Univerza v Ljubljani
Fakulteta za računalništvo in informatiko
Fakulteta za matematiko in fiziko

Dejan Kovač

# Razpodvajanje podatkov v podatkovni shrambi

diplomsko delo

univerzitetni interdisciplinarni študijski program prve stopnje računalništvo in matematika

Mentor: dr. Andrej Brodnik

Ljubljana 2014

*Besedilo je oblikovano z urejevalnikom besedil LaTeX.*

Fakulteta za računalništvo in informatiko izdaja naslednjo nalogo:

Tematika naloge:

Količina podatkov shranjenih v podatkovnih shrambah dosega nikoli videne razsežnosti. Shramba že dolgo ni več preprosta V/I diskovna naprava na uporabnikovem računalniku, ampak postaja storitev v oblaku - SaaS (Storage as a Service). S tem, ko postaja storitev, je tudi učinkovito hranjenje podatkov postalo zelo pomembno. Eden od postopkov za povečanje učinkovitosti hranjenja je takšno in drugačno stiskanje podatkov. Klasičen pristop uporablja različne algoritme za stiskanje.
Poleg klasičnega se je pojavilo tudi sodobnejše stiskanje, ki shranjuje ene podatke samo enkrat in mu rečemo razpodvajanje (deduplikacija, angl. deduplication). V diplomski nalogi preglejte različne postopke za razpodvajanje in se osredotočite na tiste, ki omogočajo razpodvajanje ne vedno enakih velikosti blokov podatkov.

# Izjava o avtorstvu diplomskega dela

Spodaj podpisani Dejan Kovač, z vpisno številko **63040418**, sem avtor diplomskega dela z naslovom:

*Razpodvajanje podatkov v podatkovni shrambi*

S svojim podpisom zagotavljam, da:

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

V Ljubljani, dne 30. oktobra 2014 Podpis avtorja:

*Zahvaljujem se mentorju dr. Andreju Brodniku za napotke pri izdelavi diplomskega dela. Za pomoč bi se rad zahvalil tudi Andreju Toliču in vsem prijateljem ter sošolcem, ki so me spremljali v času študijskih let. Zahvalil bi se še svoji družini, ki me je vzpodbujala in nudila oporo v vseh letih študija.*

# Kazalo

# Povzetek

V diplomski nalogi smo podrobneje predstavili področje razpodvajanja podatkov in implementirali algoritem shranjevanja objektov s privzetim odstranjevanjem podvojenih kosov.

V prvem delu naloge smo predstavili shranjevalni sistem kot drevesno urejeno strukturo imenikov in datotek. Opisali smo funkcije shranjevalnega sistema ter preproste načine shranjevanja podatkov na medij. Podrobneje smo pregledali lastnosti porazdeljenega shranjevalnega sistema Ceph, njegove sestavne dele in delovanje. V drugem delu predstavimo metodo razpodvajanja podatkov kot pomembno sestavino sodobnih shranjevalnih sistemov. Pregledali smo tehnike razpodvajanja na centraliziranih in porazdeljenih sistemih.

V zadnjem delu naloge smo implementirali primer razpodvajanja skupaj s preprostim sistemom za shranjevanje objektov. S pomočjo opisanih tehnik smo izvedli zaznavanje spremenljivo dolgih podvojenih kosov znotraj objektov in dodali CLI orodja za manipulacijo objektov v shrambi.

**Ključne besede:** razpodvajanje, datotečni sistem, shranjevanje podatkov

# Abstract

In our thesis we presented the area of data deduplication and implemented an algorithm for object storage with support for elimination of duplicate chunks within those objects.

In the first part we presented storage system as a tree-likestructure of directories and files. We described the features of storage system and simple ways of storing data on a medium. We examined in detail the properties of distributed storage system Ceph, it's components and operation. In the second part we presented deduplicatin as an important feature of modern storage systems. We surveyed deduplication techniques for centralized as well as distributed systems.

In the last part we implemented an example of deduplication technique along with a simple object storage system. Using the described techniques we implemented detection of variable-length duplicated chunks within objects and added CLI tools for manipulating objects in the store.

**Keywords:** deduplication, filesystem, data store

# Poglavje 1

# Uvod

Naraščajoče količine podatkov v sodobnem svetu uspešno pogojujejo razvoj inovativnih rešitev na področju shranjevanja podatkov. Nizka cena podatkovnih medijev in visoka zmogljivost sodobnih računskih sistemov omogočata realizacijo veliko možnosti pri shranjevanju informacij. Trend shranjevanja velikih količin podatkov v porazdeljenih sistemih raste. Porazdeljen sistem je povezan sistem računalnikov, strežnikov in drugih naprav v kompleksno in zmogljivo računalniško omrežje. Namenjen je shranjevanju podatkov in deljenju računskih virov. Porazdeljen shranjevalni sistem imenujemo tudi oblak ali oblačna shramba podatkov.

Izziv, s katerim se srečujemo vsi, ki podatke želimo hraniti za daljši čas je njihova podvojenost na podatkovnih medijih. Podvojeni so tisti podatki, ki so na datotečnem sistemu podatkovnega nosilca zapisani večkrat v obliki celotnih datotek ali posameznih kosov datotek. Podvojeni zapisi zasedajo prostor, ki bi ga funkcionalno lahko izkoristili bolje za hrambo novih podatkov. Veliki sistemi težavo uspešno premoščajo z optimizacijo prostora z metodo deduplikacije. Podatkovna deduplikacija je bistvena in kritična sestavina sodobnih shranjevalnih sistemov za trajno arhiviranje podatkov. Bistveno vpliva na povečanje shranjevalnih kapacitet in deluje kritično na zmogljivosti sistemov.

## 1.1 Struktura vsebine

Po krajšem uvodu je namen poglavja 2 bralcu predstaviti pomen shranjevalnega sistema in njegovo osnovo delovanje na abstraktnem nivoju. V splošnem pojasnimo pomembnost operacijskega sistema za pravilno delovanje shranjevalnega sistema. V nadaljevanju ga opredelimo kot datotečni sistem in na preprost način razložimo njegovo uporabnost in podobnost z drugimi sistemi. Z vpeljavo pojma datoteke in imenika prikažemo način za gradnjo urejene drevesne strukture za shranjevanje podatkov v sistemih UNIX.

V poglavju 3 najprej opredelimo pojem porazdeljenega shranjevalnega sistema. Opišemo sestavne dele in njihovo osnovno delovanje. Podrobneje predstavimo porazdeljen shranjevalni sistem *Ceph*.

V poglavju 4 se posvetimo deduplikaciji podatkov. Podrobneje opišemo delovanje osnovnega algoritma in kje se izvaja. Predvsem nas zanima delovanje algoritma nad vsebino datoteke in identificiranje njene vsebine po kosih različnih dolžin ter označevanje le teh s prstnimi odtisi. Na koncu poglavja primerjamo pristope odstranjevanja podvojenih kosov vsebine v datotekah na porazdeljenih in centraliziranih sistemih.

Poglavje 5 je namenjeno implementaciji in predstavitvi lastnega algoritma. Izdelamo vmesnik shrambe, ki deluje kot slovar in shranjuje datoteke kot objekte. Posebnost algoritma shrambe je sposobnost deduplikacije vsebine objektov po celotni vsebini ali po delih. Za enostavnejšo uporabo shrambe podatkov izdelalamo še uporabniško CLI orodje.

# Poglavje 2

# Shranjevalni sistemi

Operacijski sistem je programska oprema, ki omogoča učinkovito uporabo računalniških virov na uporabniku standariziran način. Predstavlja vmesnik med uporabnikom in strojno opremo, zato ga pogosto definirata uporabniški in sistemski vidik. Z uporabniškega vidika je najpomembnejša lastnost, da je sistem uporabniško prijazen in preprost pri upravljanju. S sistemskega vidika je prednost operacijskega sistema njegova učinkovitost pri delovanju.

Delovanje operacijskega sistema vodi jedro, ki je hkrati najmajši sestavni del sistema in je ves čas delovanja prisotno v glavnem pomnilniku. Med naloge operacijskega sistema štejemo upravljanje s procesi, glavnim, sekundarnim in navideznim pomnilnikom, omreženjem, zagotavljanjem varnosti in zaščite, interpretacijo ukazov in upravljenjem z datotečnim sistemom. Upravljanje z datotečnim sistemov je tisti del operacijskega sistema, ki skrbi za trajno shranjevanje podatkov tudi potem, ko se izvajanje procesa že zaključi. Za trajno shranjevanje podatkov se uporabljajo različni mediji s svojimi tehničnimi in zgodovinskimi lastnostmi, ki se med seboj razlikujejo po logični in fizični organizaciji podatkov.

Datotečni sistem definira logično organiziranost podatkov na različnih medijih s katerim upravlja operacijski sistem. Podatki se shranjujejo v datotekah (*angl. file*) razporejenih v imenikih (*angl. directory*). Naloga imenika je urejeno shranjevanje datotek in preostalih imenikov v logično razvejano

drevesno strukturo. Datoteka je enodimenzionalno polje bajtov namenjeno shranjevanju vsebine informacije. Informacija o lastnosti datoteke je shranjena v strukturi imenovani *inode*.

## 2.1 Lastnosti sistemov

Datoteka je abstraktni mehanizem za shranjevanje in vračanje informacije iz shranjevalnega medija. Implementirana je tako, da uporabniku prikaže informacije, ki ga zanimajo. Ponavadi je to vsebina in poimenovanje, ki se v procesu uporabe datoteke ne spreminja. Uporabniku zato ni potrebno vedeti kje in kako je podatek shranjen ter kako deluje shranjevanje na mediju. Datoteka je v splošnem shranjena kot zaporedje bajtov v enodimenzionalnem polju fiksne dolžine ali v drevesni strukturi. Določena je z imenom, vsebino in atributi (tudi metapodatki) za enostavnejše sistemsko delovanje.

Datoteke shranjujejo podatke za poznejšo uporabo. Različni sistemi zagotavljajo različne operacije za shranjevanje in vračanje vsebine datoteke v uporabo. Operacije uporabljajo naslednje sistemske klice [2]:

**Create**

Create ustvari datoteko brez vsebine. Datoteka vsebuje nekaj atributov in je pripravljena za uporabo.

**Delete**

Delete izbriše datoteko, ki ni uporabna več in sprosti prostor na disku.

**Open**

Open odpre datoteko in dovoli sistemu uporabo vsebovanih atributov in vsebine.

**Close**

Close zapre datoteko, ko njena vsebina in atributi niso več uporabni v sistemu. Sistemu sprosti prostor v uporabo drugim vsebinam.

**Read**

Read prebere zaporedje bajtov informacije iz datoteke in jih začasno shrani v strukturo za nadaljno uporabo.

**Write**

Write zapiše zaporedje bajtov informacije v datoteko. Velikost datoteke se poveča, če se vsebina doda na koncu obstoječega zapisa, sicer se vsebina prepiše in velikost datoteke ostane ista.

**Append**

Append doda zaporedje bajtov informacije na konec že obstoječe vsebine datoteke. Podobno deluje operacija Write.

**Seek**

Seek je sistemski klic, ki usvari kazalnik na točno določeni del vsebine. V nadaljevanju se vsebina prebere ali zapiše na določeno mesto v datoteki.

**Lock**

Lock zaklene datoteko pred nezaželjenimi dostopi s strani sistema ali uporabnika.

**Get attributes**

Get attributes je sistemski klic za zagotavljanje atributov.

**Set attributes**

Set attributes je sistemski klic za nastavljanje atributov.

Imenik (*angl. directory*) je mehanizem namenjen shranjevanju datotek in imenikov v urejeno in razumljivo drevesno strukturo. Gradnja drevesne strukture v okolju UNIX izhaja vedno iz korena (*angl. root*), ki je definiran kot zapis s poševno črto `/`. Datoteko `datoteka.txt` določa pot iz korena in jo zapišemo kot `/home/user/datoteka.txt`. Urejena drevesna struktura omogoča operacijskemu sistemu preprosto iskanje datotek nad katerimi se izvajajo datotečne operacije. Razvejanost datotečnega sistema z več imeniki

je nujna v večuporabniških operacijskih sistemih, kjer uporabniku pogosto uporabjajo enaka poimenovanja datotek. V primeru se uporabniku dodeli lasten imenik za shranjevanje podatkov. Običajno je tako na enostaven način zagotovljena varnost pred neposrednim poseganjem uporabnikov v sosednji uporabniški prostor. V določenih primerih je neposeganje v drug uporabniški prostor tudi slabost, kadar npr. želimo deliti datoteke z drugimi uporabniku. Takrat se datoteki spremenijo pravice za omogočanje dostopa drugih uporabnikov s postavitvijo v deljeno skupino. Upravljanje z imeniki je podobno kot z datotekami urejeno s posebnimi operacijami [2]:

**Create**
: Create ustvari prazen direktorij.

**Delete**
: Delete izbriše mapo, ki je predhodno prazna.

**Opendir**
: Opendir odpre mapo in omogoči branje mape.

**CloseDir**
: CloseDir zapre prebrano mapo.

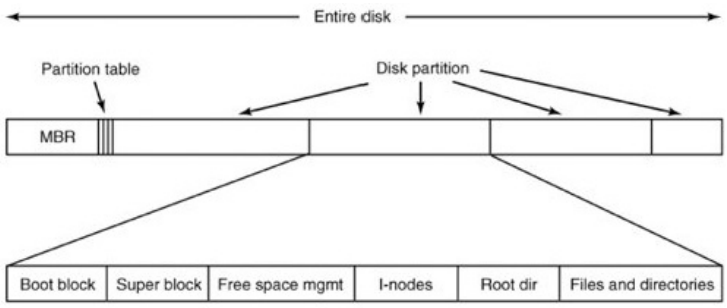
**Readdir**
: Readdir prebere vsebino mape.

**Rename**
: Rename preimenuje mapo.

**Link**
: Link omogoči povezavo do imenika.

## 2.2 Implementacija

Datotečni sistem je shranjen v pomnilniškem mediju. Sektor 0, imenovan MBR (*angl. Master Boot Record*), je namenje zagonu celotnega sistema




Slika 2.1: Razvrstitev particij shranjevalnega medija [2]

in shranjuje particijsko tabelo s pozicijskimi naslovi particij na naslednjem naslovu. Ob zagonu BIOS prebere MBR, kateri v nadaljevanju aktivira zapis prvega bloka (*angl. boot block*) aktivne particije in zažene operacijski sistem. Slika 2.1 prikazuje razporeditev particij na podatkovnem mediju. [2]

Podatke na pomnilniškem mediju je mogoče shranjevati na več načinov. Preprosto jih razporedimo v zaporedne bloke s stičnim razporejanjem (*angl. Contiguous allocation*). Primer: Datoteka velikosti 100KB je razporejena v 100 zaporednih blokov, pri predpostavki, da je velikost bloka na mediju 1KB. Prednost stičnega razporejanje je v enostavni implementaciji in hitrosti iskanja datoteki. Ker so podatki razporejeni zaporedno v blokih je pri iskanju datoteke potrebno določiti samo naslov prvega bloka. Stično razporejanje kasneje nadomesti povezan seznam (*angl. Linked list allocation*), ki veliko učinkoviteje odpravlja časovne zamike pri iskanju razdrobljenih blokov datotek in sestavljanju datotek z manjkajočimi bloki. Povezan seznam obravnava bloke kot objekte z lastnostjo kazalnika, ki se povezuje na naslednji uporaben blok. Hitrost iskanja prvega bloka je v nadaljevanju razvoja podkrepila še tabela shranjena v glavnem pomnilniku, zato lahko govorimo o povezanem seznamu z lokacijsko tabelo, preprosteje FAT (*angl. File allocation table*).

Učinkovit način shranjevanja datotek v bloke je še struktura indeksiranja blokov imenovana inode. Struktura shranjuje naslove blokov, ki pripradajo datoteki v obliki tabele. V glavnem pomnilniku se pojavi takrat, ko je datoteka odprta, sicer prostora ne zaseda.

V prejšnjih poglavjih smo opisali način shranjevanja surovih podatkov v obliki ustrezno označenih blokov. Uporabniku so prikazani na način urejene drevesne strukture datotek in imenikov. Drevesna struktura izhaja iz korena /, ki je fiksiran relativno na začetek particije. Dostop do vsebine datoteke je mogoč, ko operacijski sistem s sistemskim klicem odpre ustrezno mapo in prikaže pot do datoteke v obliki `/home/user/datoteka.txt`. Začetni naslov datoteke na particiji se prebere s pomočjo naslova stičnega razporejanja, številke bloka iz povezanega seznama ali številke indeksa pri inode strukturi. Neglede na način pa je cilj preiskovanja datotečnega sistema vedno, uporabniku vrniti vsebino naslovljene datoteke v jasni in razumljivi obliki. Posredno se pojavljajo še atributi lastništva, datuma nastanka in sprememb datoteke, ki se običajno hranijo v inode strukturi.

## 2.3 Podatki

Datotečni sistemi hranijo pomembne uporabniške podatke. Osnovna naloga sistema je zato varno shranjevanje podatkov. S stališča varnosti so postavljeni cilji za opredelitev doslednosti pri shranjevanju. Delimo jih na naslednje tri [2]:

- Zaupnost podatkov (*angl. Data confidentiality*) določa prikaz zaupnih podatkov določeni skupini uporabnikov. Uporabnik uporablja svoje podatke tako kot se odloči in jih posreduje drugim uporabnikom po svoji presoji.
- Podatkovna integriteta (*angl. Data integrity*) prepoveduje drugim uporabnikom spreminjanje vsebine datotek tako z brisanjem kot z dodajanjem napačnih informacij k obstoječi vsebini.
- Razpoložljivost sistema (*angl. System availability*) opredeljuje stabilnost datotečnega sistema pred možnimi zlorabami npr. zlonamernimi programi.

# Poglavje 3

# Porazdeljeni shranjevalni sistemi

Porazdeljeni shranjevalni sistem je povezan sistem računalnikov, strežnikov in drugih naprav v kompleksno računalniško omrežje. Namenjen je shranjevanju in deljenju podatkov med napravami. V poglavju opisujemo sestavne dele porazdeljenega sistema, njihovo delovanje in lastnosti ter se podrobneje seznanimo s porazdeljenih datotečnim sistemom Ceph.

## 3.1 Osnove

Sodobni porazdeljeni shranjevalni sistemi (*angl. Distributed Storage Systems*) so prvenstveno namenjeni shranjevanju podatkov. Običajno hranijo podatke na različnih napravah za daljše časovno obdobje. Porazdeljenim sistemov pogosto pravimo tudi mreže ali gruče (*angl. cluster*), ker so naprave med seboj logično povezane. Povezano in usklajeno delovanje omogočajo notraji algoritmi. Do uporabnika je sistem transparenten, zato pravimo, da je predstavljen kot črna škatla.

Komunikacijo uporabnika s porazdeljenim sistemov vzpostavlja uporabniški vmesnik, ki je v osnovi podoben sorodnemu sistemu na delovnih sistemih. Vzpostavljanje komunikacije z napravami omogočajo ukazi, usklajeni po PO-

SIX standardu. Vmesnik porazdeljenega datotečnega sistema sestavlja zbirka datotek s shranjenimi podatki in direktorijska struktura, ki organizira datoteke in imenike v urejeno drevesno strukturo ter vsebuje datotečne informacije oziroma metapodatke. Metapodatki so informacije o velikosti in nastanku datoteke, dovoljenja za spreminjanje vsebine uporabnikov, itd. Datotečna struktura je vedno določena s strani uporabnika oziroma aplikacije, ki jo poganja sistem. Omenjajmo zanimivejše primere datotečnih sistemov [13]:

**NFS.** Omrežni datotečni sistem (*angl. Network File System*) je razširjen primer porazdeljenega shranjevalnega sistema DFS (*angl. Distributed File System*), ki uporabniku omogoča omrežni TCP/IP dostop do podatkov. Ko uporabnik sproži zahtevek po zahtevanem podatku, se strežnik odzove z vrnitvijo željenih informacij. Razvili so ga v laboratorijih Sun Microsystems v 80-ih letih, kjer so ga prvotno uporabjali na svojih delovnih postajah. NFS je odprt standard primeren za implemetacijo poljubnih protokolov.

**AFS** je sistem (*angl. Andrew File System*), ki ga v osnovi gradijo strežniki in uporabniki. Strežniki shranjujejo podatke v obliki surovih blokov na disku. Uporabniki dostopajo do referenc podatkov v datotečni strukturi na strežniku. Ob branju in pisanju se prenese zahtevana datoteka v delovni pomnilnik na strežniku, kjer jo prevzame uporabnik. Datotečni sistem ustvarja sliko (*angl. Snapshot*) realnih podatkov in neposredno razpolaga zgolj z metapodatki.

**WAFL** je protokol (*angl. Write Anywhere File Layout*) za zagotavljanje posnetkov datotečnega sistema. Implementirana funkcija Kopiraj ob branju (*angl. Copy-On-Write*) omogoča zajemanje posnetkov ob vsaki spremembi datotečnega sistema ali zamenjavi bloka. Metapodatke shranjuje na posebno lokacijo v datoteko zaradi hitrega dostopa do njih. Uporabnikom omogoča povrnitev izbrisanih datotek.

## 3.2 Datotečni sistem Ceph

Datotečni sistem Ceph je porazdeljen shranjevalni sistem, ki uporabnikom omogoča dostop s standardnimi POSIX ukazi. Pogosto je predstavljen kot blizu-POSIX (*angl. near-POSIX*). Standarizirana logika omogoča kompleksnejši dostop do podatkov, hitrejše posodobitve, replikacijo in zanesljivost sistema, izdelovanje varnostnih kopij ter povrnitev osnovnega sistema v primeru nezgod.

Za uporabnika je značilno, da vzpostavi povezavo s sistemov Ceph s prirejeno aplikacijo. Poganja jo v uporabniškem prostoru lastnega datotečnega sistema. Aplikacija zajame lokalno strukturo imenikov in datotek ter jih prenese v porazdeljen sistem. Korenski imenik datotečne strukture določi uporabnik sam.

Podatki se shranjujejo v gruči naprav OSD (*angl. Object Store Device*). Namenjena je shranjevanju podatkov in metapodatkov. Strežnik gruče metapodatkov MDS (*angl. Metadata Server*) razpolaga z metapodatkovnimi ukazi. Tako je zagotovljena zmogljivost sistema in prožnost komunikacije. Bistvena je razširjivost prostorskih kapacitet in zmožnost upravljanje z njimi. Zato je posebna pozornost namenjena datotekam shranjenim na super gručah, ki se v kosih spreminjajo na več napravah hkrati. V splošnem so bistvene značilnosti sistema Ceph naslednje:

- Ločevanje podatkov in metapodatkov (*angl. Decoupled Data and Metadata*) - Ceph ločuje podatke na meta- in navadne podatke. Z metapodatkovnimi operacijami, kot npr. open, rename, upravlja strežnik MDS metapodatkovne gruče. Uporabniki prosto dostopajo do metapodatkovnih operacij na strežniku. Do pisalnih in bralnih ukazov za izmenjavo datotek na I/O vhodno-izhodnih sistemih dostopajo neposredno do naprav OSD. Podatki so porazdeljeni na napravah OSD v bloke označene z imenom in dolžino.

- Dinamično porazdeljevanje metapodatkov (*angl. Dynamic Distributed Metadata Management*) - Metapodatkovne operacije obremenjujejo sis-

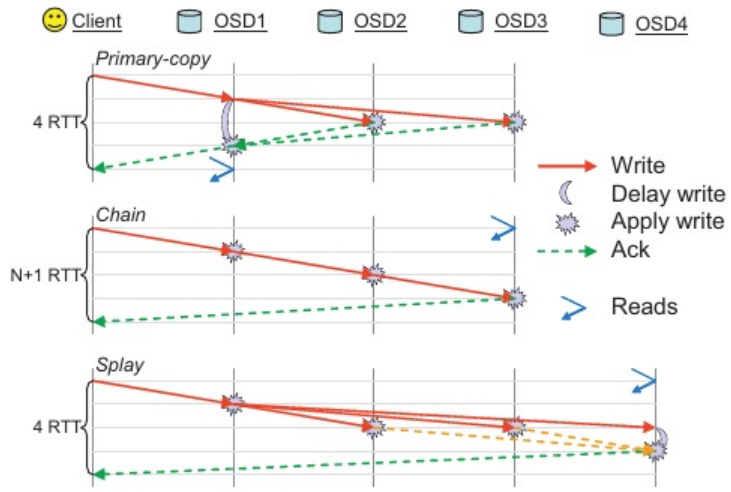
tem pogosteje kot vsi ostali prenosi podatkov. Razbremenjevanje Cepha je v dinamičnem drevesnem razporejanju (*angl. Dynamic Subtree Partitioning*) metapodatkov med strežniki MDS, ki zagotavlja lokalnost dostopov.

- Porazdeljena objektna shramba (*angl. Reliable Autonomic Distributed Object Store*) - Veliki sistemi so grajeni dinamično: kapacitete se spreminjajo z dodajanjem nove in odvzemanjem izrabljene opreme. Sistem v celoti poskrbi za zaznavanje in odpravljanje napak ter prerazporejanje podatkov med OSD.

## 3.3 Delovanje sistema RADOS

Ceph datotečni sistem uporablja RADOS za shranjevanje podatkov. Značilno je, da podatke in računske obremenitve uravnoteženo razporeja med dinamično in heterogeno gručo, ki jo sestavljajo naprave OSD in uporabniki. Naprava OSD je samostojna enota z lastnim procesorjem, dodeljenim delovnim pomnilnikom, omrežno kartico in lokalnim diskovnim poljem. Vsaka naprava shranjuje podatke kot objekte v blokih predpisanih velikosti. Ko uporabnik dostopi do željene datoteke, pošlje ustrezen POSIX ukaz metapodatkovnemu strežniku MDS. Ukaz vsebuje podatke njegove uporabnosti in pot do podatkov do katerih dostopa. Strežnik metapodatkovne gruče se odzove z ustrezno funkcijo ter uporabniku vrne podatke v obliki datotek. Uporabniške aplikacije zato dostopajo do podatkov tako kot bi dostopale do lastnega datotečnega sistema.

Porazdeljena gruča podatkov je definiriana v posebni mapi (*angl. Cluster map*), ki hrani informacijo o napravah OSD in shranjenih podatkih. Mapa se shranjuje v napravah OSD, zato uporabnikom omogoča časovno učinkovit dostop do željenih podatkov. Vsaka sprememba nad shranjenimi bloki podatkov spremeni vsebino mape na vseh omenjenih napravah. Pri dodajanju nove naprave OSD se vsebina obstoječih naprav prerazporedi in ponovno uravnovesi sistem pred preobremenitvami na posameznih napravah.




Slika 3.1: Replikacija podatka v porazdeljeni gruči

Podatki oziroma objekti, ki se prerazporejajo, se najprej preslikajo v umestitveno skupino PG (*angl. Placement group*) ali zbirko objektov kopiranih blokov naprav. Vsak objekt je določen z hash vrednostjo imena $o$, stopnjo podvojitev $r$ in z bitno masko $m$, ki določa število namestitvenih skupin v sistemu. Zapišemo kot pgid = (r, hash(o) AND m) in masko $m = 2^k - 1$. Z rastjo gruče se periodično povečuje tudi število umestitvenih skupin pripete posameznemu naprav OSD.

Porazdeljevanje objektov po napravah poteka s redistribucijskim algoritmom CRUSH. V mapi *Cluster map* se nahaja opis gruče in stanje naprav na katerih se podatki nahajajo. Za vsako umestitveno skupino PG algoritem CRUSH ustvari seznam aktivnih naprav, ki tvorijo gručo. Neaktivne naprave izloči. V primeru, ko aktivnih naprav ni, umestitvena skupina ni dosegljiva.

Informacije shranjene v porazdeljeni gruči omogočajo sistemu RADOS učinkovito upravljanje z podatki, njihovo distribucijo po vozliščih in odpravljanje napak. Podobno vlogo prevzamejo tudi takrat, kadar za shranjenimi podatki zaprosi uporabnik. Slika 3.1 prikazuje odziv sistema na uporabniški zahtevek.

Poznamo tri načine posredovanja podatkov v napravo:

1. *Primary-copy* zaprosi podatke vzporedno na več sosednjih napravah hkrati. Sestavljanje podatkov se zgodi v glavni napravi, ko pridobi vse kose podatkov iz vseh sodelujočih naprav.

2. *Chain* je verižni način, ki iskan podatek zbira in sestavlja zaporedno po napravah. V končno napravo ga dostavi sestavljenega.

3. *Splay replication* je hibridni način in upošteva oba načina hkrati.

V porazdeljenem sistemu komunikacija med napravami poteka asinhrono s sporočili označenimi z zgodovino pošiljateljivih aktivnosti (*angl. Map epoch*). Tako je zagotovljena doslednost pri pridobivanju zahtevanih podatkov in možnost zavrnitve pridobljeinh podatkov v primeru naslavljanja napačnega prejemnika. Lokalno se izvaja pošiljanje sporočil za osveževanje umestitvenih skupin PG in vozlišč OSD s posebnimi sporočili imenovanimi *heartbeat*.

# Poglavje 4

# Razpodvajanje

Naraščajoče količine podatkov in strožje zahteve po obdelavi narekujejo razvoj novih tehnologij za delo s podatki. V poglavju obravnavamo lastnosti in uporabo razpodvajanja oziroma deduplikacije kot načina prepoznavanja in odstranjevanja podvojenih podatkov.

## 4.1 Uvod

Deduplikacija je metoda za odstranjevanje podvojenih podatkov. Uporablja se za prepoznavanje podvojenih podatkov na podatkovnih nosilcih kot tudi pri optimizaciji pasovne širine računalniških omrežij.

Primarni sistemi za uporabo deduplikacije so strežniki za izdelovanje varnostnih kopij podatkov (*angl. Backup*). Varnostne kopije običajno zasedajo veliko prostora. Pogosto se zgodi, da so shranjeni podatki podvojeni. Vsako nadalnje kopiranje novih podatkov ustvarja dodatno slabo izkoriščenost prostora. Dober primer so v preteklosti prikazovali poštni strežniki v istem podjetju, kjer so veliko prostora zasedale datoteke pripete kot priponke sodelavcem. Običajno je datoteka na strežniku zasedala prostor večkrat, odvisno na koliko naslovov je bila odposlana. Veliko podvojenh podatkov je shranjenih v obliki celotnih datotečnih sistemov, kot npr. Dropbox, OneDrive, itd., oziroma slik na virtualnih strojih pri virtualizaciji.

Deduplikacija deluje za uporabnika transparentno v primerjavi z drugimi metodami optimizacije prostora in stiskanja podatkov. Pri stiskanju uporabnik shrani podatke v arhivske datoteke, ki jih v nadaljevanju stisne s poljubnim algoritmom (*npr. zip, tar, iso, itd*). To ni transparentno shranjevanje.

Pri deduplikaciji se datoteko lahko obravnava celostno (*angl. File Level*) ali po delih (*angl Sub-File Level*). Pri celostni obravnavi datoteke se prebere celotna vsebina. Prebrani datoteki se naredi odtis, njena vrednost se shrani. Odtis se hrani za primerjanje novih datotek. Enak odtis nove vrednosti pomeni enako datoteko, zato se ta ne shranjuje več. Ponovno se shrani samo odtis. Tako se sprosti prostor, ki bi ga sicer zasedala enaka datoteka. Obravnava datotek po delih primerja datoteke po kosih. Datoteka se razdeli na kose(*angl. chunks*) fiksnih (*angl. Fixed Length*) ali spremenljivih dolžin (*angl. Variable Length*). Vsakemu kosu se ustvari odtis s kriptografsko funkcijo SHA-1. Odtisi določajo prave vrednosti, ki se hranijo nepodvojene. Odtisom datoteke pravimo tudi recept.

Deduplikacija se v osnovi deli na sprotno (*angl. online*) in naknadno (*angl. offline*). Vmesna deduplikacija se izvaja sproti med kopiranjem in spreminjanjem podatkov. To pomeni, da se podatki analizirajo sproti kot se ustvarjajo, podvojeni podatki pa obravnavajo takoj. Prednost vmesne (*sinhrone*) deduplikacije je takojšnje shranjevanje podatkov brez podvojenih, ki je hitro in učinkovito. Slabost je zmanjšana zmogljivost sistema zaradi dodatnih računskih operacij. Pri naknadni deduplikaciji pa se podatki obravnavajo kasneje na zahtevo sistemskega administratorja oziroma s skripto, ki požene delovanje ob določenem času. Zaželjena je na strežniških sistemih kjer se obdelujejo velike količine podatkov, in se lahko izvaja ob določenem času (npr. ponoči), ko neposredno ne moti uporabnikov. Slabost je okrnjena uporabnost na (skoraj) polnih medijih, kjer zaradi podvojenih dnevnih podatkov, ni mogoče učinkovito zapolniti prostora v celoti. [16]

## 4.2 Opis uporabljenega algoritma

Deduplikacijo nad podatki je mogoče izvajati na več načinov. Posebnost uporabljenega algoritma je v deduplikaciji vsebine datotek po kosih.

Algoritem prebere in izvaja operacije nad vsebino datoteke. Vsebino razdeli na kose tako, da določi meje s pomočjo pomičnega okna s katerim se premika čez podatke. Čez podatke se premika desno z oknom širine 48 bajtov v intervalu enega bajta. Meje določi na podlagi vrednosti odtisa, ki je primerljiv z vrednosto že shranjenega kosa v bazi ali shrambi. Kadar enak kos v shrambi že obstaja ga ne shrani več. Shrani samo recept v datoteki z odtisi kosov, in predpostavlja, da bo pripadajoče kose našel v shrambi. Pri tem upošteva minimalno velikost kosa 2kB in maksimalno 64kB.

Na začetku izhaja iz povprečne dolžine 4kB, ker so kosi enakih dolžin. Ko pa je teh v shrambi že več se lahko zgodi, da se bo pojavila vsebina objekta, katere kosi bodo zamaknjeni, kot posledica različnega prvega kosa in enakih naslednjih. Algoritem strmi k temu, da zamaknjen kos, ki je spremenjen obravava tako, da bodo vsi ostali kosi podobni že shranjenim v bazi in jih ne bo potrebno shranjevati več. Shranila se bo vsebina prvega kosa, katerega dolžina je drugačna.

Uporabljen deduplikacijski algoritem obravnava datoteko kot objekt z vrednostjo in ključem, ki jo definira. Ključ predstavlja ime datoteke (poimenujemo ga lahko preprosteje, npr. kljuc) in vsebina njeno vrednost v bajtih. Na datotečni sistem se shranjujeta v ločena imenika, ki ga določimo takrat, ko ustvarimo shrambo podatkov.

Ključ objekta <kljuc>.dat se shrani v imeniku `D:/Recipes`. Vsebuje odtise kosov datotek (recepte), katerih prava vrednost je shranjena v shrambi. Vrednost objekta se shranjuje v datotekah <chunks>.dat v imenik `D:/Chunks`. Vsakemu kosu pripadata dve datoteki. Vsebina prve datoteke predstavlja realno vrednost v bajtih. Vsebina druge je števec (tip <int>), ki določa število objektov v shrambi. Zaradi preprostejšega iskanja sta poimenovani šestnajstiško vrednostjo, tako kot je definiran odtis kosa. Datoteka, ki shranjuje števec zaključuje končnica .refcount .

Uporabljen algoritem lahko preprosteje zapišemo v naslednjih vrstnem redu:

1. Algoritem premika okno nespremenljive dolžine $k$ *bajtov* čez podatke po koraku enega bajta. Na vsakem koraku izračuna hash vrednost odtisa.

2. Vedno kadar spodnji $d$ biti ustrezajo vrednosti $r$ ($f \equiv r \pmod{2^d}$) okno določi meje bloka dolžine $k$. Vrednost $D = 2^d$ je delitelj in $r$ ostanek. Velikost bloka se tako določi na vsakih $2^d$ bajtov.

3. Algoritem sedaj poišče podoben odtis med obstoječimi odtisi in ju primerja med seboj, če že obstaja.

4. Kos oziroma blok podatkov se trajno shrani na shranjevalni medij, če enakega odtisa ne najde, sicer govorimo kandidatu za deduplikacijo.

## 4.3 Tehnike razpodvajanja podatkov

Pasovna širina podatkovni poti včasih ne zadošča za predstavitev informacij brez napak ali časovnih zamikov. Podobno delovne postaje ne uspejo procesirati vseh podatkov sočasno. Sledeča razdelka opisujeta preproste rešitve z deduplikacijo pri centraliziranem sistemu z *Blumovim filtrov* in porazdeljenim sistemom z *LBFS*.

### 4.3.1 Blumov Filter

Shrambe podatkov hranijo velike količine podatkov in že uspešno nadomeščajo različne starejše metode snemanja arhivskih zapisov na kasete in trakove. K temu zagotovo pripomore nizka cena podatkovnih medijev za trajno shranjevanje npr. diskov.

Težava nastopi z željo po takojšnji dostopnosti podatkov hranjenih v večjih shrambah. Takojšnji dostopi so mogoči pri računalniških sistemih z zadostno količino delovnega pomnilnika RAM (*angl. Random Access Memory*), ki je sposoben hraniti veliko število naslovov za segmente in kose shranjene na disku. Delovni pomnilnik pogosto ne omogoča zadrževati vseh naslov, zato jih po zapolnitivi prostora zapisuje na disk. Takrat se začnejo pojavljati zakasnitve pri dodajanju novih zapisov na disk, zmogljivost sistema pa se posledično zmanjša.

Blumov filter (*angl. Bloom filter*) je prostorsko nezahtevna verjetnostna podatkovna struktura z vgrajenim mehanizmom bitnega vektorja (*angl. Summary Vector*), ki odpravlja vrzeli pri zakasnitvah delovnega pomnilnika tako, da hrani vrednosti hash shranjnih podatkov na disku [19]. Sistemu preprečuje izvajanje dostopov do diska v primeru, če iskane vrednosti hash za izbrani kos ali segment ne najde v delovnem pomnilniku. Filter nam v splošnem pove, ali se vrednost hash (tudi ID) nahaja v množici ali ne. Pri tem je odgovor DA lahko napačen in NE zagotovo pravilen odgovor. Kadar filter zagotovi neprisotnost vrednosti na disku, potem je tam sigurno ni. Dostop do diska se tedaj ne izvede. V primeru zagotovitve prisotnosti vre-

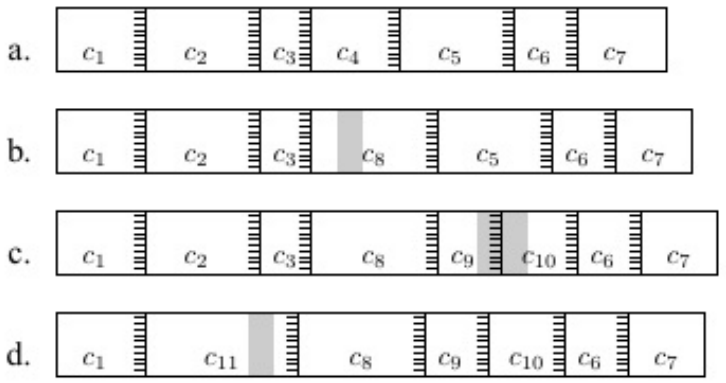
dnosti sistem preveri disk, vendar obstaja možnost, da vrednosti ne bo našel. Različni parametri določajo verjetnost, kdaj je pravilen odgovor napačen.

### 4.3.2 LBFS

Omrežni datotečni sistem LBFS (*angl. Low-Bandwith File System*) je primer porazdeljenega datotečnega sistema za prenos nestrukturiranih podatkov po omrežju pri nizki pasovni širini [20]. Povezavo strežnika z uporabnikom vzpostavljata komunikacijski protokol TCP/IP in asinhronski RPC za prenose datotek, ki deluje po načelu *close-to-open*. To pomeni, da se datoteka na strežniku osveži v uporabo ostalim uporabnikom šele takrat, ko prvi uporabnik spremembe shrani in jo zapre (podobno pri AFS).

Premikajoče okno dolžine 48 bajtov se premika čez podatke objekta. Premika se v desno po intervalu enega bajta z namenom, določanja meje podatkom v kosu. Določena je z desnim robom premikajočega okna na končni poziciji. Meje med kosi predstavljajo podobne okolice podatkov, kot jih najdemo v drugih objektih, kadar so zamaknjeni in imajo mejo premaknjeno za nekaj bajtov v desno. Pravilno postavljene meje v prvotnem objektu olajšajo iskanje podobnih kosov v drugih objektih (Glej 4.1). Deduplikacija nad enakimi kosi je zato enostavnejša in predvsem hitrejša. Ustrezno postavljene meje algoritem določi z izračunano hash vrednostjo nad premikajočim se oknom. Ker okno premika po bajtih je pomembno, da se računanje vrednosti izvaja hitro. Pogosto uporabljen način računanja vrednosti je časovno nezahteven 64 bitni Rabinov odtis.

Pri predvideni naključnost podatkov v kosu, je primerna funkcija tista, ki naključno enakomerno porazdeli svoje vhode. Če so podatki naključni in se okno premika čez njih bo določena vrednost Rabinovega odtisa, npr. 0 (niz ničel) izračunana na vsakih $2^{64}$ premikov v povprečju. Izračunana vrednost 0 pomeni določeno mejo na desnem robu. Z oknom dolžine 48 bajtov (384 bitov) je možnost postavitve meje večkratna, vendar bi bila prepogosta in se zato ne zgodi. Točna meja se definira po modulu povprečne dolžine, npr. 4096 bajtov oziroma $2^{12}$ možnosti. Tako se ustvari nova funkcija, ki vrača




Slika 4.1: Spreminjanje vsebine po kosih in novo določanje mej [20]

12 bitne vrednosti in določa meje na vsakih 4096 premikov v povprečju. Ker pa so podatki v resnici nenaključni se postavi minimalna in maksimalna vrednost za določanje spodnje meje in zgornje neglede na to, če zadnjih 12 bitov izračunana vrednost ni enaka nič.

Tehnika deljenja vsebine datoteke na kose se imenuje TTTD (*angl. Two-Threshold Two-Divisor*) delitveni algoritem. Velikokrat se uporablja kot podtehnika pri algoritmu LBFS zaradi svoje natačnosti pri izbiranju optimalnih kosov podatkov. Algoritem zajema nadgradnjo pomičnega okna BSW (*angl. Basic Sliding Window*) z možnostjo vračanja kosov podatkov različnih dolžin [4].

## 4.4 Sistemi razpodvajanja podatkov

Podatkovna deduplikacija je bistvena in kritična sestavina sodobnih shranjevalnih sistemov za trajno arhiviranje podatkov. Bistveno vpliva na povečanje shranjevalnih kapacitet in izboljšuje delovanje zmogljivosti sistemov. V naslednjih poglavjih sta podrobneje predstavljena sistema *Sparse Indexing* za centralizirano, in *Extreme Binning* za porazdeljeno deduplikacijo podatkov.

### 4.4.1 Sparse Indexing

*Sparse Indexing* je način deduplikacije segmentov [21]. Segment je definiran kot zaporedje kosov (*angl. Sequence of chunks*). Algoritem razdeli zaporedje podatkov na segmente. Najprej se podatki razdelijo na kose (*angl. chunks*) spremenljivih dolžin. Potem se zaporedju kosov dodeli pozicija v segmentu. Segmenti so definirani z recepti [12] imenovanimi *manifesti*. Manifest je struktura, ki vsebuje vrednosti kosov. Uporablja se za enostavnejše primerjanje kosov datotek z drugimi segmenti. Primerjanje omogočajo vrednosti hash s katerimi so označeni kosi podatkov znotraj segmenta. Dva segmenta sta si podobna takrat, kadar vsebujeta enake kose podatkov.

Vhodni segmenti se deduplicirajo ob prvem razvrščanju v shrambo manifestov. Najprej se razvrstijo tisti, ki so najbolj podobni že obstoječim v shrambi in jih imenujemo prvaki (*angl. champions*). Nato se izločijo že shranjeni kosi (na disku) znotraj segmentov identificirani s svojo vrednostjo hash. Preostali kosi se shranijo na disk in prevzamejo vlogo kavljev (*angl. hooks*). Kazalci na njih se hranijo v delovnem pomnilniku. Pri novih vhodnih segmentih se uporabijo za detekcijo podobnih.

### 4.4.2 Extreme Binning

Ekstremno shranjevanje je deduplikativna tehnika izvedena nad datotekami [15]. Pri delovanju ne izkorišča lokalnosti pomnilškega dostopanja, kot običajni shranjevalni sistemi. Dostop do datoteke se izvede v celoti oziroma večkrat

zaporedoma. Obstaja centralizirana različica kot tudi porazdeljena za shranjevanje v gručah.

Deduplikacija z ekstremnim shranjevanjem deluje na naslednji način. Datoteka se prebere v delovni pomnilnik RAM in razdeli na manjše dele. Ustvari se hash vrednost vsebine celotne datoteke s kriptografsko funkcijo SHA-1. Določi se reprezentativni kos ID vhodne datoteke. ID predstavlja začetne bajte vrednosti datoteke, ki jih algoritem v nadaljevanju uporablja za primerjanje. Vrednost hash, representativni vzorec ID in kazalec na kose vrednosti, se shranijo v primarni indeks (*angl. Primary index*), ki je tako določen.

Potem se preveri ali primarni indeks še ne vsebuje podvojenega vnosa (ne obstaja na disku) s primerjanje sekundarnega indeksa (*angl. Secundary index*). Če podvojenega vnosa ne najde, sekundarni indeks ustvari Bin na mediju. Vrednosti kosov datoteke se v celoti prenesejo na disk v indeks Bin. V nasprotnem primeru, ko primarni indeks obstaja algoritem predvideva, da vnos na disku že obstaja. Če se hash vrednosti primarnega indeksa v RAM-u in sekundarnega na disku ujemata, govorimo o najdenem duplikatu. V nasprotnem primeru, ko se vrednosti ne ujemata, se sekundarni indeks prenese v delovni pomnilnik. Neustrezni kosi se zamenjajo in osvežijo vrednosti v sekundarnem indeksu. Obnovljen sekundarni indeks se prenese nazaj na disk.

Kako se bo izvedla deduplikacija je odvisno od prekrivanja vsebine v datotekah in granulacije vsebine. V splošnem velja pravilo o bistveno boljši deduplikaciji, ko so kosi datotek manjši. Ekstremno shranjevanje zato deluje po načelu Broderjeve izreka [11], ki pravi, da je podobnost datotek mogoče opredeliti z deljenjem enakih kosov. Možnost, da sta dve datoteki enaki obstaja takrat, če vsebujeta podobne kose podatkov.

**Uporabnost Extrem Binning-a:**

Spletne statistike prikazujejo trend pospešenega naraščanja različnih informacij. V letu 2007 je količina podatkov presegla 281 exabajtov, pričakovanja

za leto 2011 pa so bila 10-krat večja [8]. Faktor povečanja se je do danes verjetno še povečal. Več kot 35 odstotkov podatkov v obliki dokumentov in ostalih formatov datotek se je nahajalo na strežnikih in porazdeljenih gručah. Lastniki datotek so bile vladne organizacije, podjetja in posamezniki, ki jih uporabljajo pri svojem delovanju. Zaradi velikih količin datotek, kjer so nekatere bile zapisane večkrat, so bile uporabljene deduplikacijske tehnike podatkov. Z omenjenimi tehnikami je mogoče prihraniti od 20 do 30 odstotkov prostora [9].

# Poglavje 5

# Implementacija algoritma shrambe podatkov

V sledečem poglavju je opisan postopni razvoj knjižnice za shranjevanje podatkov z implementirano deduplikacijo na nivoju objektov. Knjižnica se lahko uporablja kot API pri izgradnji prototipnih izdelkov po lastni izbiri uporabnikov ali kot hitra rešitev pri shranjavnju podatkov v ločen imenik. Poglavje se po splošnem opisu rešitve v uvodu, nadaljuje podrobnejšo implementacijo metod in evaluacijo programskega modela delujoče shrambe podatkov. V zaključku se API primerja z obstoječimi nadgradnjami in izdelki, ki so prisotni na trgu.

## 5.1 Uvod

Motivacija z deduplikacijo podatkov je bila izhodišče eksperimentalnega dela. Izdelali smo vmesnik shrambe podatkov, kateri pri shranjevanju uporablja omenjeno tehniko odstranjevanja podvojenih vrednosti objektov. Shramba je slovar, kjer spremenljivo dolge objekte oziroma vrednosti naslavljamo s ključem. Par ključa in vrednosti opisujemo s predstavitvijo <ključ,vrednost> in ga imenujemo objekt. Ključi so lahko spremenljivih dolžin (npr. 1024 bajtov) ali nespremenjivih oziroma fiksnih dolžin. Ključi spremenljivih dolžin

so priročni, kadar se nad shrambo izvede vmesnik klasičnega datotečnega sistema. Polno ime datoteke takrat zavzame vrednost ključa. Vrednost je vsebina poljubno dolgega niza bajtov, ki je pomensko določena s strani uporabnika. Običajno je predstavljen s tabelo bajtov. Zapletenejše shrambe <ključ,vrednost> podpirajo različne operacije kot npr. posnetke, vendar smo se v nalogi osredotočili le na najpomembnejše metode, ki jih potrebujemo za pripravo uporabne knjižnice.

Za realizacijo razširjenega algoritma smo si izbrali programski jezik Java 7, ker ponuja enostavno delo z datotekami in podatkovnimi tokovi. Vsebuje preprosto sintakso in je neodvisen od platforme izvajanja. V poglavju 4 smo opisali način in sisteme, ki deduplikacijo že vsebujejo in jo implementirali v algoritem. Prav tako smo implementirali orodja CLI, ki uporabnikom omogočajo preprosto dodajanje, branje in odstranjevanje objektov iz shrambe. Podrobneje so opisane v razdelku implementacije.

## 5.2 Opis implementacije

Implementacijo naloge smo si zastavili s tremi različnimi modeli shramb. Najenostavnejša shramba podatkov, deduplikacije ni imela implementirane. Namenjena je bila preverjanju ideje o pravilnem shranjevanju objektov v shrambo. Preverjali smo možnost kopiranja datoteke z zastavljenim algoritmom. Izkazala se je kot zelo dobro vodilo pri nadalnjem razumevanju koncepta shrambe in dodajanju novih dodatkov. V nadaljevanju naloge je zato ne omenjamo več.

Različica dve in tri sta namenjeni in prilagojeni za namene preverjanja rezultatov naše naloge. Bistvena razlika med obema je delovanje notranje strukture algoritma, ki vrednosti v objektih razdeli na kose enakih ali različnih dolžin. V tretji različici je deljenje vrednosti objekta mogoče na kose različnih dolžin. Dolžino kosa lahko izbiramo s spremembo vrednosti spremenljivke v glavnem razredu shrambe.

Namen druge in tretje različice je torej ugotoviti pomembnost granulacije

oziroma preveriti, kako vplivajo različne dolžine kosov na učinkovitost deduplikacije. Prepričati se torej želimo, ali je določen objekt bolje shranjevati v manjših kosih in s tem pridelati več metapodatkov v sistemu ali je enostavnje shranjevati večje kose in prihraniti sistemu dodatno delo, zaradi iskanja manjši in pogostejših kosov objektov.

Implementacija obeh različic programov shranjevanlnega sistema shrambe je precej podobna. Bistvene razlike smo že omenili. Vseeno velja poudariti, da je izvedba zadnje različice oziroma tretje kompleksnejša, ker vsebuje dodatno logiko za odmerjanje dolžin vsebini s pomičnim oknom.

V nadaljevanju si bomo ogledali zadnjo različico shrambe, ki vsebuje kompleksnejše metode in vrača boljše rezultate pri shranjevanju podatkov.

### 5.2.1 Osnovni razredi

Strukturo knjižnice shrambe smo definirali v več razredih. Zaradi kompleksne struture smo implementirali vmesnik IShramba, ki je uporabljen v obeh različicah shramb. Z njim smo dosegli, da se imena osnovnih metod ne spreminjajo ozirome se metode redefinirajo v vsakem razredu. Vmesnik IShramba je prikazan na sliki 5.1.

```
public interface IShramba
{
  public boolean dodaj(String kljuc, byte[] vrednost);
  public byte[] beri(String kljuc);
  public boolean brisi(String kljuc);
}
```

Slika 5.1: Struktura vmesnika IShramba

Sedaj poglejmo še tehnično rešitev shrambe. V končni različici knjižnice smo ustvarili razrede, ki smo jih uporabili v naslednji namen:

**IShramba** je vmesnik z osnovnimi metodami.

**Shramba** je osnovni razred, ki vsebuje glavne metode.

**RabinHashFunkction** je razred, ki vrne hash vrednost vsebine.

**UstvariShrambo** je razred z orodjem za ustvarjanje shrambe.

**DodajDatoteko** je razred z orodjem za dodajanje objekta.

**BeriVDatoteko** je razred z orodjem za branje objekta.

**Brisi** je razred z orodjem za brisanje objekta.

### 5.2.2 Uporabljene metode

Predstavljena knjižnica za shranjevanje podatkov je zgrajena podobno vendar veliko preprosteje kot POSIX FS. Objekti so določeni s predstavljeno potjo datoteke glede na izvorno particijo in določajo datotečno strukturo sistema. Metode nadomeščajo sistemske klice in so realizirane v glavnem razredu shrambe. Za osnovno delovanje algoritma smo uporabili naslednje tri metode:

**Osnovne metode razreda Shramba:**

**boolean dodaj(String kljuc, byte[] vrednost)** - Metoda vrne true kadar je objekt uspešno dodan v shrambo, sicer vrne false. To se zgodi v primeru kadar je objekt z enakim ključem že shranjen v shrambi.

**byte[] beri(String kljuc)** - Metoda vrne vrednost objekta, ki je naslovljen s pripadajočim ključem. V primeru, da objekt s pripadajočim ključem v shrambi ne obstaja vrne izjemo z opozorimo o neobstoju objekta v shrambi.

**boolean brisi(String kljuc)** - Metoda vrne true kadar je objekt s pripadajočim ključem uspešno odstranjen iz shrambe, sicer vrne false. To se zgodi v primeru, kadar objekt z pripadajočim ključem v shrambi ne obstaja.

Večino funkcij smo poskušali definiriati znotraj glavnih metod, in s tem zmanjšati reference na druga mesta. Zaradi preglednosti kode se je kasneje pokazalo, da jih je bolje ločevati in jih ob potrebi klicati v glavo metodo.

**Druge uporabljene metode razreda Shramba:**

**byte[] retrieveValue(File recipeFile)** - Metoda vrne vrednost kosa v bajtih, ki ustreza receptu. Kot vhodni podatek sprejme datoteko z recepti originalne datototeke. Vrednosti so shranjene kot rezultat kriptografske funkcije SHA-1 nad kosom vsebine.

**String hexHash(byte[] value, int offset, int len)** - Metoda vrne niz tipa String, ki ga predhodno zakodira s kriptografsko funkcijo SHA-1.

**String readTextFile(File file)** - Metoda vrne niz tipa String, ki predstavlja vrednost datoteke.

**void writeTextFile(File file, String content, boolean append)** - Metoda zapiše vsebino podane spremenljivke tipa string v datoteko.

**void updateReferenceConunt(File RefFile, int n)** - Metoda spremeni vsebino števca, ki je kot številska vrednost zapisana v datoteko s končnico .refcount.

### 5.2.3 Dodajanje objektov

Za omogočeno vstavljanje objektov v shrambo podatkov, je najprej potrebno shrambo ustvariti. Ob klicu orodja *UstvariShrambo* se na poziciji `D:/` ustvarita imenika receptov `D:/Recipes` in kosov objektov `D:/Chunks`. Ko je shramba ustvarjena je dodajanje mogoče.

Dodajanje objekta kot par <kljuc,vrednost> v shrambo je v celoti določeno s predpisano metodo *boolean dodaj(String kljuc, byte[] vrednost)* definirano v glavnem razredu. V metodi je implementiran mehanizem deduplikacije, ki vrednost razdeli na kose različnih dolžin. V nadaljevanju vsakemu kosu izračuna vrednost hash in jih shrani v datoteko receptov na `D:/Recipes/` kot <kljuc>.dat. Recept objekta opisuje datoteke z vrednostjo kosov na poziciji `D:/Chunks/` označenih kot <chunks>.dat. Primer preproste metode za dodajanje objektov je mogoče preveriti v dodatku A.

### 5.2.4 Branje objektov

Kadar je shramba podatkov prvič ustvarjena je njena vsebina v dveh sosednjih imenikih chunks in recipes prazna. Če je objekt dodan, imenika vsebujeta datoteke. Vsebino receptov objektov hranijo ključi v imeniku `D:/Recipes`. Datoteke kosov v sosednjem imeniku `D:/Chunks` hranijo vrednosti objektov. Objekt je definiram kot <kljuc,vrednost>. Če ga v shrambi ni, bo metoda vrnila napako.

Branje objektov je v osnovi branje vrednosti oziroma niza kodirane vrednosti v skupek bajtov določene dolžine. Branje je določeno na podlagi ključa, katerega prisotnost se preverja v shrambi.

Opozoriti velja, da so vrednosti lahko shranjene na dva načina: V prvem primeru kot celotna vsebina, v drugem (našem primeru) po delih. Ker deduplikacija deluje trasparentno za uporabnika ni pomembno kako se bo vrednost sestavljala v celoto temveč enoten in pravilen rezultat, ki ga sam pričakuje. Velja pravilo, da bo vrednost prava natanko tedaj, ko bomo za vračanje uporabili enako strategijo kot smo jo pri dodajanju. Branje vrednosti je mogoče

podrobneje pogledati v kodi dodatka A.

### 5.2.5 Brisanje objektov

Brisanje je omogočeno, ko je shramba ustvarjena in vsebuje shranjen objekt tipa <kljuc,vrednost>. Datoteki `D:/Recipes` in `D:/Chunks` tedaj vsebujeta ključ in vrednost objekta. Vrednosti se določi na podlagi podanega ključa. Velja podobno kot pri branju - algoritem deluje transparentno do uporabnika in odstrani vnos iz shrambe v celoti, neglede na to, ali je vrednost shranjena po delih. Če kljuca in posledično objekta v shrambi ne najde vrne false. Brisanje vrednosti je mogoče podrobnje pogledati v kodi dodatka A.

### 5.2.6 Uporabniška orodja CLI

Z uporabniškega vidika, knjižnica vzpostavlja povezavo z zunanjih svetom s pomočjo metod neposredno iz kode. Tak način je najbolj fleksibilen, saj omogoča implementacijo neposredno v končni izdelek. Ker pa smo želeli izboljšati komunikacijo s širšim krogom uporabnikov, smo dodali še naslednja štiri orodja, ki olajšajo delo s shrambo:

- Ustvari shrambo:

  ```
  java UstvariShrambo <pot do shrambe>
  ```

- Dodaj datoteko:

  ```
  java DodajDatoteko <pot do shrambe><kljuc><pot do datoteke>
  ```

- Beri v datoteko:

  ```
  java BeriVDatoteko <pot do shrambe><kljuc><pot do datoteke>
  ```

- Brisi objekt:

  ```
  java Brisi <pot so shrambe><kljuc>
  ```

Pred prvo uporabo vmesnika je potrebno shrambo podatkov ustvariti. S tem, ko jo ustvarimo, definiramo imenik kamor se bodo podatki shranjevali. To storimo na naslednji način: `java UstvariShrambo D:/`. Imenik lahko izbiramo po lastnih željah in potrebah. V opisanem primeri shranjujejmo datoteko `datoteka.txt` s kljucem `kljuc1` v imenik `D:/`.

**Primeri uporabe:**

Dodajanje: `java DodajDatoteko D:/ kljuc1 C:/datoteka.txt`

Branje: `java BeriVDatoteko D:/ kljuc1 C:/datoteka.txt`

Brisanje: `java Brisi D:/ kljuc1`

## 5.3 Rezultat testiranja

Shrambo smo testirali tako, da smo sešteli vrednosti velikosti datotek v začetnem stanju in jih primerjali z velikostjo shranjenih kosov v imeniku `D:/Chunks`.

Vhodni podatki so zajemali tri datoteke formata .docx z velikostjo 131kB (skupaj 393 kB), ki so se med seboj razlikovale po vsebini, vendar so si bile podobne v vsaj eni lastnosti (dodana slika). Razpodvojene datoteke v imeniku `D:/Chunks` so zasedale 155 kB, cca. 40 odstotkov prvotne vrednosti. Glej tabela 5.1 .

| Datoteke | Velikost | Skupna velikost | Razpodvojena velikost |
|---|---|---|---|
| D1.docx | 131 kB | 393 kB | 155 kB |
| D2.docx | 131 kB | | |
| D3.docx | 131 kB | | |

Tabela 5.1: Prikaz rezultatov razpodvajanja glede na vhodne podatke

V nalogi smo tako na preprost način dokazali, da je mogoče na enakem prostoru shraniti več informacije, če uporabimo napredne metode za shranjevanje podatkov. V našem primeru nam je to uspelo z implementacijo deduplikacijskega algoritma nad datotečnim sistemov z nestrukturiranimi vhodnimi podatki.

# Poglavje 6

# Zaključek

V diplomski nalogi smo podrobneje spoznali metodo deduplikacije podatkov. Posebno pozornost smo namenili nestrukturiranim podatkom in algoritmu, ki deluje nad njimi. Primerjali smo datotečne sisteme nad katerimi se izvaja algoritem in jih v splošnem razdelili na centralizirane in porazdeljene. Seznanili smo se z delovanjem porazdeljenega shranjevalnega sistema Ceph in ga označili kot ciljni sistem za našo rešitev.

Nalogo smo vodili v smeri izdelave knjižnice shrambe za datotečni sistem. API je slovar za shranjevanje podatkov v obliki <kljuc,vrednost> in je namenjen uporabnikom, ki želijo svoji kodi dodati novo funkcionalnost. Končni izdelek smo nadgradili s primernim CLI orodjem, za dodajanje, branje in brisanje objektov. Primerljiv izdelek na trgu je porazdeljena shramba AmazonS3 [14] ali OpenStack Object Storage.

# Dodatek A

V sledečem poglavju je prikazana izvorna koda osnovnih uporabljenih metod vmesnika Shramba (enaki kosi):

## A.1 Metoda dodaj

```
public boolean dodaj(String key, byte[] value) throws Exception
{
  File recipeFile = new File(recipesPath, key);
  if (recipeFile.exists()) return false;
  recipeFile.createNewFile();

  int offset = 0;
  int len = chunkSize;
  while (offset < value.length)
  {
   if (offset + chunkSize > value.length)
    len = value.length - offset;
   else
    len = chunkSize;

   String chunkHash = hexHash(value, offset, len);
   File chunkFile = new File(chunksPath, chunkHash);
   if (!chunkFile.exists())
   {
     FileOutputStream fos = new FileOutputStream(chunkFile);
     fos.write(value, offset, len);
     fos.close();
   }
  updateReferenceCount(new File(chunkFile.getAbsolutePath()
    + ".refcount"), 1);
  writeTextFile(recipeFile, chunkHash + "\n", true);
  offset += chunkSize;
  }
  return true;
}
```

## A.2 Metoda beri

```
public byte[] beri(String key) throws Exception
{
 File recipeFile = new File(recipesPath, key);
 if (!recipeFile.exists())
  throw new IOException("Objekt s podanim kljucem ne obstaja.");
 return retrieveValue(recipeFile);
}
```

## A.3 Metoda brisi

```
public boolean brisi(String key) throws Exception
{
 File recipeFile = new File(recipesPath, key);
  if (!recipeFile.exists()) return false;

  String[] chunkHashes = readTextFile(recipeFile).split("\n");
  for (String chunkHash : chunkHashes) {
  File chunkFile = new File(chunksPath, chunkHash);
  updateReferenceCount(new File(chunkFile.getAbsolutePath()
    + ".refcount"), -1);
  }
  recipeFile.delete();
  return true;
}
```

# Dodatek B

Programska koda se nahaja na: https://lusy.fri.uni-lj.si/svn/dejan-kovac/

# Literatura


[1] A. Silberschatz, P. B. Galvin, G. Gagne: Operating systems concepts, Eight edition

[2] A. S. Tanenbaum, A. S. Woodhull: Operating systems, Design and implementation, Third Edition

[3] A. Tridgell: "The Efficient Algorithms for Sorting and Synchronization"

[4] Moh, T.S., B. Chang.: A running time improvement for the two tresholds two divisors algorithm. In: Preceedings of the 48th Annual Southeast Regional Conference on - ACM SE '10, New York, USA (2010)

[5] F. Douglis, A.I.: Application Specific Delta-encoding via Resemblance Detection. In: Proceedings of the 2003 conference on USENIX Annual technical conference (USENIX'03) (2003)

[6] N. T. Spring, D. Wetherall: 'A protocol independent technique for eliminating redundant network traffic' IN: Proceeding of the 2000 ACM SIGCOMM Conference, Pages 87-95, Stockholm, Sweden, August 2000

[7] Galhardas H. Florescu D., and Shasha D.: An Extensible Framework for Data Cleaning - Retrived October 2003 From http://citeseer.nj.nec.com/galhardas00extensible.html

[8] J. F. Gantz, C. Chute, A. Manfrediz, S. Minton, D. Reinsel, W. Schlichting, and A. Toncheva: 'The diverse and exploding digital universe: An update forecast of worldwide information growth through 2011'

[9] H. Biggar: 'Experiencing Data De-Duplication: Improving Efficiency and Reducing Capacity Requirements, The Enterprise Strategy Group, Feb. 2007.'

[10] M. O. Rabin: 'Fingerprinting by Random Polynomials', Center for Research and Computing Technology, Harward University, Tech. Rep. TR 15-81, 1981

[11] A. Z. Broder: 'On the resemblance and containment of documents: Proceeding of the Compression and Complexity of Sequences' 1997

[12] Tolia N., Kosuch M., Satyanarayanan M., Karp B., Bressoud T. and Perig A.: 'Opportunistic use of content addressable storage for distributed file systems. In Proceeding of the General Track', 2003 USENIX, Annual Technical Conference (San Antonio, Texas, June 2003), USENIX Association, strani. 127-140

[13] A. Tolič, A. Brodnik: Survey of Distributed Storage for Unstructured Data: Technical Report LUSY-2013/1

[14] Amazon S3. Zadnji dostop, dne 27.10.2014, na:

`http://docs.aws.amazon.com/AmazonS3`

[15] Extreme Binning. Zadnji dostop, dne 27.10.2014, na:

`http://cs.ucsb.edu/~vj/projects/SDDS/ExtremeBinning.pdf`

[16] A. Furlan.: Deduplikativni datotečni sistemi. Zadnji dostop, dne 27.10.2014, na:

`http://eprints.fri.uni-lj.si/2760/`

[17] Podatkovna deduplikacija. Zadnji dostop, dne 27.10.2014, na:

`http://en.wikipedia.org/wiki/Data_deduplication`

[18] Porazdeljeni shranjevalni sistemi. Zadnji dostop, dne 27.10.2014, na:

`http://en.wikipedia.org/wiki/Distributed_data_store`

[19] Blumov filter. Zadnji dostop, dne 27.10.2014, na:

`http://en.wikipedia.org/wiki/Bloom_filter`

[20] Low Bandwith File System. Zadnji dostop, dne 27.10.2014, na:

`http://cis.poly.edu/cs623/lbfs.pdf`

[21] Sparse Indexing. Zadnji dostop, dne 27.10.2014, na:

`www.hpl.hp.com/personal/Mark_Lilibridge/Sparse/final.pdf`