running and supporting data management services in the eu datagrid project. V zborniku *Computing in High Energy and Nuclear Physics*, 2003.

[89] H. Stockinger, A. Samar, B. Allcock, I. Foster, K. Holtman, and B. Tierney. File and object replication in data grids. V zborniku *IEEE Symposium on High Performance and Distributed Computing*, 2001.

[90] A. Sulistio, U. Čibej, R. Buyya, and B. Robič. A toolkit for modelling and simulation of data grids with integration of data storage, replication and analysis. *Future Generation Computer Systems - oddano v objavo*, 2007.

[91] A. Sulistio, U. Čibej, S. Prasad, Rajkumar Buyya, and Borut Robič. GarQ: An efficient data structure for advance reservations in grid computing. *ICDCS - oddano v objavo*, 2007.

[92] A. Sulistio, U. Čibej, B. Robič, and R. Buyya. A tool for modelling and simulation of data grids with integration of data storage, replication and analysis. Tehnično poročilo, University of Melbourne, Australia, 2005.

[93] A. Takefusa, O. Tatebe, S. Matsuoka, and Y. Morita. Performance analysis of scheduling and replication algorithms on grid datafarm architecture for high-energy physics applications. V zborniku *International Symposium on High Performance Distributed Computing*, 2003.

[94] D. Talia and P. Trunfio. Toward a synergy between p2p and grids. *IEEE Internet Computing*, 2003.

[95] M. Tang, B. Lee, C. Yeo, and X. Tang. Dynamic replication algorithms for the multi-tier data grid. *Future Genereration Computer Systems*, 2005.

[96] S. Vazhkudai, S. Tuecke, and I. Foster. Replica selection in the globus data grid. V zborniku *International Symposium on Cluster Computing and the Grid*, 2001.

[97] S. Venugopal, R. Buyya, and K. Ramamohanarao. A taxonomy of data grids for distributed data sharing, management, and processing. *ACM Computing Surveys*, marec 2006.

[98] Allcock W., Foster I., Tuecke S., Chervenak A, and Kesselman C. Protocols and services for distributed data-intensive science. V zborniku *AIP Conference*, 2000.

[99] L. A. Wolsey. *Integer Programming*. Wiley-Interscience, 1998.

[100] R. Wolski, N. Spring, and C. Peterson. Implementing a performance forecasting system for metacomputing: the network weather service. V zborniku *ACM/IEEE conference on Supercomputing*,1997. ACM Press.

[101] H.M. Wong, D. Yu, B. Veeravalli, and T.G. Robertazzi. Data intensive grid scheduling: Multiple sources with capacity constraints. V zborniku *Parallel and Distributed Computing and Systems*, 2003.

[102] X. Sun X. He. Incorporating data movement into grid task scheduling. V *Lecture Notes in Computer Science*, 2005.

[103] D. Zeinalipour-Yazti and T. Folias. A quantitative analysis of the gnutella network traffic.

[104] B. Y. Zhao, J. D. Kubiatowicz, and A. D. Joseph. Tapestry: An infrastructure for fault-tolerant wide-area location and. Tehnično poročilo, Berkeley, CA, USA, 2001.

[105] Akamai, domača stran. `http://www.akamai.com/`.

[106] The astro grid project homepage. `http://www.astrogrid.org/`.

[107] Biology workbench, omrežni portal. Dostopno na `http://workbench.sdsc.edu/`.

[108] Biomedical informatics research network. Dostopno na `http://www.nbirn.net/`.

[109] Cactus, domača stran. Dostopno na http://www.cactuscode.org/.

[110] DigitalIsland, domača stran. Dostopno na http://www.digitalisland.com/.

[111] The european DataGrid project domača stran. Dostopno na http://www.eu-datagrid.org/.

[112] Grid portal development kit, domača stran. Dostopno na http:/doesciencegrid.org/projects/GPDK/.

[113] IBM grid computing, domača stran. Dostopno na http://www-1.ibm.com/grid/solutions/AA_financial.shtml.

[114] International virtual data grid laboratory, domača stran. Dostopno na http://www.ivdgl.org/.

[115] NASA launchpad, omrežni portal. Dostopno na http://portal.ipg.nasa.gov.

[116] Network for earthquake engineering simulation. Dostopno na http://www.neesgrid.org/.

[117] OptorSim, a replica optimiser simulation. http://edg-wp2.web.cern.ch/edg-wp2/optimization/optorsim.html.

[118] The particle physics data grid project homepage. Dostopno na http://www.ppdg.net/.

[119] Projekt AccessGrid. Dostopno na http://www.accessgrid.org/.

[120] Projekt FightAids@home, domača stran. Dostopno na http://www.fightaidsathome.org/.

[121] Projekt GridPhyN domača stran. Dostopno na http://www.griphyn.org/.

[122] Projekt NARA, domača stran. Dostopno na http://www.sdsc.edu/NARA/.

[123] Projekt Nug30. Dostopno na http://www-unix.mcs.anl.gov/metaneos/nug30/.

[124] Projekt SETI@home. Dostopno na http://setiathome.berkeley.edu/.

[125] Projekt TeraGrid. Dostopno na http://www.teragrid.org/.

[126] Simulator ChicSim. Dostopno na http://people.cs.uchicago.edu/krangana/ChicSim.html.

[127] Projekt FAFNER. Dostopno na `http://www.npac.syr.edu/factoring.html`, 2006.

[128] Grid computing info center. Dostopno na `www.gridcomputing.com`.

[129] Globus, domača stran. `www.globus.org`.

Dodatki

Simboli in okrajšave

Okrajšave	Opis	Definicija
P2P	peer-to-peer - sistem P2P, enak z enakim	stran 17
CDN	content delivery network - mreža za dostavo vsebine	stran 19
LHC	large hadron collider - veliki hadronski trkalnik	stran 23
OGSA	open grid service arhitecture - odprta arhitektura omrežnih storitev	stran 26
SOA	service oriented architecture - storitveno usmerjena arhitektura	stran 26
LRU	least recently used	stran 40
LFU	least frequently used	stran 40
VNS	variable neighbourhood search -lokalna optimizacija z menjavo sosednosti	stran 83
EDG	european data grid - evropsko podatkovno omrežje	stran 71
CMS	compact muon solenoid	stran 71
LCG	LHC computing grid - računsko omrežje za LHC	stran 71

Slike

Tabele

Stvarno kazalo

Izjava

Izjavljam, da sem doktorsko disertacijo izdelal samostojno pod vodstvom mentorja prof. dr. Boruta Robiča. Izkazano pomoč drugih sodelavcev sem v celoti navedel v zahvali.

Uroš Čibej