# Mot ett distribuerat filsystem

M A R C O  A H U M A D A  J U N T U N E N


KTH Datavetenskap
och kommunikation

Examensarbete
Stockholm, Sverige 2010

# Mot ett distribuerat filsystem

M A R C O  A H U M A D A  J U N T U N E N

Examensarbete i datalogi om 15 högskolepoäng
vid Programmet för datateknik
Kungliga Tekniska Högskolan år 2010
Handledare på CSC var Lars Kjelldahl
Examinator var Mads Dam

URL: www.csc.kth.se/utbildning/kandidatexjobb/datateknik/2010/
ahumada_juntunen_marco_K10072.pdf

Kungliga tekniska högskolan
*Skolan för datavetenskap och kommunikation*

**KTH** CSC
100 44 Stockholm

URL: www.kth.se/csc

# Referat

Beräkning och lagring med datormoln blir allt mer populärt. I detta arbete föreslås en implementation för ett filsystem som är tänkt att operera på ett sådant. Vidare kommer både fördelar och vinster såväl som nackdelar, brister och förluster diskuteras varpå förbättringar till systemet gradvis kommer att föreslås för att överkomma dessa och uttnyttja kapaciteten av ett filsystem i ett datormoln till fullo.

# Abstract

## Towards a distributed filesystem

With the recent growth of interest in cloud computing and storage, this paper will suggest an implementation of a file system designed to run on a computer cloud. Both advantages and benefits as well as disadvantages, losses and flaws will be discussed and solutions will gradually be introduced to improve the design to overcome any shortcomings to harness the capacity of a cloud computing file system.

# Innehåll


**1 Inledning** **1**
1.1 Inledning . . . . . 1

**2 Introduktion** **3**
2.1 Syfte . . . . . 3
2.2 Metod . . . . . 3
2.2.1 Lista över litteratur . . . . . 3
2.3 Definitioner . . . . . 4
2.3.1 Lista av definitioner . . . . . 4

**3 Bakgrund** **9**
3.1 Förutsättningar . . . . . 9
3.1.1 Anmärkningar . . . . . 9

**4 Tes och problematik** **11**
4.1 Tes . . . . . 11
4.1.1 Prestanda . . . . . 12
4.1.2 Robusthet . . . . . 12
4.1.3 Skalbarhet . . . . . 12
4.2 Problembeskrivning . . . . . 13

**5 Det distribuerade systemet** **15**
5.1 Grader av centralisering . . . . . 15
5.1.1 Grundläggande koncept . . . . . 16
5.1.2 Filserverbaserat . . . . . 18
5.1.3 Serverbaserad filtabell . . . . . 18
5.1.4 Distribuerade filtabeller . . . . . 19
5.1.5 Fullständigt distribuerat . . . . . 20
5.1.6 Generell lösning . . . . . 21
5.2 Jämförelse . . . . . 22
5.2.1 Vinster . . . . . 22
5.2.2 Förluster . . . . . 23

**6 Andra system** **25**

6.1 Liknande system . . . 25
6.1.1 Freenet . . . 25
6.1.2 Gnutella . . . 26
6.1.3 BitTorrent . . . 26

**7 Implementation** **27**
7.1 Koncept . . . 27
7.2 Nätverket . . . 28
7.2.1 Noder . . . 28
7.2.2 Anslutning . . . 30
7.3 Filsystemet . . . 32
7.3.1 Filer . . . 32
7.3.2 Systemdata . . . 33
7.4 Förfrågningar och meddelanden . . . 34
7.4.1 Förfrågan om filhämtning . . . 34
7.4.2 Förfrågan om filändring . . . 35
7.4.3 Förfrågan om fildelning . . . 38
7.4.4 Förfrågan om anslutning . . . 39
7.4.5 Förfrågan om borttagning av fil . . . 40
7.4.6 Meddelande om släppning av fildata . . . 41

**8 Förbättringar** **43**
8.1 Näverket . . . 43
8.1.1 Autentisering . . . 43
8.1.2 Användarnamn . . . 46
8.1.3 Registrering . . . 47
8.1.4 Noder . . . 47
8.2 Förfrågningar . . . 50
8.2.1 Förfrågan om filhämtning . . . 50
8.2.2 Förfrågan om filändring . . . 50
8.3 Filsystemet . . . 51
8.3.1 Filrättigheter . . . 51
8.4 Kompromisser . . . 54
8.4.1 Anslutning . . . 54
8.4.2 Anonymitet . . . 54

**9 Diskussion** **57**
9.1 Brister . . . 57
9.1.1 Förfrågnignar . . . 57
9.1.2 Tomt nätverk . . . 59
9.1.3 Disjunkta delgrafer . . . 59

# Kapitel 1

# Inledning

## 1.1 Inledning

Existensen av datormoln är resultatet av den enkla tanken att flera gör samma jobb snabbare om de gör det tillsammans på ett strukturerat sätt. Därför utvecklas det s.k. middleware[1] som utnyttjar kapaciteten hos flera datorer för bl.a. beräkningar och datalagring. Ett grid kan till exempel bestå av en mängd datorer på helt skilda geografiska positioner och låta dem samarbeta för att använda utrymme och kapacitet som kanske inte finns lokalt[2]. Ett filsystem som opererar över ett grid kallas för ett Grid File System (GFS).

Grids kan användas till att utnyttja lagringskapacitet i flera datorer och bättrar på så sätt på konventionell lagring[3] genom att den delar upp filer i fragment över flera värdar i ett nätverk i kontrast till att filerna endast finns tillgängliga i en punkt. Det är lätt att tänka sig att prestandan ökar med s.k. striping då man kan hämta flera delar av en fil med flera parallella strömmar[4] istället för en och samma punkt. Det är datorer som arbetar tillsammans istället för att det är en dator som servar alla andra. Konceptet är enkelt; man lokaliserar fragmenten av den önskade filen i nätet med hjälp av någon slags filtabell för att överföra dem till hämtaren där man sedan bygger ihop datat igen och tanken med detta är att skapa ett filsystem där varje användare har tillgång till en potentiellt väldigt stor mängd data genom att utnyttja utrymme som kanske inte finns på den egna datorn. Filer hämtas parallellt från flera värdar och sätts ihop lokalt till den önskade filen.

Detta kan bland annat jämföras med bittorrent-tekniken[5] där man också hämtar delar av den begärda filen från flera användare simultant. Man får en lista av kända delare genom en eller flera s.k. “trackers” från vilka man sedan hämtar alla filfragment. Denna teknik använder sig av en eller fler servrar som känner till var alla fragment finns tillgängliga, vilket gör systemet sårbart om serverarna själva av

[1] `http://middleware.objectweb.org/`
[2] `http://www.mcs.anl.gov/~itf/Articles/WhatIsTheGrid.pdf`
[3] Se fallet NAS
[4] Se avsnittet om striping i avsnitt 5.1.1
[5] `http://www.bittorrent.com/btusers/what-is-bittorrent`

någon anledning skulle bli otillgängliga. Decentraliseringstekniker för torrents finns i form av Peer Exchange och Distributed Hash Table som avlastar trackern på olika sätt genom att dela ut information om delare till flera värdar.

Det är nödvändigt att implementera liknande decentraliseringar för GFS:er som för torrents så att de är mer eller mindre frigjorda från en server. Beroendet av servern tynger systemet allt eftersom det växer i storlek och det är dessa distribuerade GFS:er är huvudämnet som kommer att diskuteras i följande avsnitt.

# Kapitel 2

# Introduktion

## 2.1 Syfte

Detta arbete skrivs dels som ett examensarbete på kandidatnivå och dels som del av kursen DD143X - Examensarbete inom datalogi, grundnivå - vid KTH i Stockholm.

## 2.2 Metod

Detta arbete är i huvudsak menat att diskutera en abstrakt metod för att över ett nätverk distribuera ett filsystem så att information och kan hämtas parallellt från flera värdar. Den större delen av arbetet har således gått ut på att undersöka vilka metoder som finns för att uppnå önskade resultatet för att sedan föreslå en implementation som ger den grundläggande funktionaliteten som önskas.

När implementationen är färdig läggs förslag fram för att bättra på funktionaliteten eller utvidga den med andra önskvärda funktioner. De brister och tankar som kvarstår efter detta tas upp i det sista kapitlet där implementationen diskuteras.

### 2.2.1 Lista över litteratur

Nedan följer en lista av refererade verk

**Artiklar**

- Ian Foster (2002) - "What is the grid?"
  `[http://www.mcs.anl.gov/~itf/Articles/WhatIsTheGrid.pdf]`
- David A Patterson, Garth Gibson, and Randy H Katz (1988) - "A Case for Redundant Arrays of Inexpensive Disks (RAID)"
  `[http://www-2.cs.cmu.edu/~garth/RAIDpaper/Patterson88.pdf]`
- Hari Balakrishnan, M. Frans Kaashoek, David Karger, Robert Morris, Ion Stoica (2003) - "Looking up data in P2P systems"

[http://www.cs.berkeley.edu/~istoica/papers/2003/cacm03.pdf\\ISSN: 0001-0782]

**Websidor**


- BitTorrent Inc (2001-2010) - "What is BitTorrent?"
  [http://www.bittorrent.com/btusers/what-is-bittorrent]
  Som publicerad 2010/05/01
- Gnufu wiki - "Problems of the FoF-model -> Changes"
  [http://www.bittorrent.com/btusers/what-is-bittorrent]
  Som publicerad 2010/05/01
- Wikipedia (Citation saknas) - "Distributed hash table"
  [http://en.wikipedia.org/wiki/Distributed_hash_table]
  Som publicerad 2010/05/01
- ObjectWeb(Sacha Krakowiak) - "What's middleware"
  [http://middleware.objectweb.org/]
  Som publicerad 2010/06/21


## 2.3 Definitioner

Nedan följer en lista av termer och uttryck som förknippas med arbetet på grund av att de beskriver koncept i eller kring arbetet och är mest tänkt att fungera för att slå upp uttryck som är obekanta under läsningen.

### 2.3.1 Lista av definitioner

**Kapitel 1: Inledning**

Middleware ..... 1

Programvara som förbinder samverkande datorer i ett nätverk. Syftet är att tillföra transparens så att användaren kan använda system trots att det under ytan är distribuerat och är således ett abstraktionslager.

Grid File System (GFS) ..... 1

Ett filsystem baserat på ett grid, där varje dator tillgängliggör en del av sin lagringsyta för de andra datorerna i nätverket. Datornätverkets geografiska utbredning är irrelevant - det kan variera från ett lokalt nätverk till ett världsomspännande nät.

Network-attached storage (NAS) ..... 1

En enhet i ett nätverk vars främsta syfte är att tillgängliggöra data.

Filfragment ..... 1

Delsekvens av en fil. En fil kan delas upp i filfragment och sedan byggas ihop

igen genom att fragmenten sätts ihop i rätt ordning.

Striping ..... 1
Spridning av delar av en fil över flera ytor så att de kan hämtas parallellt från dessa platser.

Fil ..... 1
Logiskt sammanhängande (sekventiell) mängd data av godtycklig storlek.

Filtabell ..... 1
Metod för att slå upp filer i en filstruktur. Strukturen kan finnas på en hårddisk eller på ett, över ett nätverk, distribuerat filsystem. Tabellen kan ses som en funktion som pekar ut en unik (möjligvis tom) mängd positioner för varje unik fil.

Bittorrent ..... 2
Protokoll och teknik som används för att hämta och dela ut filer till en mängd användare.

Torrent ..... 2
Information som används för att lokalisera filer i ett bittorrentnätverk. Innehåller information om vilken fil som ska lokaliseras.

Tracker (Bittorrent) ..... 2
Trackern är hjärtat i bittorrenttekniken. Trackern har en lista över tillgängligheten av data i form av en hashtabell. Varje torrent kan finnas registerad på flera trackers.

Peer exchange (PEX) ..... 2
En teknik för bittorrentklienter som syftar till att avlasta trackern genom att låta klienter fråga anslutna klienter om information istället för trackern eller en DHT. De anslutna klienterna frågar i sin tur sina anslutna klienter o.s.v..

Distributed Hash Table (DHT) ..... 2
En teknik för att distribuera filtabeller över ett nätverk. Används bl.a. i bittorrentnätverk för att avlasta trackers.

Distribuerat GFS ..... 2
Ett GFS utan en enskild central filtabell.

**Kapitel 2: Introduktion**
**Kapitel 3: Bakgrund**
**Kapitel 4: Tes och problematik**
**Kapitel 5: Det distribuerade systemet**
Redundant Array of Independent Disks(RAID) ..... 17

En uppsättning diskar som delar datan genom sig genom spegling eller striping. I RAID-system lagras överflödig (redundant) information i utbyte mot att man kan återställa data efter att en eller flera godtyckliga diskar krashar. Man kan till och med forsätta drift med saknande diskar. Läs- och skrivprestandan i ett RAID-system är också högre än att läsa från en disk med motsvarande lagringsyta.

Query-flooding ..... 18
Sökningsmetod i p2p-nätverk där frågor vidarebefordras av grannar. Metoden är simpel att implementera men fungerar sämre ju fler användare systemet har.

Breddenförstsökning (BFS) ..... 18
En sökning som utgår från en nod och dess grannar varpå den söker igenom grannarnas grannar tills nätverket är uttömt.

Backup ..... 18
Säkerhetskopia av data som används för att undvika dataförluster vid diskkrashar och anda olyckor.

Nod ..... 18
(Här) En Anslutningspunkt i ett datornätverk.

Fillåsning ..... 19
Metod för att undvika kollisioner i samtidiga ändringar av filer. Om en fil är låst kan ingen annan användare skriva till den.

Storage Area Network (SAN) ..... 19
Det nätverk som bildas av ett antal lagringsenther. Kommunikation över ett SAN sker oftast med ett blockbaserat protokoll i kontrast till ett filbaserat protokoll som exempelvis mot en NAS.

Granne ..... 20
I datornätverkssammanhang är en nod granne till varje nod som den är direkt ansluten till.

Route-only-node ..... 21
Nod i ett nätverk som endast vidarebefordrar information och inte delar någon data.

**Kapitel 6: Andra system**
Löv ..... 26
I grafsammanhang en nod som endast är ansluten till en annan nod.

**Kapitel 7: Implementation**
Hashfunktion ..... 33

Funktion som till synes slumpvis förvränger information så att det inte går att känna igen, men resultatet är unikt för varje objekt i en domän. Om samma objekt hashas två gånger är resultatet av funktionen samma värde. En hash fungerar som en envägskryptering på det sättet att datat är oanvändbart för mottagaren, men duger för jämförelser.

Time-to-live timer ..... 34

En räknare som kan associeras med ett objekt och används för att avgöra när det är inaktuellt och borde bortses från.

**Kapitel 8: Förbättringar**
**Kapitel 9: Diskussion**

# Kapitel 3

# Bakgrund

## 3.1 Förutsättningar

Populära applikationer som har implementerats som fullständigt distribuerade är bland annat Gnutella och Freenet. I takt med att grids och datormoln ökar i popularitet tillkommer också allt mer så kallad middleware som är programvara som utyttjar kapaciteten hos en mängd värdar som är anslutna till ett nätverk.

Syftet med detta arbete är dock att ta fram ett förslag på ett decentraliserat filsystem och utröna vad som krävs för att det ska fungera genom resonemang.

### 3.1.1 Anmärkningar

Systemet som kommer att beskrivas nedan är främst tänkt att användas i mindre nätverk, som till exempel ett företags lokala nätverk. Det är dock intressant att undersöka hur ett liknande system skulle fungera för ett världsomspännande nätverk och vad som krävs för att bibehålla en rimlig nivå av prestanda när antalet användare växer.

# Kapitel 4

# Tes och problematik

Här presenterar jag min tes, de synvinklar under vilka det kan vara till fördel med systemet jag tänker beskriva. Jag lämnar här läsaren med en mängd påståenden som jag senare ämnar styrka i avsnitt 5.

## 4.1 Tes

Systemet är tänkt att möjliggöra för en grupp användare att dela på en mängd filer. Gruppen kan tänkas vara sluten eller öppen, men dess funktionalitet kan jämföras med till exempel en NAS eller någon fildelningsteknik som till exempel Bittorrent. Fördelen med systemet är att man kan utnyttja resurser som redan existerar i stället för att köpa en filserver eller andra system för att uppnå samma sak.

Systemet är tänkt att användas av en klientmjukvara som ett abstraktionslager. Applikationen kan exempelvis simulera en disk på operativsystemet så att dess innehåll, filsystemet, kan bläddras igenom som om det vore en egen disk analogt med en via USB eller ethernet ansluten enhet vilket gör användandet transparent och intuitivt för användaren.

Ett av nyckelorden för systemet är robusthet. Även om inte varje aktör i nätverket är tillgänglig ska systemet fungera som vanligt för användarna.

### 4.1.1 Prestanda

System som det här beskrivna delar upp arbete och lagring över flera lagringsytor vilket sparar lagringsyta eftersom det behöver skapas färre dubletter av filerna som delas mellan användare[1]. Istället för att varje användare ska ha var sin kopia av filen sprids den ut i ett fåtal exemplar över nätverket. Man sparar också prestanda eftersom man delar upp uppladdningsarbetet över flera värdar istället för att lägga allt arbete på dedikerade filservrar. Genom att hämta filer från flera platser parallellt kan man också mångdubbla överföringshastigheten och även maskiner med lägre prestanda kan delta i större filöverföringar på samma sätt och under samma förutsättningar som dataöverföring går snabbare från diskar i ett raidsystem[2].

Det blir förvisso en viss overhead beroende på exempelvis hur mycket statistik man vill använda sig av i nätverket men tanken är att statistiken ska hjälpa till att öka prestandan och att den med den intjänade prestandan ska kompensera för detta.

### 4.1.2 Robusthet

Ett system som använder centrala filtabeller är ofrånkomligt sårbart eftersom hela systemet beror på en eller ett fåtal värdar som inte nödvändigtvis är tillräckligt tillgängliga. Det kan därför vara fördelaktigt att införa någon grad av decentralisering och göra systemet helt eller delvis oberoende av servrar.

### 4.1.3 Skalbarhet

Ett mer eller mindre distribuerat system medför också enligt avsnitt 4.1.1 att servern avlastas vilket gör det lättare att upprätthålla prestanda i takt med att nätverket växer eftersom man slipper överbelastning i kritiska punkter. Förhållandet mellan punkter som har ansvar mot den trafik som genereras är en avgörande faktor i hur väl systemet fungerar. Ett fullständigt distribuerat system kan dock likväl generera

[1] Se den senare diskussionen om redundans i avsnitt 5.2.2

[2] Se 5.1.1

så mycket information att noderna inte hinner med att processera denna vilket resulterar i stora prestandaförluster[3].

## 4.2 Problembeskrivning

Vi ställer här några frågor som läsaren ska ha i åtanke under den fortsatta läsningen. Frågorna kommer leda oss till tankar som hjälper oss att lösa problemet.

- Vilka är de grundläggande funktionerna i ett distriberat GFS?
- Vad är det som krävs för att kunna ge en grundläggande funktionalitet?
- Vilka svårigheter kan man tänkas stöta på i implementationen?
- Hur påverkas nätverket när antalet användare ökar?
- Vad finns det för sätt för nätverket att underhålla sig själv så att prestandan bibehålls även för ett större antal användare?

[3]Se Query Flooding

# Kapitel 5

# Det distribuerade systemet

Filsystemet är baserat på ett nätverk där varje deltagare i nätet ses som en nod som förbinds med andra noder i nätverket med någon anslutning (t.ex över Internet eller ett lokalt nätverk). Den grundläggande tanken är att varje användare har en uppsättning grannar genom vilka den kommunicerar med resten av nätverket. Det går att bygga upp systemet på olika sätt, alla med sina för- och nackdelar. Noderna kan ha mer eller mindre ansvar i nätverket, men eftersom målet är ett fullständigt distribuerat nätverk kommer vår ambition vara att alla noder har lika mycket ansvar vilket innebär full decentralisering. Vi börjar med att diskutera olika nivåer av decentralisering.

## 5.1 Grader av centralisering

Här ges exempel på extremfallen av centralisering samt ett mellanting. Vi diskuterar den relativt rättframma serverbaserade modellen, den mer komplicerade fullständigt distribuerade modellen samt något som kan ses som en blandning därav, nämligen distribuering av filtabeller.

Vi gör också en kort jämförelse med NAS-baserade modellen som är något av en invers till det vi försöker uppnå.

### 5.1.1 Grundläggande koncept

Innan vi börjar diskutera implementationer ska vi dock se de fundamentala koncept som krävs för att kunna jämföra de olika systemen.

#### Filtabellen

Varje system bygger på att det finns något sätt att lokalisera det data man är ute efter. För att effektivt och pålitligt kunna göra detta krävs att det unikt går att identifiera datamängden. Samma identifierare kan alltså inte peka på två olika datamängder, eftersom det skulle leda till tvetydlighet.

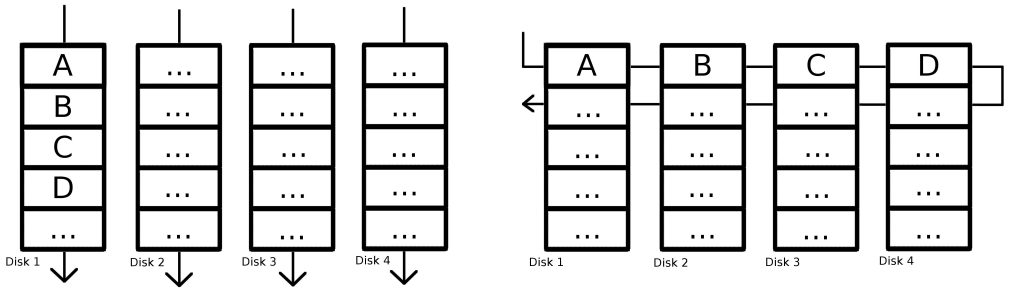
I resten av arbetet antas, om inget annat anges, filer vara uppdelade över flera ytordato i nätverket varför filtabellen kan peka på flera olika platser i nätverket där varje värd tillgängliggör delar av den önskade filen.

Varje nod kan då också tänkas ha lokala filtabeller som är förteckningar över var data förknippat med en viss identifierare finns lagrat.

#### Striping

När man har flera fysiska ytor att lagra filer på kan man strukturera den på ett sådant sätt att man kan skriva och hämta data parallellt, så länge man har en tillräckligt snabb processor. Man får då ut högre överföringshastigheter än om man endast skulle ha en disk eftersom man medan en disk är upptagen med att läsa eller skriva kan ge direktiv till nästa.

Vi ser i figur 5.1a en mängd diskar som inte använder striping. För att läsa datasekvensen ska vi läsa fyra block från disk 1. För varje diskläsning måste vi vänta på att disk 1 har läst färdigt och överfört sin data innan den kan börja utföra nästa läsning. Om vi då har en processor som är snabbare än disken kommer kommer vi ha slösat tid som kunde gått åt till att utföra nästa läsning.




(a) Lagringsytor som inte använder striping (b) Lagringsytor som använder striping

Figur 5.1: Lagringsytor med och utan striping

I figur5.1b kan vi börja läsningen i disk 1, och medan den pågår påbörja läsning från disk 2 och på så sätt slutföra läsningen under en kortare tid eftersom processorn kan utnyttja dödtiden mellan läsningarna till att hantera inkommande data från diskarna samtidigt som läsningar sker i andra.

Skrivning till stripade diskar går på motsvarande sätt snabbare. Istället för att vänta på att en disk skrivit färdigt kan vi ge andra diskar direktiv att skriva nästa del av datasekvensen.

Det finns tekniker för att återställa data efter diskhaverier och till och med ha fortsatt drift. Exempel på sådana uppsättningar av diskar är RAID-system[1].

Här har lokala diskar använts i illustrationssyfte, men principen kan tillämpas för ytor förbundna över ett nätverk. För att uppnå bättre hastighet krävs att den hämtande parten kan ta emot data i samma takt eller snabbare än de sändande parterna kan sända data.

### Sökmodell

Sättet en nod i nätverket lokaliserar andra noder på kan variera stort och påverkar vilket data som bör lagras i noderna. De flesta sökmodeller baseras på så query-

[1] `http://www-2.cs.cmu.edu/~garth/RAIDpaper/Patterson88.pdf`

flooding eller distribuerade routingtabeller (DHT).

Den tidigare av dem går ut på att sökningar i nätverket går genom varje granne och på så sätt propagerar ut i nätverkets periferier likt en bredden-först sökning. Alternativet är att varje nod innehåller information om hur sökningen kan styras till en nod som är närmare resultatet vilket kräver regelbundna kontroller av nodens grannskap.

### 5.1.2 Filserverbaserat

Filseverna är det 'klassiska' sättet av dela ut filer över ett nätverk. Det finns en eller flera dedikerade servrar varifrån alla behöriga kan hämta filer och lagra sina ändringar. Varje fil finns således typiskt endast i en punkt i nätverket. Filservern kan för visso använda sig av ett RAID-system bakom, men överföringen av data över nätverket sker ändå bara mellan två punkter. Filservrar kan också tänkas ha fler nätverksportar för att snabba på överföringen över nätverket, samt uppgraderas efter behov när bördan från användare ökar. De närbesläktade konceptet NAS fungerar i princip likandant och samma resonemang gäller det. Filservrarna är de enda som delar ut filer och har således endast filtabeller över de filer som de själva delar ut och inga andra. Om en filserver skulle blir otillgänglig blir också all data som den lagrar det. Skulle den haverera försvinner också all data den lagrar, om man inte förutsätter en backup. Fördelarna med en filserver är att ändringar i filer endast behöver ske på ett ställe om man inte antar en backupfunktion och att det är enkelt att låsa filer för läsning.

### 5.1.3 Serverbaserad filtabell

Likt bittorent är alla noder anslutna till ett enda nav. All information om var filerna finns att tillgå finns på ett ställe och kan på så sätt vara snabbt att få tag på, men allt eftersom nätverket blir större blir bördan större för navet. Fördelen är att uppdateringar av filtabellen därför också blir triviala. En stor nackdel med systemet är dock att nätverket beror helt på en enda punkt, och mycket arbete

kan potentiellt behöva läggas ner på tillförlitligheten av detta nav. Den centrala filtabellen har dock inga egna filer utan pekar till platser där filerna kan hittas, och fungerar på så sätt mer som ett index. Man kan lätt införa redundans i systemet genom att sprida ut kopior av data över flera och sedan ange varje plats den finns på motsvarande post i filtabellen.

Precis som med en central filtabell är det också trivialt att meddela om filändringar och att exklusivt låsa filer och införa policies för samtidiga ändringar av filer.

Det är också tämligen uppenbart hur anslutning till nätverket skall ske; den anslutande noden kan registera sig mot servern och kan då få tillgång till informationen[2] och registerar de filer den gör tillgänglig. Alternativt kan någon annan nod i servern kontaktas som då vidarebefordrar den anslutande parten till servern eller en kombination därav.

Denna metod är besläktad med SAN (Storage Area Network) på så vis att lagringsytorna sinsemellan bildar ett nätverk. Den serverbaserade filtabellen går då också att använda som ett nav för filservrar i något som liknar ett NAS-SAN.

### 5.1.4 Distribuerade filtabeller

Delar man upp informationen i filtabellen över flera noder delar man belastningen och har på samma gång fortfarande informationen någorlunda centralt. Fördelen här är fortfarande att informationen finns tämligen nära alla noder, men medför också att det fortfarande finns kritiska punkter i nätverket. Om filtabellerna inte kan överlappa varandra är uppdateringen fortfarande trivial, men om samma information kan tänkas finnas på flera ställen måste alla informationsnoder meddelas om ändringen. Alla förfrågningar av filer kan då potentiellt passera samtliga informationsnoder genom någon väg i nätverket.

Proceduren för fillåsning är i fallet med överlappande filtabeller mer komplicerad, men fortfarande relativt enkel förutsatt att alla informationsnoder lätt kan

[2]Se fallet bittorrentklient och tracker

nås och uppdateras någorlunda parallellt, dock kan ändå konflikter uppstå om två noder krävt att få låsa en fil samtidigt.

Anslutning till nätverket är likartat det tidigare fallet där man kan ansluta direkt till någon av informationsnoderna. Det är också tänkbart att det finns en central server som är tillägnad att ta hand om inkommande anslutningar till nätverket och eventuell autentisering av klienter på ett lämpligt sätt.

### 5.1.5 Fullständigt distribuerat

I detta fall är alla noder är likvärda. Eftersom det inte finns något nav i nätverket bör varje nod känna till en mängd andra noder, vilka då blir nodens konceptuella "grannar".

Här är filtabellen fördelad över hela nätverket, och varje nod har här en egen filtabell som kan hålla information om en delmängd av filerna i nätverket. Filtabellen för varje nod kan t.ex. innehålla

1. Filerna som dess närmaste grannskap håller i (Routingtabeller)

2. Filerna som noden själv håller i

Det första fallet kan man tänka sig att noder ställer frågan "Känner du någon som har fil X?" till varandra. Detta kräver dock att varje nod hålls uppdaterad på alla ändringar i sina grannoder för att filtabellen ska vara aktuell. För att göra kommunikationen lättare borde dock ambitionen vara varje nod i grafen ska vara helt självständig och bara bryr sig om sin egen data. Frågan är nämligen väldigt lik "Kan du fråga någon granne om de har fil x?". Denna ansats visar sig dock vara naiv. Vi kommer dock fortsätta anta att inga filer har routingtabeller tills vi utvidgar eller förbättrar implementationen.

I varje fall måste varje nods egna filtabell forfarande uppdateras så fort det sker en ändring någon av filerna den delar. Detta sker genom att noden, där ändringen sker, lokaliserar varje nod i nätverket som delar ut fragment av den ändrade filen och meddelar att den versionen de nu håller i är inaktuell varpå noderna kan ta relevanta

åtgärder. Hur noden faktiskt lokaliserar filfragmenten beror på sökmodellen som systemet använder.

Anslutning till nätverket kan eftersom noderna är likvärda ske till vilken nod i nätverket som helst, varpå den nyanslute tilldelas en mängd grannar.

### 5.1.6 Generell lösning

En intressant observation är att de tre tidigare lösningarna kan ses som specialfall av det fullständigt distribuerade systemet.

Antag att det endast finns en nod i nätet som delar någon information, och de andra noderna enbart hämtar sin data från denna. Detta skulle fungera analogt med en NAS eller en filserver. Ponera att det på samma sätt finns en nod som endast routar förfrågningar, och att den använder filtabellen som anger alla grannars filer.

För system som använder distribuerade routingtabeller betyder det att varje nod kan peka ut data så att det ligger i den delande noden. I det andra fallet gäller att varje nod routar användaren vidare till indexnoden. Motsvarande för query-flood-metoden är att varje nod är granne med den centrala noden i det första fallet, och helst ingen annan eftersom frågorna då kommer direkt till den delande noden och all data kan hämtas därifrån om man då vänder på förhållandet så att den centrala noden inte delar något, kommer den endast att vidarebefordra frågorna till alla sina grannar, vilket är alla utom den frågande noden.

Om man istället låter ett något större antal route-only-noder har vi något som liknar distribuerade filtabeller. Det är då tydligt att ett system enligt ovanstående beskrivning som kan modiferas med enkla regler så att man kan få ut en prestanda och funktion som är närapå dem man får ut ur dessa tre exempel och antagligen varje mellanting.

## 5.2 Jämförelse

Vi jämför här olika aspekter med att ha ett fullständigt distribuerat filsystem jämfört med de tre ovannämnda.

### 5.2.1 Vinster

#### Prestanda

Varje nod jobbar mindre per sändning av data, och skickar dessutom en mindre mängd fildata jämfört med en NAS per filsökning så arbetet är utdelat över nätverket även om det totala arbetet för att överföra en fil är större. Parallell hämtning kan också snabba på hämtningsprocessen.

#### Pålitlighet

Nätverket är inte beroende av en eller ett fåtal punkter. Det krävs mycket för att nätverket ska sluta fungera. Tillgängligheten beror på hur bra algoritmer som finns för att se till att det finns tillräckligt stor redundans av data för att ingenting ska försvinna.

#### Kostnad

Ett företag som har många arbetsstationer med outnyttjat diskutrymme kan utnyttja denna genom att ansluta alla sina arbetstationer till nätverket och låta dem delta i filsystemet, eller komplettera filservern med det utrymme som finns på arbetsstationerna.

#### Säkerhet

Om en enhet i nätverket blir stulen finns det inget sätt att utvinna information från nätverket eftersom det bara är slumpmässiga filfragment.

### 5.2.2 Förluster

#### Overhead

Det finns en viss overhead för att hantera frågor och blir på så sätt trafik som inte gynnar den enskilda noden eftersom de flesta noder i en sökning inte hämtar en fil utan vidarebefordrar den. Jämfört med en filserver är det också mycket information som inte är direkt filinformation som sänds och tas emot av varje nod. Algoritmer som baserat på statistik ska förbättra prestandan i nätverket såsom grafbalansering och tillgänglighetskontroller producerar även mer overhead-data.

#### Fillåsning

Det är komplicerat att implementera fillåsning eftersom filen är slumpmässigt spridd över nätverket.

#### Redundans

För att öka tillgängligheten av filer krävs att man duplicerar filer dynamiskt över flera värdar, vilket ökar mängden trafik som krävs samt lagringsutrymme. För att se till att varje fragment av en fil finns tillgängliga behövs procedurer dels för att avgöra om någon fil behöver distribueras mer, och som utför distributionen. Om en fil har många användare i förhållande till hur många exemplar den är spridd i över nätverket är det dock troligt att redundansen snarare är fördel för nätverket.

#### Fördröjning

Eftersom varje fråga måste färdas genom nätverket uppstår en viss fördröjning vid filhämtning som inte uppstår i andra situationer. Vid hämtning av små datamängder går motsvarande hämtning snabbare i en NAS eftersom som lokalisering sker fortare.

# Kapitel 6

# Andra system

Här diskuteras andra, liknande system, eller i varje fall system som har samma mål. För att kunna jämföra ett system med ett annat måste man först klargöra vilka mål systemen är tänka att uppfylla.

## 6.1 Liknande system

### 6.1.1 Freenet

Freenet är ett nätverk vars syfte är att ge dess användare full anonymitet och är tänkt att främja tryckfrihet och att vara omöjligt att censurera. Varje användare ska vara fullkomligt anonym, men Freenet använder en underliggande struktur som liknar den som kommer att presenteras nedan.

Freenet använder sig av distribuerade hashtabeller för att routa sökningar till noder som troligen är närmare. För att bibehålla användarnas anonymitet drabbas dock sökningsalgoritmen så att den inte garanterat hittar filer[1].

Det i dokumentet beskrivna systemet har ingen avsikt att låta användare vara anonyma, och systemen har inte allt för många likheter förutom sättet att lokalisera delare av filer och att all data är distribuerad över nätverket. Det finns därför en stor skillnad i avsikterna med systemet, men det har intressanta aspekter särskilt

[1] `http://en.wikipedia.org/wiki/Distributed_hash_table`

för fildelning på grund av rädsla för de juridiska konsekvenser som kan förknippas med den.

### 6.1.2 Gnutella

Gnutella anses vara det första populära fullständigt distribuerade fildelningsnätverket. Gnutellas mål är väldigt likt det som fastställs i detta arbete, men är inriktat på att göra filer tillgängliga för en bred publik och inte att implementera ett komplett filsystem. Gnutella har således inte filrättigheter utan varje fil som delas kan hämtas av varje användare.

Till en början implementerade Gnutella query-flood för sökning, men efter observationer att trafiken ökade drastiskt med antalet användare tvingades man utvidga protokollet så att nyansluta noder fick utgöra löv  och andra noder, med mer kapacitet, fick utgöra så kallade *ultranoder* som routade sökningar mellan löven analogt med distribuerade filtabeller vilket medförde att systemet skalade bättre mot mängden användare[2]. Därmed hade projektet i praktiken det problem som detta arbete tänker ta upp i teorin.

### 6.1.3 BitTorrent

Likt Gnutella är huvudmålet med BitTorrent att göra filer tillgängliga för en bred publik och är ett populärt protokoll för fildelningsklienter. Precis som Gnutella och det i dokumentet föreslagna systemet hämtar den data från flera delare parallellt, men beror traditionellt på en tracker för att lokalisera delare. Senare har det också tillkommit en DHT-implementation som möjliggör för användare att finna delare utan trackers.

Skillandena mellan dessa system är att man måste ha information utifrån i form av torrent-filen för att kunna hämta de filer man vill ha. Det är alltså inget komplett filsystem utan endast ett verktyg för att hämta och dela enstaka filer.

[2] `http://rakjar.de/gnufu/index.php/GnuFU_en#Problems_of_the_FoF-model_-.3E_Changes`

# Kapitel 7

# Implementation

Kapitlet beskriver ett förslag på en implementation av ett distribuerat filsystem. Implementationsförslaget är till en början inriktat på att göra systemet fullständigt distribuerat för att visa på eventuella brister, svårigheter och problematik för att sedan ta upp dessa och föreslå förändringar och kompromisser för att försöka överkomma dessa i kapitel 8.

## 7.1 Koncept

Systemet är baserat på ett nätverk som har ett godtyckligt antal användare som alla delar ut lagringsyta. På denna yta lagras den data som är gemensam för nätverket och på så sätt gör den tillgänglig för andra användare av nätverket. Användare kan också hämta och ändra data som andra användare gjort tillgängliga. Aktörerna i nätverket kallas för noder, och kan vara en användare eller en autonom enhet. Noderna i nätverket är anslutna till varandra över till exempel ett ethernet eller ett internet och delar information genom att skicka förfrågningar om upp- och nedladdning av filer mellan varandra, men varje givet par av noder är nödvändigtvis inte anslutna till varandra vid varje givet tillfälle. I själva verket är det en balansgång att avgöra hur många sådana 'grannar' som är lämpligt att ha.

Alla noder i nätverket är likvärdiga vilket är tänkt att ge nätverket en robusthet

genom att nätverket bibehåller sin funktionalitet även om inte varje nod är tillgänglig vid alla tillfällen. Ingen nod är därför i princip viktigare än någon annan för att driva nätverket[1]. Anslutning till nätverket kan ske genom vilken redan ansluten nod som helst eftersom alla noder är likvärdiga.

Kommunikation i nätverket sker genom att en nod ställer förfrågningar till alla sina grannar vilka sedan vidarebefordrar den till alla sina, på vilket sätt varje nod i nätverket slutligen tagit del av frågan. Detta kallas query-flooding, och ansatsen är medvetet naiv[2]. Metoden för sökning ska ses som en del av ett protokoll som är utbytbart.

## 7.2 Nätverket

### 7.2.1 Noder

Varje nod representerar en aktör i nätverket och den data som ska finnas tillgänglig på nätverket. Här diskuteras den grundläggande funktionaliteten som varje nod kräver för att nätverket ska fungera.

#### Grannar

Varje nod har en uppsättning grannar genom vilka den kommunicerar med resten av nätverket. För att balansera prestandan i nätverket borde varje nods grannskap bestämmas dynamiskt - har man för många grannar ökar arbetet per nod, och med för få måste frågorna färdas längre sträckor och återanslutning till grafen blir svårare. Vi låter antagandet gälla att en nod måste ha minst två grannar eftersom man då bildar en väg mellan två noder[3]. För vidare diskussion och ett förslag på ett mer sofistikerat sätt att uppdatera sitt grannskap se stycke 8.1.4.

[1] I princip, därför att det faktiskt möjligt att data endast finns på en nod, även om målet är att undvika detta

[2] Se kapitel 8 och 9 för en mer invecklad diskussion

[3] Naturligtvis förutsatt att det finns minst tre noder i grafen.

### Cache

Varje nod har en intern filtabell som innehåller de filer som den gör tillgänglig genom att dela något eller några av dess fragment. Filtabellen har en post för varje filfragment där den lagrar åtminstone följande information:

**Fil-ID** Den unika identifieraren för den fil som fragmentet tillhör.

**Fil-datum** Det senaste tillfället denna fil ändrades.

**Fragmentnummer** Fragmentets ordningsummer i filen den tillhör.

Informationen om fil-ID används för att unikt identifiera en fil på nätverket. Vid en efterfrågan om en fil används denna information för att avgöra om användaren har filen varpå filen kan påbörja en sändning tillsammans med fragmentets ordningsnummer som mottagaren kräver för att bygga ihop filen.

Filens datum är ekvivalent med filversion och används för att avgöra om en fil är inaktuell eller inte. Om någon efterfrågar en fil med ett gammalt ID svarar noden med att be den uppdatera sin information. Vi bygger vidare på nodens funktionalitet i 8.1.4 genom att låta den lagra lite mer information för att förbättra filintegritet och prestanda i nätverket. Endast ett versionsnummer är inte tillräckligt, utan datumet krävs för att kunna avgöra om konflikterande ändringar införts.

### Frågekö

I fallet query-flooding kan förfrågningar nå en nod genom flera olika vägar. För att förhindra att någon fråga hanteras flera gånger har varje nod en lista över frågor den nyligen hanterat. Det krävs därför ett sätt att unikt identifiera även frågor. Frågorna placeras i en kö när de kommer in, hanteras och rensas sedan ur kön när de blir inaktuella. Eftersom en nod inte kan hantera hur många frågor som helst samtidigt fungerar frågekön på samma gång som lagring för frågor som kommer att hanteras av noden.

När en nod får en fråga som den redan hanterat kastar den helt enkelt bort frågan eftersom dess grannar redan tagit del av den. På detta sätt kommer frågor bli inaktuella och så småningom dö ut, förutsatt att noderna lagrar frågan tillräckligt länge i sin kö. Om noden hinner ta bort frågan ur kön innan den återkommer kommer den sändas på nytt i nätverket vilket resulterar i att frågan kommer fortsätta eka vidare i nätverket.

#### Utgående filer

För att undvika att data förloras i ändringskonflikter[4] behåller en nod en lista av utgående filer. Tabellen fungerar som den ordinarie filtabellen, men här lagras också information som anger av vilken anledning filen är utgående. Filsökningar går först genom ordinarie filtabellen och därefter tabellen med utgående filer. Finns filen i den senare, meddelas den sökande noden om att filen är utgående och varför. Undantag gäller om filen ska släppas då frågan ska ignoreras. En fil kan vara utgående av tre anledningar:

**A)** Någon har begärt att filen ska tas bort ur systemet

**B)** Det finns en nyare version av filen tillgänglig

**C)** Någon har begärt att noden ska släppa filen

När noden beslutar att filen inte efterfrågas tillräckligt ofta i annat fall raderas den ur tabellen och försvinner sedan för gott ur nodens lagringsyta.

### 7.2.2 Anslutning

Implementationen beror inte på hur anslutning till nätverket går till, utan det lämnas till klientmjukvaran som opererar på nätverket. Anslutning till nätverket kan dock ske genom vilken redan ansluten nod som helst, varpå den nyanslutna noden tilldelas en uppsättning grannar. När en användare lämnar nätverket behåller den

[4]Se 7.4.2

informationen för att ansluta till noderna den senast var ansluten till. Dock finns det här en uppenbar brist: vad händer om de gamla grannarna är otillgängliga vid nästa anslutningstillfälle? Detta diskuterar vi vidare i avsnitt 8.4.1.

### Cachekontroll

När en nod ansluter till ett nätverk börjar den med att kontrollera sina filers aktualitet. Detta kan göras genom att man gör en sökning för varje fil man delar. Om någon fil är utdaterad ersätter man sina filfragment från de noder som svarat på sökningen. På detta sätt kan man se till att ens filer alltid hålls uppdaterade och att man inte delar inaktuell information.

Ett problem med detta är noder som ansluter till nätverket efter en så lång tid att dess data har blivit helt inaktuellt. Alla noder har då tagit bort data om dess inaktuella filer från sin lista av utgående filer och det finns ingen nod som kan meddela den anslutande noden om att filerna faktiskt är inaktuella. Noden kommer då att fortsätta hålla i data som inte kommer att användas. En möjlig lösning till detta är att den anslutande parten kontollerar katalogerna som dess filer ligger i och ser om någon fil har raderats varpå den lägger till filen bland sina utgångna filer. Ett annat alternativ är att en nod själv släpper de filer som inte verkar finnas kvar på nätverket om den avgör att den varit offline för länge och effektivt förnyar sin cache för säkerhets skull.

### Hämtning av rotkatalogen

Direkt vid anslutning hämtar varje nod systemets rotkatalog för att kunna påbörja bläddringen i nätverket. Rotkatalogens ID bör därför vara sådant att varje nod i grafen känner till den, förslagsvis tilldelas den vid anslutning till nätverket eller att dess ID är en konstant som är specifik för nätverket, som exempelvis nätverkets namn eller liknande.

## 7.3 Filsystemet

Filsystemet är uppbyggt efter en katalogstruktur. Varje fil pekas ut av en annan, med undantag för rotkatalogen. De filer som pekar ut andra filer kallas kataloger, analogt med de flesta operativsystem. Utöver vanliga filer lagras systemfiler som krävs för att systemet ska kunna fungera. Innehållet i dessa systemfiler är olika beroende på vilken funktionalitet de uppfyller.

### 7.3.1 Filer

Data i nätverket är logiskt uppdelat i filer, så att varje fil består av ett antal fragment. Man efterfrågar alltså filer, och får som svar en uppsättning av fragment som ska byggas ihop till filer. Till varje fil finns det en systemfil som innehåller dess metadata, som till exempel dess filnamn. Se avsnittet om systemdata nedan.

#### Filrymd

Varje fil tillhör en filrymd som bestämmer dess egenskaper och hur de hanteras i olika situationer. De vanliga datafilerna tillhör en filrymd och deras metadata hör till en annan.

Det finns i denna grundläggande implementation tre filrymder:

**Metadata** En fil som innehåller data om andra filer.

**Datafil** En fil vars innehåll varierar och är den information som är mest intressant för användaren. Även katalogerna hör till denna kategori.

Det kan tänkas finnas flera sådana filrymder och därför har utrymme lämnats för att tillåta definition av nya filrymder i framtiden.

#### Fil-ID

Varje fil har ett namn som ska kunna läsas av användarna, men filerna ska unikt kunna identifieras och då duger inte enbart filnamnet. Man måste också ta hänsyn

till att filen ska kunna flyttas och ändå behålla sitt ID, så enbart sökväg fungerar inte heller. Ett lämpligt ID är dock hashen av filens ursprungliga sökväg, det datum som filen skapades konkatenerat med något värde som represtenterar dess filrymd. Två filer av samma typ med samma namn kan inte skapas i samma sökväg samtidigt och därför är ett ID unikt. Om detta mot förmodan inträffar kan konflikten hanteras. Läs 7.4.3

En fördel med att ha en hash som filnamn är att trafiken blir svårare att tyda och en utomstående har mindre möjlighet att utröna vad som skickas mellan punkterna i nätverket.

Anledningen till att ha filrymden med i ett fil-ID är att det blir lättare att associera filer i olika filrymder. Till exempel metafiler till vanliga filer och kataloger. Om en metafil har samma hash som filen eller katalogen den är associerad till med undantag från den del av strängen som anger filrymden blir det lättare att lokalisera informationen.

Hashalgoritmen som används är av betydelse här. Ju längre hashkod man har, desto mer data måste sändas, och ju kortare den är desto större blir risken för hashkollisioner som man inte kan återhämta sig ifrån. Om man exempelvis använder algoritmen SHA-256 för att hasha katalogen admin under rotkatalogen med klockslaget 1302167231 som är antalet sekunder som passerat sedan den första Januari 1970. Resultatet (“39ab46bc248a242c36129a376b2dba5df53c09ccec775709bd6af3ffd684157600”) avslöjar inget för användaren men pekar ändå exakt ut en fil i systemet precis. De två sista tecknen anger filrymd medan resten av hashen är filhashen.

### 7.3.2 Systemdata

Beroende på vilken funktionalitet man vill ha ur systemet finns det olika uppsättningar systemfiler där varje typ av systemfiler utgör en egen filrymd. Deras huvudfunktionalitet i denna implementation är dock att ge information om hämtade filer. Utökad funktionalitet som autentisering och filrättigheter kräver också systemfiler

i fler filrymder, men de är inte ett minimikrav och tas därför upp i nästa kapitel eller lämnas till vidareutvecklingar av implementationen.

## 7.4 Förfrågningar och meddelanden

Kommunikation över nätverket sker med förfrågningar och svar på dessa frågor samt meddelanden mellan noder. Det finns en bred uppsättning frågor i denna grundläggande implementation. I varje fråga finns förutom informationen som krävs för att avgöra om man ska svara på frågan eller inte information för att unikt kunna identifiera frågan. Varje fråga har dessutom en TTL-räknare som räknar ner med ett tal som dynamiskt bestäms för att inte belasta nätverket mer än nödvändigt, men ändå är tillräckligt lång för att nå fram till tillräckligt många.

### 7.4.1 Förfrågan om filhämtning

När en part vill ha data för en viss fil skickar den ut en förfrågan om en fil. Enkelt översatt kan den tolkas som “Har du eller någon av dina grannar denna fil? Svara då till denna adress”. Denna förfrågan innehåller följande information:

**Svarsadress** Anslutningsinformation för att svara på frågan.

**Fil-ID** Filen till vilken data begärs.

**Fil-Datum** Versionen av filen som begärs.

**Frågedatum** Tidpunkten då frågan ställdes.

Detta är den minimala informationen som krävs för att en fil ska kunna överföras och hamna på rätt plats samt inte bli kvar allt för länge i nätverket.

#### Svar

När en nod mottar en fråga om filhämtning för en fil som den har, ansluter den till den efterfrågande noden med den tillhandahållna adressen och påbörjar överföringen

av filen med information om vilket fragment det rör sig om. Överföringen kan här avbrytas av den mottagande parten om det av någon anledning inte skulle vara aktuellt längre. Svaret innehåller följande information:

**Mottagaradress** Adressen från vilken filen kan hämtas.

**Fragmentmängd** En mängd intervall av de fragment som finns tillgängliga.

Den svarande parten kan dessutom välja att skicka ett felmeddelande till den frågande noden om exempelvis den efterfrågande filen har en gammal version. Om parten som då ställde frågan upplever felmeddelanden kan den välja att uppdatera sin filinformation och göra om sökningen. Svarande noder som håller i en gammal version av filen kan här också välja att uppdatera sin information.

**Feltyp** Vilken typ av fel som inträffat

**Feldata** Information om felet

Beroende på vilken typ av fel det rör sig om är det upp till den frågande parten att rätta till felet eller ignorera det.

### 7.4.2 Förfrågan om filändring

När en fil ändras måste alla delare notifieras om ändringen så att de kan uppdatera sin information. Följande information hör till denna typ av fråga:

**Svarsadress** Adressen till noden som gjorde ändringen.

**Fil-ID** Filen som ändringen skett i.

**Fil-Datum** Datumet för filen som ändrats.

**Frågedatum** Tidpunkt då ändringen gjordes.

Frågan kan tolkas som “Alla ni som har denna fil, uppdatera ert data till följande version.”. Denna fråga körs varje gång en fil ändras, det vill säga även katalogerna som borttagna eller tillagda filer låg i vid ändringstillfället i eftersom de måste uppdateras när filer tas bort eller läggs till.

### Svar

När en nod tar emot en förfrågan om filändring kontrollerar den om den ändrade filversionen matchar den cachade versionen. Vi har följande möjliga scenarier:

**A)** Om den ändrade filens version är nyare än den cachade versionen är det antagligen en normal ändring.

**B)** Om den ändrade filen är äldre eller samtidig som den cachade eller har vi en konflikt i ändringar.

**C)** Om den ändrade filen tillhör en filrymd som inte är en datafil eller inte tillåter kopior.

Scenario **B)** är möjligt eftersom det inte effektivt går att låsa filer i systemet. Om alla noder upplever scenario **A)** har vi ingen konflikt och noden kan fortsätta och hämta den nya versionen av filen med en hämtningsfråga.

### Hantering av ändringskonflikt

Eftersom noder meddelas om en uppdatering vid olika tidpunkter i delar av nätverket kommer det dock alltid finnas risk för att det finns noder som upplever scenario **A)** samtidigt som andra upplever scenario **B)**. När en nod upptäcker att en konflikt har skett skickas ett felmeddelande till noden som gjorde den äldsta ändringen (eller den med lägst adress ifall ändringarna sammanfaller på sekunden) om att en konflikt har upptäckts. Inträffar scenario **C)** tar man bara hänsyn till den nyaste ändringen.

Noden som utförde den tidigaste ändringen måste då skapa en fil och återställa information i den äldsta av ändringarna i den. Den äldre versionen av filen får då ett eget ID och versionen blir den samma som tidpunkten för ändringen. Datat för ändringen finns redan på nätverket så det är bara en fråga om att få noderna att ändra fil-ID för datan som tillhör den. Noden skickar alltså ut en fråga om filåterställning.

**Fil-ID** ID för filen som ska återställas

**Fildatum** Versionen av filen som ska återställas

**Nytt fil-ID** Ett nytt ID för den återställda filen.

**Frågedatum** Datumet återställningen påbörjades

På samma gång måste alltså noden som utförde den senaste ändringen också ändra katalogen som den nya filen ska ligga i. På detta sätt försvinner ingen data vid ändringskonflikter, även om fler än en konflikt sker på samma gång och data är därmed säkrat från förlust. I fallet då fler än två noder rapporterar en ändring i en fil samtidigt kommer flera noder behöva skapa kopior av sina ändringar, vilket kommer resultera i att de kommer ändra samma katalog ungefär samtidigt vilket kan resultera i ännu en konflikt. Alla utom den senaste av ändringarna kommer behöva skapa en kopia av katalogen på vilket sätt det fortsätter ner i katalogstrukturen. Då är är dock upp till användarna att sammanställa ändringarna manuellt även om detta kanske inte är önskvärt för alla filer. Exempelvis vill vi inte inte ha kopior av rotkatalogen eller vissa typer av systemfiler utan då används enbart den senaste versionen.

#### Filåterställning

När en nod får en förfrågan om att återställa en fil tar den helt enkelt den data som lagrats för den utgående filen och flyttar den till sin filtabell där den associeras med ett nytt ID och en ny version.

Detta ger ett visst skydd mot att användare med ont uppsåt raderar data, eller om data raderats mot ens vilja eftersom man kan återställa den förlorade informationen en tid efter den raderats.

**Fortsatt diskussion** Denna typ av konflikthantering kan bli problematisk i fallen då konflikter rör speciella filer som exempelvis systemdatafiler och rotkatalogen. Beroende på filrymd vill man kanske inte tillåta sådan återställning av filer.

En diskussion om att förbättra förfarandet som löser en del av dessa problem vid filändringar kan läsas i 8.2.2.

### 7.4.3 Förfrågan om fildelning

Om en nod vill tillgängliggöra (dela ut) en fil eller en mängd fragment i nätverket skickar den ut en fråga om att dela ut en fil. Användaren efterfrågar då en mängd noder med ledig cacheyta och lagrar datan hos dem. Noden skickar ut följande information med sin förfrågan.

**Svarsadress** Anslutningsinformation för att svara på frågan

**Fil-ID** Det unika ID:t för den nya filen

**Fil-datum** Datumet då den nya filen lades upp.

**Frågedatum** Datumet då frågan utfärdades

#### Svar

En nod som tar emot denna fråga börjar med att kontrollera om frågan ursprungligen kommer från en av sina grannar. I så fall sker en kontroll om huruvida det har inträffat en ID-kollision. Två filer med samma ID kan inte existera tillsammans i systemet. I detta fall skickar noden ett felmeddelande vilket får avsändaren att försöka igen med ett nytt ID.

Om en nod väljer att svara ansluter den till svarsadressen i frågan och meddelar hur många fragment den är villig att ta emot.

När den uppladdande parten fått ihop tillräckligt många noder som är beredda att lägga upp data fördelar den data över noderna och i tillräckligt många kopior för att säkerställa filens tillgänglighet.

**Grannkontroll** Genom att endast göra kontroller som den ovan nämnda om avsändaren är en granne, kan man bespara de andra noderna i nätverket att göra samma kontroll igen eftersom de med största sannolikhet skulle ha kommit fram

till samma sak. Grannen skickar således endast vidare frågan om det inte skett en kollision och alla andra i nätverket kan vara ganska säkra på att ingen kollision sker i och med att filen läggs till.

#### ID-kollision

När två filer inom en kort tidsrymd läggs till i nätverket och har samma ID inträffar en kollision. Två filer med samma ID kan inte tillåtas existera, och till den som senast lade till filen skickas ett felmeddelande och får försöka på nytt. Sannolikheten för detta är dock väldigt liten och beror i så fall troligen på brister i hashalgoritmen som används för att generera filens ID. Om det inträffat en ID-kollision så har den med stor sannolikhet inträffat innan filen lades till.

Den som får felmeddelandet skickar också ett felmeddelande till de noder som svarat på frågan att transaktionen måste avbrytas och göras om.

#### Metadata

Till varje fil hör en systemfil som lagrar information om den. Även denna fil och alla andra filer med relaterad information ska skapas eller ändras när en fil läggs till.

### 7.4.4 Förfrågan om anslutning

När en nod vill lägga till en annan som granne måste den andra noden godkänna detta och själv lägga till den noden. Noden som önskar ansluta till en annan skickar följande fråga:

**Svarsadress** Adressen till noden som försöker ansluta

**Fråge-datum** Tidpunkten då frågan utfärdades

När en nod får denna förfrågan avgör den om den borde lägga till den anslutande noden eller inte, och lägger i så fall till avsändaren bland sina grannar och meddelar denne om att begäran godtogs så att denne också kan lägga till noden bland sina grannar.

Om en nod upplever att den har för många grannar kan den avstå från att ta nya grannar och istället svara med ett felmeddelande. Om nod A sägs vara granne med nod B måste även nod B vara granne med nod A. Det betyder att en nod måste släppa en granne så fort den inte verkar vara ansluten längre.

Det är vid detta stadium ett autentiseringsprocedut kan tänkas läggas till. Detta diskuteras mer i detalj i sektion 8.1.1.

### 7.4.5 Förfrågan om borttagning av fil

Om en användare vill radera en fil i nätverket skickar han ut en fråga med följande information:

**Svarsadress** Adress till noden som begärt filen raderad

**Fil-ID** Identifieraren för filen som ska raderas

**Fil-datum** Versionen av filen som ska tas bort

Alla noder som får detta meddelande tar bort filen ur sin cache, men behåller den i sin lista av utgående filer och behåller den där ett tag. Om filen som ska tas bort är en metadatafil ska noden kontrollera att den ordinarie filen ska tas bort. Frågan vidarebefordras inte förrän en begäran om att radera den ordinarie filen mottagits. Om filen är en vanlig fil ska noden ställa en borttagningsfråga för dess metadatafil med samma frågedatum som den mottagna borttagningsfrågan.

#### Borttagningskonflikt

Det är tänkbart att två olika noder inom en kort tidsrymd beslutar att ta bort samma fil. Då kan en del av nätverket ha raderat filen innan frågan från den andra kommer fram och omvänt på samma sätt som filändringskonflikter. Detta är egentligen inget problem eftersom de båda borttagningarna får samma effekt. Eventuellt inträffar en ändringskollision när de båda noderna som försöker ta bort filen ändrar katalogen filen ligger i.

#### Borttagning av katalog

Om användaren raderar en katalog som pekar till andra filer kanske det inte längre finns några kataloger som refererar till dessa filer. Det går därför inte att ta bort kataloger som inte är tomma. Det kan dock tänkas att det genom användargränssnittet är möjligt att rekursivt utföra en borttagning på alla underliggande filer.

### 7.4.6 Meddelande om släppning av fildata

En nod kan tänkas vilja be en annan att släppa data om en fil, till exempel om någon fil är överrepresenterad på nätverket för att frigöra utrymme. Noden tar bort filen ur sin filtabell, men behåller den i sin lista över utgående filer.

# Kapitel 8

# Förbättringar

I detta kapitel presenteras förändringar som kan göras till den grundläggande funktionaliteten för att underlätta eller eliminera det som är problematiskt med den ursprungliga implementationen eller skapa funktionalitet som inte var möjlig tidigare.

Vi fortsätter här att anta att Query-flooding används för sökning på grund av dess simplicitet, men formulerar förbättringarna så att de går att använda även med en annan sökmetod.

## 8.1 Näverket

I den grundläggande implementationen finns det inga begränsningar för vilka som kan använda systemet, vilket inte alltid är önskvärt. Här presenteras däremot ett förslag till ett autentiseringssystem vilket möjliggör bland annat filrättigheter och inloggning.

### 8.1.1 Autentisering

För att begränsa användandet av nätverket så att inte oönskade personer kan använda det krävs någon slags autentiseringsprocedur. På grund av nätverkets distribuerade natur finns det en del problematik.

Autentiseringsinformationen måste lagras på nätverket, så att alla noder kan använda den och utföra autentiseringsproceduren. Detta betyder även att varje användares autentiseringsuppgifter är distribuerade på nätverket vilket ställer en del krav på denna data och själva proceduren.

Vid autentiseringsproceduren sker ett utbyte av information med vilken noden som utför anslutningsproceduren kan avgöra om noden är behörig eller inte. För högsta möjliga tillgänglighet bör varje redan ansluten nod kunna utföra autentiseringen. Varje nod ska autentisera varje ny granne den antar, och noder som misslyckats med sin autentisering eller inte utfört någon har således inga grannar vilket betyder att den inte har tillgång till några filer. Detta i sin tur betyder att den faktiskt inte *kan* utföra proceduren.

Man kan ta säkerhetsåtgärderna ett steg längre och innan varje dataöverföring kontrollera om mottagaren är autentiserad, men antagandet att ingen obehörig nod har grannar och således inte kan utfärda frågor torde vara tillräckligt.

### Krav på autentisering till systemet

Att ta fram en felfri autentiseringslösning för systemet är utanför detta arbetes omfång, och vi antar att varje av dessa krav är uppfyllda för resten av dess implementation och användning.

- Krav på autentiseringsdata
  - Varje behörig användare förknippas med en uppsättning systemfiler vilka innehåller informationen som krävs för att utföra en autentisering.
  - Data som lagras ska inte exponera någon information om användare för andra användare.
    - Det ska inte gå att utge sig för att vara någon annan med hjälp av den lagrade informationen.
    - Det ska inte gå att utvinna information om användare med den lagrade informationen.
  - Data som lagras ska inte vara vara användbart utanför tillfället för autentiseringen.
  - Autentiseringsdata ska vara tillgängligt för alla redan autentiserade användare vid varje tillfälle.
- Krav på autentiseringsproceduren
  - Autentiseringsproceduren ska inte exponera någon information om användare för andra användare.
    - Det går inte att ange sig för att vara någon annan med hjälp av den överförda datan.
    - Det ska inte gå att utvinna information om användare med den överförda informationen.
  - Data som överförs ska inte vara användbart utanför tillfället för autentiseringen.

- Krav på noderna
  - Varje redan ansluten nod ska kunna utföra autentiseringsproceduren.
  - Varje nod ska autentisera sig för att kunna vara verksam i nätverket.
    - Varje nod ska autentisera en nod som vill bilda grannskap.
    - Varje nod ska neka grannskap till obehöriga noder.
  - Varje nod ska vid varje givet tillfälle kunna autentisera sig.

### 8.1.2 Användarnamn

En autentisering kräver att användaren kan identifiera sig unikt med ett användarnamn, Detta är till systemets fördel då man kan kräva att skicka med sitt namn med vissa frågor och på så sätt undvika att en användare anger sig för att vara någon annan. Man kan anta att autentisering till nätverket sker genom att man anger sitt användarnamn och lösenord och att man på så sätt kan binda varje användare till ett unikt ID. Eftersom användarnamnet är hälften av den information som krävs för att logga in vill man undvika att användarnamn skickas runt i klartext på nätverket. Man kan då istället skicka en hash som även den unikt identifierar en användare på nätverket. Det är då lätt att associera autentiseringsinformation till användaren om dessa filer utgör egna filrymder där filhashdelen utgörs av hashen av användarnamnet.

För att se till att ingen försöker utge sig för att vara någon annan kan man anta att man utöver svarsadressen bifogar en användarhash till frågor man ställer. Då kan en nod innan den vidarebefordrar en fråga kontrollera om den kommer från en granne, i vilket fall den kontrollerar om användarhashen stämmer med den i hans egna register. Om man då endast vidarebefordrar frågan om dessa stämmer överens kan man säkert anta att ingen användare försöker utge sig för att vara någon annan[1].

[1] Förutsatt att varje nod faktiskt utför kontrollen.

### 8.1.3 Registrering

För att det ska skapas autentiseringsinformation om en användare måste den ha registerat sig. Därför måste en ny typ av förfrågan göras för att kunna registera nya användare till systemet. Vid registering anges ett användarnamn som blir permanent och kommer förknippas med användaren när den använder nätverket. Användarnamnet kommer endast att användas i sin hashade form på nätverket. Till varje användare associeras filer med autentiseringsinformation som krävs för att användaren ska autentisera sig igen.

### 8.1.4 Noder

#### Statistik

En nod kan tänkas lagra information om hur den upplever nätverket för att förbättra sin och därigenom andras situation. En nod kan samla statistik genom frågorna den ställer till sina grannar. Exempel på statistik som kan vara intressant presenteras i följande stycken. Läsaren bör ha i åtanke att en del av teknikerna endast ger utbetalning i väldigt stora nätverk där prestandavinsterna är större än värdet av det lagringsutrymme man offrar i utbyte.

#### Grannbyte

Om man till varje fråga lägger till en lista av alla noder som hanterat frågan allt eftersom den passerar genom noder kan man få reda på hur många hopp frågan tog innan den kom fram. När någon nod svarar får noden som ställde frågan reda på hur lång vägen var, avgöra om vägen är för lång och ta noden på mitten som granne. På så sätt flyttas noder om i nätverket så att genomsnittslängden på alla frågor minskar. Eftersom denna lista potentiellt blir väldigt lång kan man tänka sig att begränsa de frågor som lagrar sådan information till vissa frågor[2].

[2]Se 8.2.1

Antag att man för varje sådan fråga beräknar en genomsnittslängd på frågornas vägar, och när vägen en fråga färdas är längre än genomsnittet tar man noden som ligger på mitten av vägen till sin granne och kastar bort den granne som frågan ursprungligen gick igenom.

Eftersom man endast halverar vägar när de är längre än genomsnittet betyder det att man kommer närma sig en konstant som är det faktiska genomsnittet. Det är tänkbart att skära ner på sådana frågor när tillräckligt många sökningar är nära genomsnittet för att inte behöva skicka runt så mycket data.

### Cachestatus

Med jämna mellanrum bör det utföras cachekontroller av filfragmenten. Man kan här utnyttja att alla delare av en fil söks upp vid filhämtningsförfrågningar. Den efterfrågande noden har då samlat alla delare av en fil och kan snabbt meddela dem om filfragmentens cachestatus efter att den eventuellt omfördelat filfragment efter behov.

De flesta filer cachekontrolleras automatiskt i och med att användare återansluter till nätverket, och de mest populära kontrolleras ofta när de används i nätverket. Denna funktionalitet kan användas för att fasa ut filer i system där sådant beteende är önskvärt. När då en fil är tillräckligt utfasad begär noden som tycker att den är inaktuell att filen ska tas bort.

Denna förbättring uttrycker sig i ett ökat behov av datatrafik, men ger i utbyte möjlighet att tillgodose filernas tillgänglighet vilket är en investering som inte har så stor effekt på nätverkets prestanda om inte antalet delare per fil är väldigt stort.

### Populäritet

Allt eftersom filer eftersöks kan man ranka dem efter populäritet. En mer populär fil bör vara mer tillgänglig i nätverket, men detta har följden att mindre populära, men kanske viktigare filer blir lidande.

### Routing

Utöver att ha en förteckning med vilka filer som en nod har tillgängliga kan en nod lagra vilka användare som gett ett jakande svar på någon fråga associerad med en viss fil. På detta sätt kan noder hålla en routingtabeller, så att man kan styra filsökningar i den riktning som ett jakande svar kom ifrån. Det räcker att noden vet i vilken riktning (mot vilken granne) sökningen ska gå. Detta ökar å andra sidan lagringsbördan för varje nod och det krävs antagligen ett väldigt stort nätverk för att prestandavinsterna ska bli större än förlusterna av att offra lagringsutrymme för statistik.

Det finns dock inga garantier för att routingtabellen är aktuell eftersom en grannes avlägsna granne kan tänkas lämna nätverket och därför måste samtliga grannar ändå delges frågan. Således avhjälper inte detta särskilt mycket, om inte det statiskt går att avgöra vilka användare som direkt kan associeras med vilken information[3].

Det är tänkbart att begränsa denna routingtabell till att endast omfatta en del filer, till exempel baserat på populäritet, vilket dock kräver att noder delges filers populäritet genom de meddelanden noden får efter lyckade filsökningar som används för att bygga upp routingtabellen.

En annan implementation av routingtabellen kan vara att ranka sina grannar efter hur många lyckade frågor som skickats i den riktningen. Detta kan mot en mindre kostnad i lagringsyta bidra till att ge en bättre fördelning av filer genom att styra om fildelningsfrågor till lägre rankade grannar och bättre prestanda genom att styra filhämtningsfrågor till högre rankade grannar.

[3]Se DHT

## 8.2 Förfrågningar

### 8.2.1 Förfrågan om filhämtning

Eftersom denna fråga samlar mycket information om tillgängligheten av filer över nätverket kan den utvidgas så att den transportera en hel mängd information för att öka prestandan på nätverket.

Under en förfrågan om filhämtning har en nod kontakt med alla tillgängliga delare av en viss fil. Samtliga dessa noder kan delges denna information efter att en redistribution av filerna tagit rum. Se 8.1.4 för en utförlig diskussion.

Om varje nod när de får en filhämtningsfråga lägger sin kontaktinformation i en lista som skickas med frågan och varje svarande nod sedan skickar tillbaks listan till den efterfrågande noden kan man utifrån den informationen sträva för att balansera grafen. Dels genom att byta grannar så att sökvägarna blir kortare[4] och dels genom att informera noderna som vägen gick igenom om genom vilka grannar som de jakande svaren gick för att bygga routingtabeller i varje nod[5]

### 8.2.2 Förfrågan om filändring

De flesta ändringar i filer består i att man lägger till ny data i slutet av en fil eller ändrar en del av filen. Om man kan utröna exakt i vilka filfragment ändringen har skett kan man redistribuera de ändrade fragmenten och behålla resten eller endast flytta dem några steg åt sidan. Detta öppnar för möjligheter att slå ihop ändringar i filer. Detta kräver dock förändringar i protokollet så att en nod kan begära att en nod ändrar ordningsnummer eller version på sina filfragment utan att behöva redistribuera sitt data.

Detta löser problemet med kolliderande ändringar i kataloger och vissa systemfiler. Olika typer av procedurer för ändringar kan definieras för andra filrymder.

[4]Se 8.1.4
[5]Se 8.1.4

## 8.3 Filsystemet

### 8.3.1 Filrättigheter

Utöver autentisering kan man lägga på ytterligare ett säkerhetslager på systemet. Ibland kan man tänkas vilja begränsa vilka delar av filsystemet en användare har tillgång till och vad denne kan göra med informationen. Då är det lämpligt att implementera ett rättighetssystem. Med filrättigheter kan man hindra användare från att läsa eller ändra vissa filer och på så sätt effektivt skärma av användare från delar av filsystemet de inte ska ha tillgång till.

Det tänkta rättighetssystemet har ungefär samma funktionalitet som unix-liknande system där varje fil har en uppsättning rättigheter för en ägare, en för någon grupp samt en uppsättning för övriga användare. Varje användare kan därför tillhöra ett antal grupper som delar rättigheter för vissa filer.

Detta filrättighetssystem kräver att informationen om vilka rättigheter som gäller varje fil lagras på nätverket. Detta betyder att informationen om fillrättigheter, liksom den mesta andra, är distribuerad över nätverket varför det krävs ytterligare en uppsättning systemfiler med metadata.

#### Krav på rättigheter i filsystemet

För att filrättigheter ska vara meningsfulla krävs dels att det finns en fungerande autentiseringsprocedur enligt 8.1.1 samt att alla antaganden som gjorts där gäller.

- Krav på filrättighetsinformationen
  - Till varje fil associeras med en systemfil som innehåller rättigheterna som gäller för den, det vill säga vilken användare och vilken grupp den tillhör samt deras och övrigas rättigheter över filen.
    - Undantag gäller för alla systemfiler. Systemfiler har speciella uppsättningar av rättigheter.

  - För varje grupp finns en förteckning över vilka användare som tillhör den gruppen.
  - Rättighetsinformation ska vara tillgänglig för alla noder i nätverket.

- Krav på rättighetsfilernas utformning
  - En användare utan rättigheter att utföra en handling på en fil kan inte utföra den handlingen.
  - En användare kan inte utge sig för att ha tillgång till en fil utan att vara berättigad till det.
  - En användare kan inte ange sig för att tillhöra en grupp utan att göra det.
- Krav på noder
  - Varje nod ska vid frågor bifoga sin användarhash för identifiering och rättighetskontroll.
  - Varje nod ska vid fråga om hämtning och ändring av filer se till att användaren som utfärdade frågan har rättigheter att utföra operationen. I annat fall ska ett felmeddelande skickas.

### Systemfiler och metadata

Eftersom systemet är distribuerat är också systemfilerna spridda på nätverket. Dessa filer är speciella då alla ska kunna använda informationen men kanske endast tillåta administratörer att ändra i dem. Ett oadministrerat system kan tänkas, men det skulle i så fall vara väldigt utsatt för sabotage.

Den stora skillnaden mellan systemfiler och andra filer är att man inte kan ha rättighetsfiler för rättighetsfiler. Därför krävs en annan metod för att avgöra vem som kan ändra i systemfiler om inte alla kan det.

### Rättighetskontroll

När en användare skickar en fråga om en fil, kommer alla nodens grannar kontrollera om förfrågan kommer från en granne i vilket fall den kontrollerar om användaren har rättigheter för att utföra den handlingen för filen. Den hämtar den till filen associerade rättighetsinformationen och avgör om noden är ägare till filen och har rättighet att utföra handlingen. Om så inte är fallet fortsätter den med att kontrollera om användaren tillhör gruppen och om den har sagda rättigheter. Om användaren inte hör till gruppen heller fortsätter den med att kontrollera vilka rättigheter övriga har på filen och om användaren då inte har rätt att utför handlingen skickas ett felmeddelande och förfrågan avbryts.

Faller är dock annorlunda vad gäller systemfiler. Är förfrågan en hämtning görs generellt ingen kontroll, men om användaren vill ändra filen måste andra kontroller utföras. Om användaren hör till administratörsgruppen skickas frågan vidare, i annat fall måste man kontrollera filrymden. Beroende på filrymd utförs olika kontroller. Här följer några exempel på dessa.

**Metadata Rättigheter** Om användaren har rättigheter att ändra filen som metadatafilen beskriver godkänns förfrågan, det vill säga om användaren är ägare eller hör till samma grupp och har skrivrättigheter

**Grupptillhörighet** Om användaren tillhör gruppen som filen beskriver har den tillåtelse att ändra filen.

**Autentiseringsinformation** Om användarens hash överensstämmer med filhashen har den tillåtelse att ändra filen

### Förteckningsfiler

Det finns för systemet filer som innehåller listor av systemfiler. Dessa agerar som förteckningar av den filtyp den innehåller. Det kan inte finnas kopior av dessa, och åtkomsten till dem är begränsad. Dessa filer ägs av administratören och kan endast ändras av denne genom att olika typer av förfrågningar.

Dessa filer är i princip kataloger som innehåller pekare till vissa typer av systemfiler. Exempel på filrymder som kräver förteckningar är grupptillhörighetsfiler och autentiseringinformation. Typer av data som inte kräver förteckningar är filer som är förknippade med andra filer såsom metadata. Dessa filer går alltid att komma och radera genom andra vägar i nätverket, medan information om användare och grupper inte går att nå utan referens till grupperna. En följd av detta är att ingen grupp eller användare kan läggas till om ingen administratör finns tillgänglig.

Det är tänkbart att införa ytterligare begränsningar i systemet genom att låta metadata ha förteckningar. Detta får följden att filer endast kan läggas till när en administratör finns tillgänglig. Vilket i sig kan vara önskvärt i vissa sammanhang.

## 8.4 Kompromisser

### 8.4.1 Anslutning

Det finns ingen garanti för att de noder som var tillgängliga när en nod lämnade nätverket sist är tillgängliga igen när den återansluter. För att motverka denna typ av uteslutning kan man definiera en eller flera punkter i nätverket som i normala fall kan antas finnas tillgänglig, och på så sätt kan fungera som reserv ifall ingen av de tidigare noderna skulle vara tillgängliga. Denna funktionalitet är analog med den av en inloggningsserver, men är dock inget beroende för nätverket. Detta minskar sannolikheten för ett fall då det är omöjligt att automatiskt ansluta till nätverket igen.

### 8.4.2 Anonymitet

Eftersom det med varje förfrågan skickas med både IP-adress och användarnamn kan varje användarhash förknippas med en IP-adress vilket innebär att anonymiteten är hotad. Man kan implementera sökningar och frågor så att de när ett svar noteras skickar tillbaks svaret längs vägen den kom så att ingen får reda på var frågan hade sitt ursprung. Detta orsakar dock en del extra trafik på nätverket, vilket

inte alltid är önskvärt.

# Kapitel 9

# Diskussion

## 9.1 Brister

### 9.1.1 Förfrågnignar

Metoden som hittils antagits för att varje nod ska ta del av förfrågningar genererar en potentiellt väldigt stor mängd trafik vilket kan orsaka tröghet i nätverket eller till och med att systemet kollapsar under sin egen trafik. Metoden är därmed sagt naiv i sin utformning och man kan tänka sig att implementera en annan sökningsmetod.

#### DHT

Istället för att skicka förfrågan vidare till alla sina grannar kan man tänka sig att endast skicka den till en delmängd av dessa. Det är då viktigt att den skickas till rätt grannar. Eftersom alla noder måste vara likvärda kan man inte använda en hierarkisk struktur likt ett sökträd, utan måste använda sig av en metod som från en godtycklig plats i en graf kan närma sig en plats med den sökta datan. På så sätt är endast en liten del av nätverket inblandade i varje förfrågan. Här kan någon av DHT-teknikerna CAN, Chord, Pastry eller Tapestry implementeras[1].

[1]Mer läsning om dessa finns här `http://www.cs.berkeley.edu/~istoica/papers/2003/cacm03.pdf`

Vilken av dem nu än implementeras så är de förbättringar över de föreslagna på flera områden, men påverkar endast hur noderna som ska få eller dela informationen nås. Skillnaden är att när man söker efter en fil får man ut en lista över alla delare av en fil istället för filfragmenten - det vill säga en DHT. Det är lätt för en delare av en fil att registrera sig som delare hos noden som har tabellen då det bara är att göra en sökning och ange sin adress tillsammans med filen man delar.

En nackdel med detta är att man ständigt måste kontrollera om informationen man håller i är aktuell, samt ständigt uppdatera sitt grannskap så att det stämmer.

### Krav på DHT-metoden

Det är dock inte intressant för implementation för DHT-protokoller är implementerat bortsett från vissa detaljer:

- Sökningen ska utgå från användar-ID.

- En användare ska slumpmässigt kunna placera ut filer genom att slumpmässigt generera användar ID:s och på så sätt få ut den användare som är närmast det genererade ID:t.

Vilket gör det lämpligast för en CAN-liknande implementation med en n-dimensionell rymd i bas 16 där n är längden på en filhash. Då kan varje fil unikt pekas ut som en punkt i rummet med sin hash och lätt förknippas med en användare. En användare har en lista av varje delare av filerna den förknippas med vilket gör att man kan utföra de ovan definierade operationerna utan ändringar.

Detta bättrar på implementationen så att man kan slå upp en fil i nätverket och få kontakt med alla noder som delar den, vilket är grunden för all funktionalitet i nätverket. Med denna information kan man enligt implementationen ovan ändra, radera och ta bort filer med små ändringar till procedurerna.

### 9.1.2 Tomt nätverk

Ett annat tänkbart problem är att det vid något tillfälle inte finns några anslutna noder. Detta får följden att ingen kan återansluta till nätverket, om inte noderna på något sätt kan finna varandra när de behöver det. Detta är ett argument för att ha en inloggningserver för att koordinera anslutningar och för att kunna ha en punkt som ger viss stabilitet.

Om alla noder vid något tillfälle skulle råka bli offline krävs det att tillräckligt många medlemmar av nätverket på något sätt kan finna varandra och ha tillräckligt med information för att driva nätverket. Det finns alltså en kritisk punkt av antal användare som minst måste finnas tillgängliga för att driva nätverket. Om antalet understiger denna gräns finns det risk för att systemet slutar fungera om inte alla nödvändiga filer finns tillgängliga.

Ett sätt att förhindra att viktiga filer försvinner är att dela ut dem mer, och eventuellt att införa ett rankingsystem för hur hög sannolikhet det är för att en viss nod är tillgänglig vid varje givet tillfälle och placera viktig data hos dem.

### 9.1.3 Disjunkta delgrafer

Eftersom varje nod endast känner till ett fåtal andra riskerar det att bildas disjunkta celler av nätverket så att det inte finns någon väg i grafen som passerar varje nod. Detta är katastrofalt för systemet eftersom frågor inte kan nå fram till sina destinationer, och resultatet blir att inte alla filer blir tillgängliga för alla noder.

Detta är svårt för någon nod att lösa individuellt, och dessutom omöjligt att upptäcka. Det krävs att det finns någon procedur som kan övervaka hur många noder det finns i nätverket och att de alla är förbundna på något sätt. Detta ger dock systemet ett beroende av en viss eller vissa punkter eftersom det inte går att garantera att hela grafen är sammanhängande om inte dessa övervakare alltid är online. Det är också tänkbart att det finns en tjänst som anslutna noder kan få slumpmässiga grannar tilldelade till sig, och att alla noder i nätverket kan använda denna tjänst. Då motverkas cellbildandet något. Dessutom motverkas det ytterligare

av att noderna har fler grannar.