BACHELOROPPGAVE:

Lagring, Backup, Recovery og Kræsjhåndtering i OpenStack

bsrmgr

FORFATTERE:
KIM HOLMEBAKKEN
THOMAS RONGVED KRISTIANSEN
KENT-MARIUS WERNER

Dato:
19.05.2014

# SAMMENDRAG

| | | |
|---|---|---|
| Tittel: | bsrmgr | Dato : 19.05.2014 |
| | Lagring, Backup, Recovery og Kræsjhåndtering i OpenStack | |
| Deltakere: | Kim Holmebakken<br>Thomas Rongved Kristiansen<br>Kent-Marius Werner | |
| Veiledere: | Øivind Kolloen | |
| Evt. oppdragsgiver: | Erik Hjelmås | |
| Stikkord/nøkkelord (3-5 stk) | Backup, storage, recovery, SAN, kræsjhåndtering | |
| Antall sider(ord): 236 (26 124) | Antall vedlegg: 10 | Publiseringsavtale inngått: ja |


Kort beskrivelse av bacheloroppgaven:

I dagens tid blir det stadig mer behov for at man klarer å opprettholde integriteten og konsistensen i alt av informasjon man behandler, noe som kan være vanskelig siden det blir benyttet virtuelle maskiner og skyløsninger.

Behandling av informasjonen som blir produsert i de virtuelle maskinene vil være viktig å ta hånd om, og sikre at de er tilgjengelige på alle tider brukeren kan ønske å bruke den. I denne oppgaven har gruppen satt opp to forskjellige systemer for å sammenligne ytelse med tanke på lesing og skriving av data. Det vil bli gjort analyse av testdata fra tester gjennomført med ytelsesmålingsverktøy, som blir kjørt både mot filsystemer, men også direkte mot block devices.

Målet med denne bacheloroppgaven er å sette de to systemene opp mot hverandre, samt å se hvordan de kan håndtere strømbrudd og defekte hardware-komponenter. Det fremkommer tydelig av testresultatene at ytelsen er avhengig av typen programvare som brukes for å håndtere caching og lagring mot backend, hvor hver av systemene har sine fordeler og ulemper. Leseren vil gjennom denne rapporten få kunnskaper om teorien og praksisen rundt XFS, bcache og ZFS on Linux, samt noen best-practices rundt disse temaene.

# ABSTRACT

| | | |
|---|---|---|
| Title: | bsrmgr | Date : 19.05.2014 |
| | Storage, Backup, Recovery and Crash Handling in OpenStack | |
| Participants: | Kim Holmebakken<br>Thomas Rongved Kristiansen<br>Kent-Marius Werner | |
| Supervisor | Øivind Kolloen | |
| Employer: | Erik Hjelmås | |
| Keywords (3-5) | Backup, storage, recovery, SAN, crashhandling | |
| Number of pages(words): 236 (26 124) | Number of appendices: 10 | Availability: open |


Short description of the bachelor thesis:

It has become increasingly demanding that we stride to maintain the integrity and consistency of all information produced, which can be difficult with the use of virtual machines and cloud solutions.

Information management are therefore important to do properly, especially since the use of virtual machines increase every day, including the demand for high availability. In this thesis, the group has set up two different systems ande compared performance for reading and writing data. There will be an analysis of test data from the tests conducted with performance measurement tools, which are ran on file systems, but also directly on block devices .

The aim of this bachelor thesis is to put the two systems up against each other, and to see how they deal with power outages and faulty hardware components. It is obvious from the test results that performance are dependent on the type of software used to manage caching and storing, where each system has its advantages and disadvantages. The readers will acquire knowledge on the theory and practice of XFS, bcache and ZFS on Linux as well as some best-practices around these issues.

# Forord

Av alle bacheloroppgavene som var foreslått av oppdragsgiverne høsten 2013, var det oppgaven som omhandlet kræsjhåndtering som virket mest interessant. Grunnen til dette er at gruppen synes temaet datahåndtering er relevant for den tidsperioden vi er inne i. Gode backuprutiner og opprettholdelse av integritet ved en eventuell kræsj, er noe som stadig blir mer og mer viktig.

Det har for hele gruppen vært en bratt læringskurve, hvor hvert av medlemmene har måttet lære seg en del ny teori og praksis. Gruppa synes all den nye kunnskapen er verdifull, ettersom det kan gi en fordel når man skal ut i arbeidslivet der kunnskap om datalagring er spesielt ettertraktet.

I løpet av denne perioden har gruppen vært i kontakt med både oppdragsgiver, veileder og andre fagpersoner for hjelp med bacheloroppgaven. Det rettes derfor en stor takk til Erik Hjelmås (oppdragsgiver), Øivind Kolloen(veileder) og Jon Langseth for bistand i bacheloroppgaven.

Videre er det ønskelig å sende en takk til de personene som har hjulpet til med å forbedre oppgaven i form av rettingskriving og ordning av det logiske oppsettet i rapporten. Hovedsaklig er det her snakk om Elisabeth Rongved(rettskriving) og Sven Erik Espeland(Evry).

Gruppen ønsker også å takke de to andre gruppene som har skrevet bacheloroppgave innenfor samme studieretning som gruppen vår, da disse to gruppene har hjulpet til med å få et nytt syn på en rekke av temaene gruppen har vært gjennom i løpet av denne perioden.

Gjøvik 19.05.2014

| | | |
|---|---|---|
| Holmebakken, Kim | Kristiansen, Thomas R. | Werner, Kent-Marius |
| Kim Holmebakken | Thomas R. Kristiansen | Kent-Marius Werner |

# Innhold

# Figurer

# Kapittel 1

# Innledning

## 1.1 Organisering av Rapporten

- I kapittel 1 vil innledningen til rapporten være, samt noen overordnede forklaringer til generelle begreper som er brukt i utførelsen av bacheloroppgaven.
- I kapittel 2 vil rapporten inneholde kravspesifikasjonen og prinsippene rundt de valgene som er tatt.
- I kapittel 3 går rapporten inn på utstyr som er i bruk, samt hvordan designen på arkitekturen rundt oppsettene er.
- I kapittel 4 vil ta rapporten for seg hvordan de ulike oppsettene som ble designet i kapittel 3 er implementert. Her vil det bli tatt for seg utførelsesfasen av de gitte oppsettene.
- I kapittel 5 vil rapporten basere seg på testresultater. Det vil her også bli diskusjon av resultatene
- I kapittel 6 vil konklusjonen på vår bacheloroppgave være, der det oppsummeres meninger og resultatene gruppen fant.
- I kapittel 7 vil det komme noe ekstra informasjon, basert på tester gjort med noen endrede parametere og gjennomføringsmetoder.
- I kapittel 8 er vår litteraturliste, der kildene som er brukt i oppgaven listes.
- I kapittel 9 vil inneholde vedlegg til oppgaven, det omfatter forprosjektrapporten, møtene og dokumentasjonen gjort underveis.

## 1.2 Bakgrunn

*"Siden tidenes morgen har menneskeheten lurt på om man kan samle flere virtuelle maskiner under ett system, og i moderne dager har mennesket kommet frem til en løsning som kalles for OpenStack. Her vil man kunne ha flere virtuelle maskiner samlet under en logisk inndeling, og man vil kunne håndtere denne logiske inndelingen som en enhet eller se på de individuelle virtuelle maskinene."*

Kim Holmebakken, 2014

Openstack er et cloud-computing system som er lagt opp til å håndtere offentlige og private sky-baserte systemer. Selve OpenStack er open source og er lisensiert under Apache.

Systemet, eller prosjektet, er drevet av OpenStack Foundation som er en non-profit organisasjon og mer enn 200 bedrifter verden over er med på å utvikle miljøet. Herunder faller bedrifter som Hewlett-Packard og IBM, samt AT&T som alle er store internasjonale bedrifter. [1]

Teknologien rundt OpenStack består av flere små moduler som jobber sammen slik at systemet som en helhet har kontroll over prosessorbruk, lagring og nettverksressurser. Dette kan monitoreres ved hjelp av et datasenter som er bygget opp av et web-basert dashbord, kommandolinjer eller en egen API.

OpenStack har ingen "smart" løsning for håndtering av backup, lagring og kræsjhåndtering siden standard praksis er å implementere en Uninterruptable Power Supply, forkortet til UPS. Det er fordi at disse UPS'ene vil gi en batteritid som er nok til at data kan skrives til disk og deretter slås helt av.

Oppgaven vår bygger på ovenstående opplysninger.

## 1.3 Oppgavedefinisjon

Arbeidstittel:
Lagring, backup, recovery og kræsjhåndtering i OpenStack.

Rapporten vil ta for seg hovedområdene som oppstår når man skal håndtere backup, lagring og kræsjhåndtering i et OpenStack miljø. Den rettes mot at funnene skal kunne benyttes i Høgskolen i Gjøviks cloud-nettverk, SkyHiG. I cloudmiljøet er det liten til ingen håndtering for feil som kan forekomme. Dette er et grunnsystem som er lagt til for å kunne gjøre virtualisering av ulike servertyper, hvor disse kan brukes i de ulike fagene som tilbys ved HiG.

Systemet skal kunne bli tilgjengelig for studenter slik at de kan praktisere de ulike teoriene man lærer i fagene skolen tilbyr. Det vil bli sett på om dette kan løses uten en UPS, eventuelt om det er nødvendig med en UPS for å oppnå den integriteten som kreves av systemet. Dette fører til at det må undersøkes hvilke UPS som er tilgjengelige og mest aktuelle for OpenStack miljøet. Her vil det bli tatt hensyn til pris og hvor mye kapasitet som må til for å kunne bringe systemet trygt ned.

For at dataintegritet skal ivaretas, så må det implementeres en løsning som kan gi støtte for korrekt recovery, selv om systemet skal gå ned. Det er her vår oppgave kommer inn, og det som kommer til å implementeres er et system som kan bringe OpenStack miljøet opp igjen, uten at noe eller svært lite data går tapt. Ordet data blir beskrevet videre under delkapittel terminologibruk.

Det ble presentert ulike oppsett som kan være ønskelig å implementere. Etter å ha betraktet de ulike alternativene har gruppen kommet frem til to ulike oppsett som kan gi oppdragsgiver det ønskede resultatet.

Det ene oppsettet er en XFS/btrfs løsning, denne er bygget opp av:

- GlusterFS
- XFS (eventuelt btrfs), frontend
- SSD bcache, frontend
- Logical Volume Manager (LVM), backend
- HWRAID10 (1+0), backend

Det andre oppsettet er en ZFS on Linux løsning som består av:

- GlusterFS
- ZFS on Linux, frontend
- Logical Volume Manager (LVM), backend
- HWRAID10 (1+0), backend

Det bør merkes at ZFS on Linux også vil implementere en SSD cache, og ha en storage pool som kan simulere et RAID. Dersom man ønsker å implementere RAID i ZFS pool, må dette gjøres ved hjelp av software RAID via ZFS on Linux egne RAID-funksjon. Selve storage poolen til ZFS on Linux vil fungere på tilnærmet samme måte som en LVM. Disse to aspektene ved ZFS on Linux vil ikke bli brukt i vårt oppsett.

For begge oppsettene skal det sikres redundans mellom serverene i miljøet, dvs. at det skal kjøres et clustersystem for å ta seg av dette. Det vil i begge oppsettene brukes GlusterFS, som skal sikre at redundansen opprettholdes.

Hovedtyngden av prosjektet vil ligge på testing av begge oppsettene, både ved spesifikke scenariotester, som vil beskrives nærmere nedenfor, men også ved bruk av et program som heter fio.

Det vil være testresultatene fra fio-testingen som legges til grunn for valg av oppsett, dersom scenariotestene gir sammenlignbare resultater i begge oppsettene. Valget vil derfor være basert på ytelse fremfor kræsjhåndtering, ettersom begge oppsettene er forventet å håndtere dette på en tilfredsstillende måte gjennom bruk av GlusterFS mellom klientmaskin og frontendserverne, og RAID10 på backendserverne.

Deretter vil det testes to ulike scenarioer som kan forekomme der det er nødvendig å ha en backup foreliggende, samt hvordan disse ulike oppsettene eventuelt kan bringe opp systemet som gikk ned, og forhåpenligvis i samme tilstand som før det gikk ned.

De to scenarioene vil bestå av disse handlingene:

**Scenario 1:**
Strømmen vil bli fjernet fysisk fra oppsettet, for deretter å bringe det opp igjen, og eventuelt kjøre recovery på systemet for å se hvilken tilstand man får gjenopprettet systemet i. Her er hensikten å kartlegge hvorvidt oppsettet tilfredsstiller de kravene som stilles, samt å se hvilken tilstand OpenStack miljøet kommer tilbake i.

**Scenario 2:**
Her vil det fjernes enkelte servere for å se hvordan oppsettet vil klare å holde seg stabilt dersom forbindelesen til en av serverene som driver miljøet blir brutt, samt å se hvorvidt GlusterFS klarer å holde på den overordnede organiseringen av miljøet.

## 1.4 Formål

Formålet med oppgaven er først og fremst å kunne skape et oppsett som er med på å sikre data-integritet i form av konfigurasjoner på virtuelle maskiner, samt sikre rask tilgjengelighet dersom OpenStack miljøet skulle gå ned. Det skal være mulig å kunne hente opp den tidligere tilstanden miljøet var i da det gikk ned. Dette vil sikre gode backuprutiner og et håndteringsmiljø som vil fjerne behovet for en ekstern UPS. Det at en bedrift slipper å gå til innkjøp av en UPS kan være meget pengebesparende. Like viktig som å kunne håndtere en kræsj-situasjon, vil det være nødvendig at serverne kan håndtere dataflyt fra virtuelle maskiner.

## 1.5 Målgruppe

Målgruppe for prosjektrapporten vil i første rekke være oppdragsgiver, sensor, veileder og emneansvarlig, samt lærere som har samme fagfelt som oppgaven berører.

Målgruppe for systemet, etter det er satt igang, vil være studenter og ansatte ved HiG, som gjennom SkyHiGh vil bruke vårt lagringssystem.

## 1.6 Studentenes faglige bakgrunn

Gruppemedlemmene er alle fra studieretningen drift av nettverk og datasystemer ved Høgskolen i Gjøvik. Ingen av medlemmene har relevant arbeidserfaring innen fagfeltet, og må derfor tilegne seg kunnskaper om montering og oppsett av "best practice" innen fagfeltet.

Gjennom studiet har medlemmene tilegnet seg kunnskap om noen av de ulike teknologiene som blir tatt ibruk. Her legges det vekt på at medlemmene har hatt kurset database- og applikasjonsdrift samt operativsystemer og systemadministrasjon. Tidligere har gruppen hatt innføring i bruk av linux-servere, GlusterFS, og hvordan noen av de ulike systemene er bygget opp.

## 1.7 Termonologibruk

I løpet av denne rappporten vil det bli brukt noen ord for å lettere kunne forklare hva som er gjennomført av arbeid. Det vil også bli vekslet noe mellom ulike versjoner av samme ord. Det er derfor satt opp en oversikt over ord som kan forekomme i ulike varianter.

Med data menes det hvordan de ulike serverkonfigurasjonene er satt opp. Det skal også ivareta antallet virtualiseringer som er satt igang, dvs. at dersom det er 50 servere som er virtualisert skal 50 servere med riktig konfigurasjon komme opp igjen. Det bør merkes at data også er referert til som ren informasjon som blir produsert av serverne.

Når det gjelder begrepene backup, storage og recovery, finnes det ingen spesielt gode oversettelser til norsk. Bruken av ordene backup og recovery vil bli fulgt gjennom hele rapporten, mens storage noen steder også vil være forklart som lagring. Sammen med disse vil også ordet repository forekomme flere ganger. Med repository menes en felles lagringslokasjon, hvor flere brukere kan hente informasjon fra.

Flere ord som kommer til å gå igjen er ZFS, XFS og btrfs. Dette er navn på filsystemer som er brukt i løpet av prosjektet, og vil derfor være en del av mange beskrivelser. Betydningen av ordene er irrelevant for forståelsen av informasjonen i denne rapporten.

LVM står for Logical Volume Manager, og er nevnt noen ganger i rapporten. LVM brukes for å håndrere harddisker og andre lagringsenheter, som for eksempel HDD og SSD. HDD og SSD er benevnelser på to lagringsenheter som man kan finne eksempelvis i en laptop eller stasjonær maskin. [2]

Med virtuelle maskiner menes servere som arbeider på delte ressurser på en klient server, men som ikke eksisterer fysisk. Vår oppgave går på å ta hånd om data'ene som blir brukt av disse.

UPS, kort for Uninterrruptable Power Supply, er en enhet som brukes for sikker avslutning av serverne dersom det forekommer et strømbrudd. UPS'en har ett eller flere batterier som har nødvendig med strøm slik at et aggregat kan startes, eventuelt nok strøm for å kunne slå serverene trygt av.

Redundant Array of Independent/Inexpensive Disks, RAID, er en lagringsteknologi som brukes for å koble sammen disker i et array(rekke) for å få bedre redundans og ytesle. RAID finnes i flere varianter, men her er kun 0, 1 og 10 nevnt. For forklaring av disse typene RAID se kapittel 2.3. [3]

Caching og dets betydning vil ikke avvike noe fra hva som forståes ved vanlig bruk av dette ordet, nemlig en mekanisme som mellomlagrer data som kan bidra med å "lette på trykket" når det forekommer en informasjonsoverførelse fra server til harddisk. Cache har i seg selv en høyere hastighet når det gjelder bits per sekund, noe som gjør at serveren kan sende dens data til cache, og bruker mindre tid på å gjøre denne prosessen. Dette fører igjen til at serveren kan benytte dens ressurser til å utføre andre operasjoner.

Cache har ansvaret for å føre dette over til harddisk som nevnt, men kan være sårbar siden dette er det som kalles volatile storage, som tilsier at dersom strømmen går forsvinner informasjonen som er lagret på cache. Det er her igjen man kan se viktigheten i bruken av oppsettet. [4]

Bcache er en mekanisme som dreier seg om det samme som en vanlig cache, men differerer noe. Dette vil foregå på en eller flere Solid State Drive (SSD) som er satt i RAID. Dette fører igjen med seg at mellomlagringen er non volatile, noe som tilsier at dersom strømmen går vil man ikke miste data som er lagret her. I tillegg vil bruken av SSD øke ytelsen, større cache-område i form av mer bits per lagringsenhet, mot hva vanlige RAM-brikker vil kunne gi.

Det vil i denne rapporten bli vekslet noe mellom ordene folder, directory, mappe og katalog. Disse ordene betyr det samme, men brukes i ulike sammenhenger i forhold til hva som passer best i det gitte tilfellet.

For full stikkordliste vises det til appendix A - Stikkordliste.

## 1.8 Roller

Ved prosjektet er førsteamanuensis Erik Hjelmås, ved Høgskolen i Gjøvik, oppdragsgiver. Veileder er høgskolelektor Øivind Kolloen ved Høgskolen i Gjøvik. Hans faglige kunnskap vil benyttes for å oppnå den progresjonen som er ønsket.

### *Gruppen består av av 3 medlemmer:*

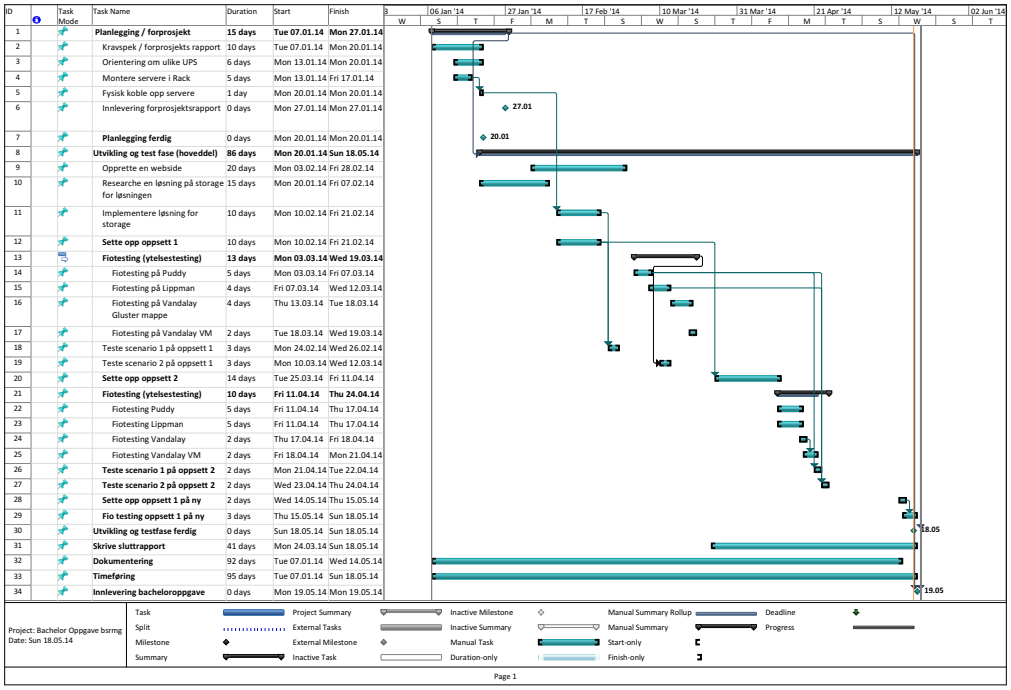
Kim Holmebakken: Gruppeleder, hovedkontaktpersonen.
Thomas R. Kristiansen: Gruppemedlem, dokumentansvarlig.
Kent-Marius Werner: Gruppemedlem, webansvarlig.

Rollene er kun tilstede for å ha en overordnet person som kan passe på at de ulike oppgavene blir gjort på en akseptabel måte, sett fra gruppens perspektiv. Selv om man har en hovedkontaktperson, kan de andre allikevel kontaktes hvis dette er ønskelig. Samme ordning gjelder dokumenter, selv om en er ansvarlig, vil også de andre gruppemedlemmene jobbe med denne delen av bachelor-oppgaven.

# Kapittel 2

# Prinsipper - Teori

## 2.1 Gantt-diagram

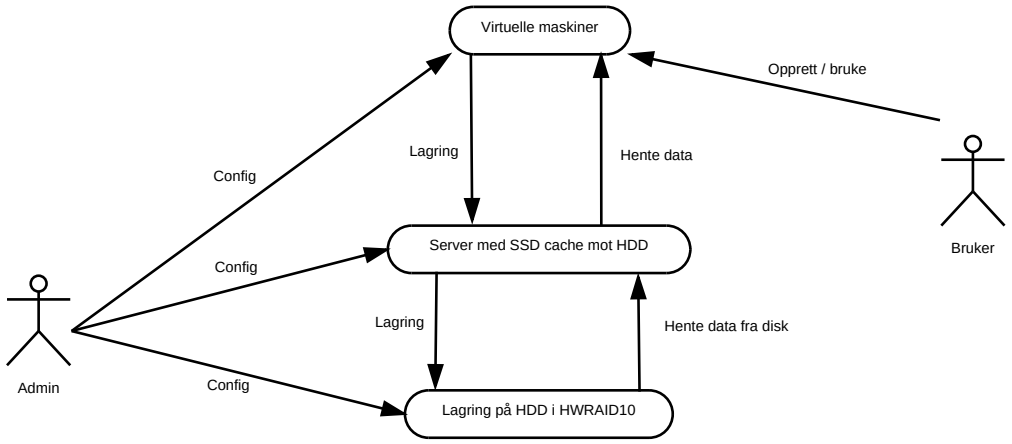


| ID | | Task Mode | Task Name | Duration | Start | Finish |
|---|---|---|---|---|---|---|
| 1 | | | **Planlegging / forprosjekt** | **15 days** | **Tue 07.01.14** | **Mon 27.01.14** |
| 2 | | | Kravspek / forprosjekts rapport | 10 days | Tue 07.01.14 | Mon 20.01.14 |
| 3 | | | Orientering om ulike UPS | 6 days | Mon 13.01.14 | Mon 20.01.14 |
| 4 | | | Montere servere i Rack | 5 days | Mon 13.01.14 | Fri 17.01.14 |
| 5 | | | Fysisk koble opp servere | 1 day | Mon 20.01.14 | Mon 20.01.14 |
| 6 | | | Innlevering forprosjektsrapport | 0 days | Mon 27.01.14 | Mon 27.01.14 |
| 7 | | | **Planlegging ferdig** | 0 days | Mon 20.01.14 | Mon 20.01.14 |
| 8 | | | **Utvikling og test fase (hoveddel)** | **86 days** | **Mon 20.01.14** | **Sun 18.05.14** |
| 9 | | | Opprette en webside | 20 days | Mon 03.02.14 | Fri 28.02.14 |
| 10 | | | Researche en løsning på storage for løsningen | 15 days | Mon 20.01.14 | Fri 07.02.14 |
| 11 | | | Implementere løsning for storage | 10 days | Mon 10.02.14 | Fri 21.02.14 |
| 12 | | | **Sette opp oppsett 1** | 10 days | Mon 10.02.14 | Fri 21.02.14 |
| 13 | | | **Fiotesting (ytelsestesting)** | **13 days** | **Mon 03.03.14** | **Wed 19.03.14** |
| 14 | | | Fiotesting på Puddy | 5 days | Mon 03.03.14 | Fri 07.03.14 |
| 15 | | | Fiotesting på Lippman | 4 days | Fri 07.03.14 | Wed 12.03.14 |
| 16 | | | Fiotesting på Vandalay Gluster mappe | 4 days | Thu 13.03.14 | Tue 18.03.14 |
| 17 | | | Fiotesting på Vandalay VM | 2 days | Tue 18.03.14 | Wed 19.03.14 |
| 18 | | | Teste scenario 1 på oppsett 1 | 3 days | Mon 24.02.14 | Wed 26.02.14 |
| 19 | | | Teste scenario 2 på oppsett 1 | 3 days | Mon 10.03.14 | Wed 12.03.14 |
| 20 | | | **Sette opp oppsett 2** | 14 days | Tue 25.03.14 | Fri 11.04.14 |
| 21 | | | **Fiotesting (ytelsestesting)** | **10 days** | **Fri 11.04.14** | **Thu 24.04.14** |
| 22 | | | Fiotesting Puddy | 5 days | Fri 11.04.14 | Thu 17.04.14 |
| 23 | | | Fiotesting Lippman | 5 days | Fri 11.04.14 | Thu 17.04.14 |
| 24 | | | Fiotesting Vandalay | 2 days | Thu 17.04.14 | Fri 18.04.14 |
| 25 | | | Fiotesting Vandalay VM | 2 days | Fri 18.04.14 | Mon 21.04.14 |
| 26 | | | **Teste scenario 1 på oppsett 2** | 2 days | Mon 21.04.14 | Tue 22.04.14 |
| 27 | | | **Teste scenario 2 på oppsett 2** | 2 days | Wed 23.04.14 | Thu 24.04.14 |
| 28 | | | **Sette opp oppsett 1 på ny** | 2 days | Wed 14.05.14 | Thu 15.05.14 |
| 29 | | | **Fio testing oppsett 1 på ny** | 3 days | Thu 15.05.14 | Sun 18.05.14 |
| 30 | | | **Utvikling og testfase ferdig** | 0 days | Sun 18.05.14 | Sun 18.05.14 |
| 31 | | | **Skrive sluttrapport** | 41 days | Mon 24.03.14 | Sun 18.05.14 |
| 32 | | | **Dokumentering** | 92 days | Tue 07.01.14 | Wed 14.05.14 |
| 33 | | | **Timeføring** | 95 days | Tue 07.01.14 | Sun 18.05.14 |
| 34 | | | **Innlevering bacheloroppgave** | 0 days | Mon 19.05.14 | Mon 19.05.14 |



Figur 2.1: Gantt-diagram slutt

## 2.2 UseCase




Figur 2.2: Use case

### 2.2.1 Høynivå use case

| | |
|---|---|
| Use case: | Virtuelle maskiner |
| Aktør: | Bruker, Admin |
| Mening: | Virtuelle maskiner kjøres i SkyHiGh. |
| Beskrivelse: | Administratoren skal konfiguere og tildele tilgang til de virtuelle maskinene. Brukerne skal benytte seg av disse serverene til forskjellige fag og oppgaver. De virtuelle maskinene skal lagre data opp imot serverne som har SSD i seg. De skal også hente data derifra. |

| | |
|---|---|
| Use case: | Server med SSD cache mot HDD |
| Aktør: | Admin |
| Mening: | Skal være frontend mot 3,5” HDD storage. |
| Beskrivelse: | Skal ta imot data som kommer fra de virtuelle maskinene, deretter skal det lagres på cache som er satt i et bcache-miljø(oppsett 1)/ZFS on Linux-miljø(oppsett 2) med SSD. Cachen skal så tømmes til 3,5” HDD som er en egen server for lagring på HDD i HWRAID10(1+0). |

| Use case: | Lagring på HDD i HWRAID10 |
|---|---|
| Aktør: | Admin |
| Mening: | Ligger som en treg backend i systemet, lagrer data fra cache. |
| Beskrivelse: | HDD i det fysiske oppsettet vil i all hovedsak ligge som en treg backend. HDD er satt opp i HWRAID10(RAID1 + 0). Dette vil si at man bruker både speiling og striping, som igjen medfører at systemet kan utnytte $^1/_2$ av den totale lagringskapasiteten. Striping vil føre til at systemet får økt hastighet. RAID'et skal ta imot informasjon fra SSD cache når lasten er lav nok, og levere data til SSD cache ved etterspørsel. |

### 2.2.2 Kravspesifikasjon

Selve kravspesifikasjonen kan man se ut fra oppgavebeskrivelsen, men i korte trekk ønsker oppdragsgiver at følgende skal utføres:

**Overordnet**

- Sette opp et system som kan håndtrere lagring, backup, recovery og med dette kræsjhåndtering for Openstack miljøet.
- Oppsettet skal kunne være skalerbart horisontalt, dvs. at oppsettet skal kunne ha flere servere tilkoblet hvor disse gir mer backend storage.
- Sammenligne oppsettene med prisen og funksjonaliteten til en UPS.
- For å kunne se om ytelsen på de ulike oppsettene er forskjellig fra ytelsen ved bruk av lokale disker, vil det bli målt ytelse i MB/s og IOPS.
- Ytelsen som blir målt må kartlegges og kunne presenteres på en slik måte at den blir leselig, dvs. at målingene må settes opp på grafisk form slik at man kan få en oversikt over ytelsen for de ulike ytelesestestene som blir kjørt på de ulike serverene.

**Serverbasert**

- GlusterFS skal være implementert over frontendserverene slik at man får en form for high availability lagring.
- Om det er mulighet for å kunne skifte ut RAID controlleren i serverene som er utlånt av Høgskolen i Gjøvik.

- Oppsettene skal være kompatible med det nåværende SkyHiGh miljøet, dette tilsier at oppsettene må basere seg på en Linux distribusjon. I dette tilfelle vil dette være Ubuntu Server 13.10 “Saucy Salamander”.
- Som caching device skal det brukes en SSD på frontendserverene.
- Backend lagring skal være et RAID.
- Det skal kartlegges om serverene støtter 2TB 3,5” HDD i Backend RAID’et.

## 2.3 Teori

### Bcache

Bcache er en cache-løsning som tar i bruk SSD som en cachingsenhet, noe som er med på å skape en hybrid lagringsløsning. Vi får her benyttet en cachingsenhet som kan cache opp til hele lagringsplassen som SSD'en kan bidra med. Dette fører til at man kan bruke SSD'en sin hastighet til å mellomlagre data for å senere skrive dette til HDD, noe som gir en betydelig ytelsesforbedring. Bcache er en linux kernel kompatibel block cache, derav navnet bcache.

Bcache vil ta imot random writes fra systemet, og deretter gjøre de sekvensielle når de skrives til den "trege" lagringsenheten som er bakenforliggende. Det bør merkes at en SSD har ekstremt mye bedre ytelse når det kommer til å håndtere random writes kontra en HDD. I en HDD's tilfelle vil det optimale være å lagre data sekvensielt.

Bcache har støtte både for write-back og write-through funksjonen, men kan også støtte write-around. Write-trough funksjonen er den som er satt som default når du implementerer en slik løsning, og blir brukt på en slik måte at det kan bli en ytelsesreduksjon dersom man cacher bare skrevet data.

Bruker man derimot en write-back policy istedenfor, vil man få mer data lagret på SSD for å senere få dette skrevet til HDD. Dette blir gjort på en seek-vennlig måte, noe som er bra egnet for bruk av HDD. Dette fører også til at man kan bruke bcache som en I/O scheduler i tillegg. Dette er en god måte å sikre at data når både SSD og HDD, siden data ikke vil bli merket som utført før den har nådd begge enhetene. Bruker man denne policien kombinert med å batch-skrive til HDD vil dette gi en god flyt til et bakenforliggende RAID ved at man som nevnt får skrevet data sekvensielt til HDD.

Det bør også nevnes at all caching som blir utført blir håndtert på blocklevel, noe som gjør denne teknologien uavhengig av filsystemetet som også kjøres, forutsatt at filsystemet bruker UUID. Dette er fordi man vil oppgi en UUID til enhetene som det skal caches til, noe som også er vanlig å gjøre for Linux når det kommer til filsystemer og swap-partisjoner. Data som caches i bcache kan bli redusert fra deres opprinnelige størrelse ned til HDD's sektor størrelse. [5, 6]

Bcache har tre forskjellige caching-mekanismer:

- write-through
- write-back (wrtie-behind)
- write-around

Write-through betyr at data skrives til cache og backing device synkront. Poenget med dette er at man ikke mister data dersom det skjer en kræsj, og man har rask tilgang til data som ligger i cache. Den andre er write-back, eller write-behind, som betyr at man skriver data fra cache til disk når cache er mindre opptatt, eller at blokkene med data er i ferd med å måtte byttes ut. Dette for at ytelsen ikke blir redusert ved skriving til backing device, og det er denne metoden som skal brukes i det oppsettet som skal settes opp. Den siste metoden, write-around, gjør det som navnet sier, nemlig at data ikke skrives til cache, men direkte til backing device, der det kun er lesing fra disk som caches. [4]

## FIO - Flexible I/O tester

For å kunne måle ytelse på diskene kan man benytte seg av et program som kalles FIO. Programmet i seg selv ble laget av Jens Axboe. FIO er med på å simulere workload som kan forekomme på diskene som skal tas ut i produksjon. Den utfører operasjoner som er av typen read eller write. Herunder kan kan spesifisere sekvensiell eller random access metode, noe som igjen er viktig å ta hensyn til når man har ulike typer disker. For eksempel vil en HDD ha en bedre ytelse dersom data som skal inn på disken kommer i en sekvensiell rekkefølge, mens en SSD vil kunne håndtere random writes bedre. FIO i seg selv er fleksibelt når det benyttes både for blockdevices og filer.

FIO simulerer I/O på to måter, ved bruk av tråder eller forks, for å kunne utnytte mesteparten av ressursene til serveren, og dertetter utfører en rekke writes eller reads til disken. Outputet som fio produserer er detaljert og gjør det lettere for oss å kunne prosessere data. Noe av den informasjonen man får er IO type, block size, IO size, IO engine, IO depth, num files, num threads. For mer informasjon kan man oppgi flere parametre for å få det ønskede resultatet. [7, 8, 9]

### GlusterFS

GlusterFS ble først utviklet av Gluster Inc., og senere videreutviklet av Red Hat etter oppkjøpet i 2011. GlusterFS er i hovedsak et NAS(network-attached filesystem) filsystem. Dette innebærer at man kan koble sammen en rekke ulike servere over nettverket, uten at de trenger å ha noe felles utenom data fra GlusterFS. Med GlusterFS kan man bevare integriteten av data-ene ved hjelp av replikering over to eller flere servere, man kan sette opp en form for lastbalansering ved hjelp av distribuering av data, og man kan sette opp striping. Både striping og distribuering av data vil føre til bedre ytelse.

I oppsettet vil det kun bli brukt distribuering og replikering, hvor replikering av data skjer over to servere, og distribuering skjer mellom fem mapper på hver av de to serverne. All data som blir lagret vil ligge på to steder, og dermed sørge for at mindre data går tapt dersom en eventuell kræsj forekommer. Ved hjelp av replikering vil også clusteret kunne bygge seg opp igjen, ved at data blir overført mellom serverne, hvis for eksempel den ene serveren skulle gå ned under produksjon. [10, 11]

### HDD vs SSD

Nå som markedet har introdusert nye harddisker i form av SSD, kan spørsmålet stilles om hva som bør brukes og hvor de ulike diskene er bra egnet. Hver type disk har sine bruksområder, og har sine ulike funksjonaliteter. Det som ble valgt var at den "treige" lagringen skal brukes i den form at man tar ibruk 3,5" HDD (Hard Drive Disk). Men for å kunne øke hastigheten man kan behandle data i, er det implementert SSD (Solid State Drive) som cache. Dette gir oss forskjellige fordeler og ulemper som rapporten nå skal ta for seg.

I oppsettetene som er valgt å sette opp tas det ibruk Western Digital Red 2TB 3,5" HDD og
Intel SSD s3700. Grunnen til at det er valgt en Intel SSD s3700 er at den er testet på et enterprise nivå, noe som også den billigere s3500 er, men s3700 viser mye bedre ytelse når det kommer til håndtering av random writes. Dette er igjen noe som gruppen og oppdragsgiver mener er viktig for ytelsen til oppsettene som skal testes. Denne påstanden er noe som har gått igjen hos de ulike brukerene av produktet. Når det kommer til målingen av ytelse som grunnlag for valget, kan man ta hensyn til storagereveiw sin analyse og sammenligning, det vil her si at s3700 yter bedre enn s3500, spesefikt på, som nevnt, random writes. Ved at hele poenget med en SSD som cache er at man skal ta inn random writes fra oppsettet og konvertere dette til sekvensielle writes til HDD ble dette et naturlig valg. [12]

Dersom man skal ta hensyn til pris, er HDD den klare vinneren som skal benyttes til lagring. Man kan se at prisene for de forskjellige teknologiene, i vårt oppsett, har en vesentlig forskjell. Det man kan se her er at SSD'ene som brukes er av typen Intel SSD DC S3700 serie, denne ligger på ca. kr. 1554 for 100GB (Gigabyte), noe som gir oss en pris på 15,54 kr pr. GB.

Sammenligner man denne prisen med en HDD som brukes i oppsettet, Western Digital Red NAS 2TB, vil denne ligge på ca. kr. 800 noe som igjen gir en pris på 0,4kr pr. GB. Dette vil si at skulle man velge ut fra pris ville man bare benyttet seg av HDD for å kunne spare mest mulig ressurser. Det bør allikevel presiseres at prisene på HDD varierer i forhold til om man har 2,5" eller 3,5" disker.

Skal man heller se på overføringshastighet i form av read / write operasjoner vil man kunne se at en SSD er langt raskere enn en "vanlig" HDD. I vårt tilfelle vil SSD ha en øvre grense på ca. 293 MB/s read og 125 MB/s write på random 4k blokkstørrelse skrivinger, kontra en HDD som ligger på 0,177 MB/s read og 0,440 MB/s write. Tar man også hensyn til sekvensielle skrivinger og lesinger kommer en HDD noe bedre ut med en 142 MB/s read og en 143 MB/s write. Men en HDD taper forsatt mot en SSD på dette området da en SSD ligger på 500 MB/s read og 200MB/s write. Men ser man derimot på hvilket behov man har kan man se at HDD på sekvensielle skrivinger har en mindre differanse med en SSD motpart.

Skal man så se på noen funksjonaliteter rundt disken vil man kunne se at en HDD sliter med fragmentering, da filer kan deles opp og legges på forskjellige plasser på den spinnende disken, noe som gjør at lesearmen må flyttes mye og kan medføre reduksjoner i ytelse på disken. Dette er noe en SSD ikke trenger å bry seg om siden denne ikke har noen lesearm.

Når man snakker om deler innenfor en disk, er det verdt å nevne at en HDD har en spinnede disk internt i disken, noe som gjør at den fysiske størrelsen på disken ikke kan gjøres stort mindre. En SSD har ikke dette problemet, og kan derfor bli mindre i størrelse dersom dette er ønskelig i fremtiden. Når man ser på bevegelige komponenter innenfor disken, kan man også se at en HDD er mindre rustet mot støt og sjokkskader som som kan sette platen i ubalanse og dermed gjøre disken ubrukelig, dette er også noe en SSD har kvittet seg med. For de som også vil bry seg om lyd som kommer fra disken vil man ha et betydeligere høyere støynivå på HDD fremfor en SSD.

Men en SSD har også noen ulemper, blant annet med at den er mye mindre tilgjengelig enn en HDD. Det bør også merkes at en HDD maks lagringskapasitet per disk i dagens marked vil ligge på rundt 4TB, mens en SSD vil kun komme med en 1TB disker.

Så hvorfor skal man ikke blankt kan velge en SSD på alle områder når det gjelder lagring, og hva bør man tenke på når man skal sette opp en lagringsløsning ?

For å oppsumere litt så har en HDD bedre tilgjengelighet på den fysiske disken, og kan gi flere GB pr. NOK, samt at man får mer lagring pr disk. Mens en SSD vinner i ytelse og har i tillegg et lite ekstra sikkerhetslag når det gjelder mobilitet, altså den er mer støtsikker.

Man vil i alle nevnte tilfeller ha begrenset med ressurser som kan benyttes for å opprette en "lagringsboks" (sammensetning av ulike hard disker). Når man tar hensyn til dette vil en HDD egne seg bra for et lite mobilt oppsett, samt at det vil egne seg når man trenger mye lagringsplass. Men skal man fokusere på ytelse og mobilitet vil man helst benytte seg av en SSD løsning.

I dette tilfellet er det brukt HDD som en backendløsning samt at det er benyttet en SSD som skal cache til disse HDD'ene. Dette vil medføre at man får benyttet HDD på det bruksområdet den er best på nemlig sekvensielle skrivinger, og man får benyttet en SSD på det området den er best på nemlig random read / writes. Dette vil si at det benyttes en SSD til å ta imot random read / writes, gjøre disse om til sekvensielle skrivinger / lesinger som skal overføres til en HDD. Ved å benytte en arkitektur som denn, kan man få utnyttet diskene på det området de er best på, samt at man kan spare ekstra kostnader som en full SSD-implementasjon vil medføre.

Når man ser på lagringskapasistet i en produskjonssammenheng vil man trenge store disker som kan håndtere mye data. For å kunne utnytte en best mulig GB pr harddrive-slot på server, vil man også benytte seg av en HDD backend lagringsløsning. Når man tar hensyn til disse faktorene vil man komme frem til en tilnærmet optimal løsning når det kommer til ytelse, NOK pr. GB, og lagringskapsitet. [13, 14, 15, 16, 17]

### Hdparm

Hdparm er hovedsaklig et verktøy i linux kommandolinje som brukes for å vise informasjon om ATA harddisk drive hardware parametere. Ved hjelp av hdparm kan man også endre på disse parameterne. I tillegg til dette kan også hdparm brukes for å måle ytelsen til en valgfri hard disk som sitter i serveren du bruker. En ulempe ved kommandoen er at å bruke noen av parameterne vil føre til kræsj, som igjen kan gjøre at data blir utilgjengelige. [18]

### High Availability Storage

Dette innebærer at lagringen ikke går ned av en single point of failure, men har muligheten til å være tilgjengelig selv om en server skulle gå ned. Når det kommer til implementering av high availability storage, kan det bli tatt ibruk software som for eksempel GlusterFS, som kan replikere data utover forskjellige servere. Når det kommer til selve high availability storage generelt sett, er det et mangfold av tiltak som kan gjøres, men å drøfte disse tiltakene blir for omfattende for denne oppgaven. [19, 20]

### iSCSI

iSCSI - Internet Small Computer System Interface, er primært en IP-basert standard brukt for sammenkobling av lagringsenheter(lagringsbokser). Poenget med iSCSI er at man kan sende SCSI kommandoer over IP-nettverk. iSCSI kan brukes både i LAN, WAN og over internett.

iSCSI virker på den måten at man har en *initiator* og en *target* i et nettverk. En initiator, eller enklere sagt en klient, er den som setter igang dataoverførselen. En target, eller lagringsenhet, er der hvor man henter ut informasjonen fra. For at iSCSI skal virke, trenger man derfor minst en target og en initiator i et nettverk. iSCSI er en SAN-protokoll.

iSCSI bruker CHAP som autentiseringsmekanisme mellom initiator og target. CHAP i seg selv er sårbar for ordbok-angrep, spoofing og reflection attacks. Dette kan imidlertid unngås hvis man implementerer det på en god måte. I tillegg til CHAP kan man sette opp sikkerhet i nettverk-laget. Det er mulighet for implementasjon av andre sikkerhetsmekanismer også, men det er begrenset i forhold til systemenes mulighet til å samarbeide. [21, 22]

## KVM/QEMU

Kvm - Kernel based Virtual Machine/QEMU - Quick EMUlator, er to teknikker man bruker sammen for å kunne bruke kjernen i linux, men også andre operativsystemer, som hypervisor for virtuelle maskiner. For at KVM skal kunne kjøre, trenger man en prosessor med støtte for hardware virtualisering, men den har også støtte for paravirtualisering hvis dette er ønskelig. Et av hovedpoengene for å bruke KVM er at det er åpen kildekode, og at man med virtualisering kan kjøre flere virtuelle maskiner på samme vertsserver, hvor hver virtuell maskiner har sitt eget nettverkskort, virtuelle disker og grafikk-kort, for å gi noen eksempler.

KVM alene kjører ingen emulering, men gjør istedet /dev/kvm sitt grensesnitt synlig for de virtuelle maskinene. Poenget med å bruke QEMU som overliggende system er at QEMU bruker KVM for å snakke med hardware på vertsserveren, og dermed kan de virtuelle maskinene kjøre med tilnærmet samme hastighet som vertsserveren. Hvis ikke KVM er tilstede på vertsserveren, vil QEMU gå tilbake til å kjøre software emulering. KVM/QEMU er støttet i lbivirt, forklart videre nedenfor. [23, 24]

## Libvirt

Libvirt er et grensesnitt for å administrere virtualiseringsplatformer, med fri kildekode, og derfor et godt verktøy å bruke på ubuntu-server ved virtualisering. Grensesnittet støtter en rekke underliggende systemer, som for eksempel Xen, VirtualBox, VMware og KVM/QEMU. Libvirt i seg selv er oppbygd av et C-bibliotek, men har også sammenkoblinger til andre programmeringsspråk som Python, Perl og Java. Vi skal bruke libvirt for å få et "penere" grensesnitt over KVM/QEMU. [25, 26, 27]

## LVM

LVM - Logical Volume Manager, er et begrep som brukes når man snakker om moderne operativsystemer. Koden til LVM ble skrevet av Heinz Mauelshagen i 1998, hvor hovedinspirasjonen ble hentet fra HP-UX's volume manager. LVM er i all hovedsak et usynlig lag som ligger over langringsenhetene i en datamaskin, og holder orden på de logiske diskene som er opprettet i operativsystemet.

LVM har en rekke funksjoner som hjelper til i de fleste former for langring. Blandt disse funksjonene finner man konsistent backup, snapshots av logiske disker, logiske disker kan settes opp med samme trekk som RAID0 og JBOD. Funksjonen i LVM som er mest attraktiv for vårt

tilfelle, og som gruppen mener er det beste, er at man ved et senere tidspunkt kan øke lagrings-kapasiteten ved å legge til flere disker. I tillegg er det mulig med hot swapping. [2, 28]

### QCOW

QCOW - QEMU Copy On Write, er et alternativt filformat på disk image for opprettelse av virtuelle maskiner. QCOW bruker en lagringsstrategi som utsetter skriving helt til det er nødvendig, det gjør at lagring på disken blir optimalisert. Det er idag to ulike versjoner av QCOW-filer, qcow og qcow2, og de respektive disk image'ene har derfor filendinger .qcow og .qcow2.

En av de store fordelene med å bruke QCOW som filformat, er at filene kan fortsette å vokse ettersom mere data blir lagt til. I tillegg til dette har hver av de virtuelle maskinene kun en kopi av OS'et som kjører, og et eget private område hvor de kan lagre sine endringer. Fordelen med sistnevnte er at man får mindre bruk av disk, ettersom man kun har en kopi av OS. Dette fungere slik at man har basen av OS'et i en qcow-fil, og hver virtuelle maskin har sin egen qcow-fil som linker tilbake til OS'et sin qcow-fil.

Forskjellen mellom de to versjonene av QCOW er at ved qcow2 kan man effektivt ta snapshots av en virtuell maskin ved hjelp av en ny fleksibel lagringsmekanisme for snapshots. Siden det er to versjoner av QCOW, har KVM og QEMU tatt vare på bakoverkompatibiliteten, slik at begge versjonene virker på dagens software og hardware. QCOW har også tilleggsfunksjoner som AES-kryptering og zlib-basert komprimering. [29]

### RAID

RAID - Redundant Array of Independent/Inexpensive Disks. Er en metode for å logisk koble sammen disker, slik at man kan få både økt ytelse og redundans. De typene RAID brukt i våre oppsett er som følgende: [3]

- 0 - striping av data over det antall disker som settes i RAID. Minimum to disker for å bruke RAID0.
- 1 - speiling/replikering av data, to og to disker speiles om hverandre. Minimum to disker for å bruke RAID1.
- 10 - bruker en kombinasjon av både RAID1 og RAID0, i den rekkefølgen. To og to disker blir speilet om hverandre med RAID 1, hvor det så blir gjort striping ved RAID0 over hver av disk-parene opprettet av RAID1. Minimum 4 disker for å bruke RAID10.

### SAN

SAN - Storage Area Network, er et dedikert lagringsnettverk. Her er disker, tape eller andre lagrings-medium satt i et nettverk som kommuniserer med en eller flere servere. Kommunikasjonen gjøres på en slik måte at servere får listet disse lagringsmediumene som om de skulle ligge lokalt på serveren. Typisk sett er det tre forskjellige oppkoblingsløsninger som brukes i dag, det er iSCSI over TCP/IP, Fibre Channel over Ethernet (FCoE), eller en direkte Fibre Channel løsning.

Ved å sette opp et SAN vil kun utvalgte servere kunne ha tilgang til lagringsenhetene, noe som er med på å isolere data til kun gitte servere, som igjen gir en liten form for sikkerhet. Når det gjelder kommunikasjonsprotokollene kan man også opprette brukere slik at man får en autentiserings-mekasnisme som støtter opp mot isoleringen dersom enheter skulle få tilgang til SAN nettverket.

Et SAN vil operere hovedsaklig på blocklevel og har kun aksess til blokker. Ved å innføre et SAN OS eller et SAN filsystem kan man få tilgang til ulike filer på filnivå.

SAN vil dele lagringsplassen seg imellom slik at man kan få utnyttet alle "lagringsboksene" fra en enkelt server. Dette er med på å skape god fleksibilitet og er med på å fremme cluster-arkitekturen. Noe som også fører til at man kan boote devices fra SAN nettverket dersom dette er nødvendig.

Ved å clustre flere elementer innenfor et SAN vil man ha muligheten til å skape redundans i den form av at man får fordelt lik data utover alle diskene som er i nettverket. Dette gjøres ved at man benytter seg av replikering- og stripingsteknologi, noe som er optimalt ved bruk av et RAID oppsett.

Ved å bruke et SAN åpner det for muligheter for bruk av LVM og andre logiske partisjonerings-mekanismer. Det er med på å organisere lagringsstrukturen slik at man kan fordele gitte logiske partisjoner til de som trenger det.

Noen ulemper ved å ta ibruk et SAN er at elementene som er med på å definere QoS er elementer som raskt kan bli ustablie, nemlig båndbredde, latency og kø-dybde. [30, 31, 32]

### XFS

XFS er et 64-bits journalførings- og filsystem. I IRIX utgivelser fra versjon 5.3 og utover var XFS standard, og det har senere blitt portert over til Linux. XFS er spesielt god på parallell I/O på grunn av bruken av tildelingsgrupper. Årsaken til at det er så effektivt er fordi hver tildelingsgruppe er ansvarlig for hver sine inoder og ledig lagringsrom separat, noe som igjen medfører at mange tråder kan gjøre I/O-operasjoner på samme filsystem samtidig.

XFS sikrer at dataene forholder seg konsistent ved at metadata stadig blir journalført, samt at XFS bruker noe som kalles "write barriers".[33, 34] Disse sørger for at filsystemet skriver dataene til disk på riktig måte, og i riktig rekkefølge. Datalagringen blir gjort vha. B+ trær, som øker ytelsen til selve filsystemet, og spesielt ved håndtering av store filer. Delayed allocation, eller allocate-on-flush, hjelper til med å forebygge fragmentering. Måten dette gjøres på er at hvis data skal skrives til disk, men det er mye trafikk, kan mengden data bli trukket ifra free-space teller, men ikke i free-space bitmap.[35, 36]

I tillegg til de nevnte spesifikasjonene har XFS striping, variable block-størrelser, direkte I/O og DMAPI. Striping øker gjennomstrømning, variable block-størrelser tillater personlig tilpasning, direkte I/O fører til direkte skriving til disk. DMAPI, Disk Management API, og brukes for å støtte hierarkisk data-administrasjon. [37, 38, 39, 40, 41]

### ZFS on Linux

ZFS on Linux er en ZFS løsning som kan kjøres native i linux. Dette innebærer at man kan få funksjonaliteten til ZFS på en linux server. Denne formen for bruk av ZFS er produsert av Lawrence Livermore National Laboratory, men selve ZFS er laget av Oracle Corporation.

For å kunne forstå hva ZFS er må man se på hva som ligger i funksjonaliteten til ZFS i seg selv.

ZFS er i all hovedsak en kombinasjon av et eget filsystem og en logical volume manager (LVM). Dette systemet ble designet av Sun Microsystems og har primært blitt benyttet på Solaris systemene.

ZFS har et mangfold av funksjonaliteter, men primært sett for vår oppgave har det noen hovedmomenter som er viktige for utførelsen. Dette innbærer funksjonaliteten som omhandler beskyttelse mot datakorrupsjon, støtte for store lagringsmedier, selve kombinasjonen av en LVM

og et filsystem, muligheten for å ta snapshots av systemet, automatisk reparasjons og kontinuerlig sjekking av dataintegritet. [42]

Hvordan ZFS beskytter seg mot datakorrupsjon og bevarer intergritet:
ZFS vil bruke en checksum eller en hash gjennom hele filsystemet, der hver block er blitt gjort en checksum på og verdien til checksummen er lagret i en peker til den gitte blokken. Deretter vil pekeren bli kontrollert av checksum og verdien vil bli lagret til dens peker. Slik forsetter sjekkingen til og med root-noden. Med andre ord så vil ZFS lagre checksum til en gitt node eller block i dens foreldre-node og på den måten vil alle nodene kunne sjekke seg selv.

Videre har gruppen lyst til å skrive litt om noen momenter å tenke på når man skal sette opp ZFS on Linux. Som det ser ut av den følgende linken([43]) er det foreløpig ingen stabil versjon av ZFS on Linux for ubuntu-server 13.10, men dette vil rapporten komme tilbake til når installeringen skal gjennomføres.

Det er en rekke forskjellige /dev/-navn man kan bruke ved opprettelse av ZFS storage-pools. I hovedsak er det fire forskjellige navnemetoder man kan bruke, som er de følgende:

- /dev/sdX/, /dev/hdX/
- /dev/disk/by-id/
- /dev/disk/by-path/
- /dev/disk/by-vdev/

Rapporten kommer til å vise en overordnet diskusjon av hvilken navneordning som er den riktige i forhold til vår situasjon.

Når det gjelder det første alternativet listet, /dev/sdX, vil nok dette være det som passer best i vårt prosjekt. Dette er fordi /dev/sdX ofte er brukt for testing og utvikling, noe som gjør det enkelt og raskt å bruke. Navnene som brukes er korte, og det er tilgjengelig i alle linux-distribusjoner. Det er også noen ulemper. Navnene kan endres i forhold til hvilken rekkefølge diskene oppdages i, samt at ved addisjon eller fjerning av disker kan navnene også endres. Selv om det er et par "store" ulemper med denne metoden, mener gruppen at det ikke er av stor betydning for utførelsen av bacheloroppgaven.

/dev/disk/by-id/ er det lurt å bruke dersom man har mindre enn 10 disker. Ved denne metoden har man identifisering med noe mer forståelige beskrivelser sett fra menneskenes side. Identifikasjonen inneholder eksempelvis interface type, merke(les: selger), modellnummer og serienummer. Her spiller det ingen rolle hvordan diskene er koblet til systemet, ettersom det er garantert at de ikke endrer navn. En ulempe er at navnegiving basert på geografisk lokasjon kan være vanskelig og føre til feil. Hvis ikke det første alternativet er tilstrekkelig godt nok, vil dette være vårt andrevalg.

/dev/disk/by-path/ og /dev/disk/by-vdev/ er gode alternativer for oppsett med flere enn 10 disker, hvorav /dev/disk/by-vdev/ er det beste alternativet ved høyere diskantall. Hovedforskjellen ved disse er at /by-path/ har beskrivelser som inneholder det fysiske utseende av systemet, koblet til en spesifikk lokasjon, mens /by-vdev/ baserer seg på betydningsfulle navn for administratoren. /by-path/ kan gi lange og vanskelige navn, mens /by-vdev/ krever at man har fila for vdev_id.conf riktig konfigurert.

Med tanke på ytelse, både når det gjelder backend og frontend, er det noen ting man må tenke på ved utrulling av ZFS. For det første er det viktig å tenke på hvordan man plasserer disker i forhold til kontrollerne. Ved "feil plassering" av diskene kan man få nedsatt ytelse på systemet. En annen ting å ta hensyn til i forhold til ZFS, er partisjonering av disker. Med dette menes at man bruker hele disker når man kjører kommandoen *zpool create*, slik at ZFS kan partisjonere disken riktig, og at andre deler av ZFS kan fungere slik de er designet til å gjøre.

Som tillegg til de to nevnte punktene ovenfor, må man tenke på RAM, samt at man kan øke ytelsen for spesielle typer data(workloads). ZFS har som minimumkrav at systemet har 2BGRAM, men det er anbefalt at man har noe mer hvis man bruker kompresjon og deduplisering. Ved å sette *ashift=x*, hvor x kan være et tall mellom 9 og 16, ved *zpool create* vil føre til at man "tvinger" ZFS til å bruke en bestemt sektorstørrelse. Størrelse regnes som 2'erpotenser, så *ashift=9* blir 512. I vårt tilfelle er det ønskelig med sektorstørrelse på 4K, og derfor vil man bruke "ashift=12", $2^{12}$ = 4 096, som tilsvarer 4KB. Det må merkes at denne parameteren kun kan gjøres første gang *zpool create* kjøres, og medfører litt nedsatt lagringskapasitet. [43]

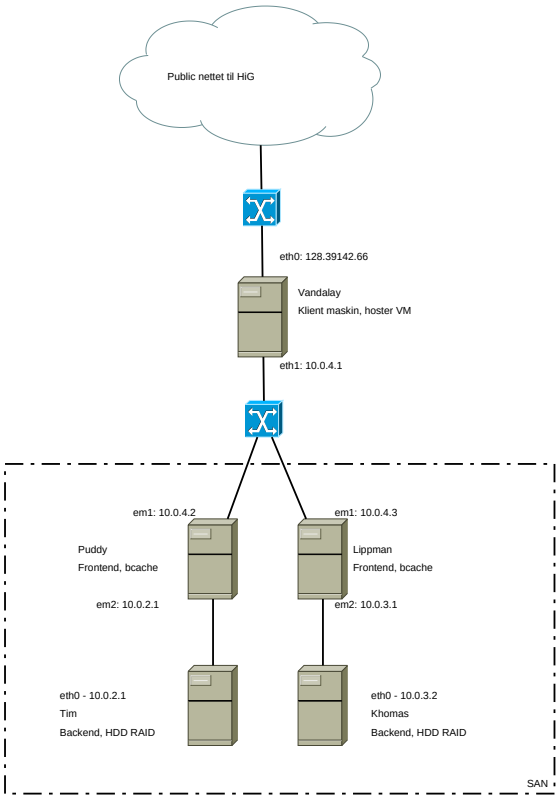
# Kapittel 3

# Design / Utstyr

## 3.1 Overordnet Oppsett

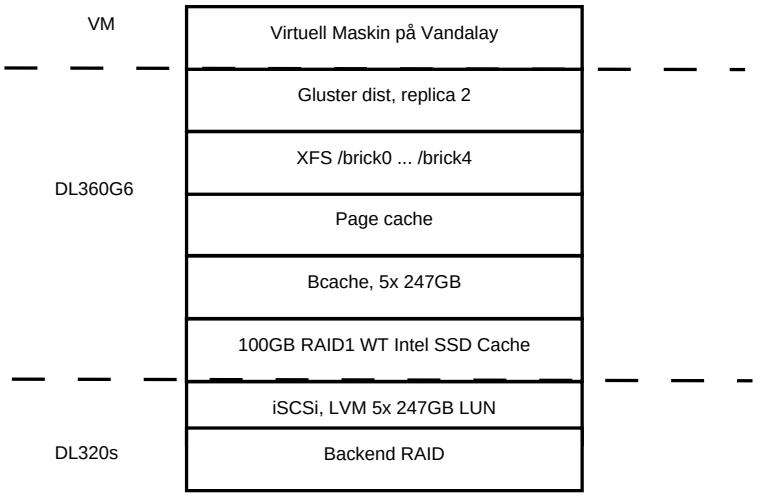
I løpet av dette prosjektet vil det bli benyttet en rekke servere som er utlånt av oppdragsgiver. Disse serverne skal brukes til å gjennomføre oppgaven som er tildelt fra oppdragsgiveren. Det har også blitt tildelt en rekke disker som skal benyttes. Noen HDD som skal brukes som backend lagring, noen HDD for OS'et på frontend, samt et par SSD som skal brukes til caching. Hvordan dette er satt opp og konfigurert vil rapporten komme tilbake til.

Det vil hovedsaklig være to ulike oppsett, jfr. figur 3.2 og figur 3.3, men de er begge underlagt en generell arkitektur. Det som varierer er oppsettet internt på serveren, men ikke selve arkitekturen. Arkitekturen vil se ut som følgende:



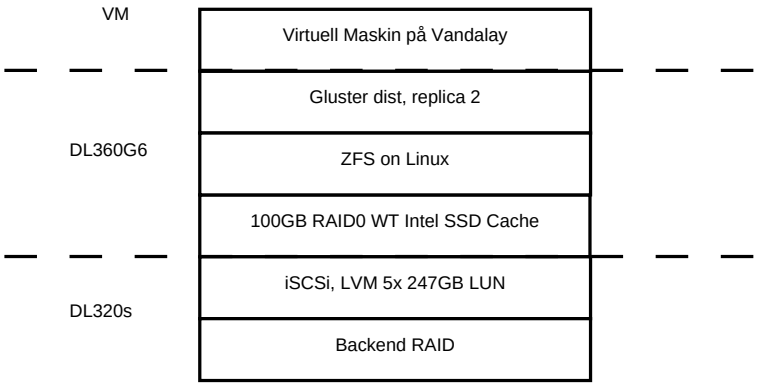

Figur 3.1: Overorndet arkitetktur

Herunder kommer, som nevnt, ulike konfigurasjoner, definert ved hjelp av dette oppsettet:




Figur 3.2: Lagvis inndeling oppsett 1

For å kunne sammenligne hva som eventuelt kan være mest effektivt for oppdragsgiver vil oppsettet i figur 3.2 testes mot dette oppsettet:




Figur 3.3: Lagvis inndeling oppsett 2

I tillegg til de to oppsettene som er beskrevet over, ble det forevist en mulighet til. I stedet for ZFS on Linux er det mulig å bruke btrfs, som har mye av den samme funksjonaliteten som ZFS on Linux. Hovedgrunnen for at det ble valgt XFS med bache i det ene oppsettet og ZFS on Linux i det andre, var delvis preget av oppdragsgivers ønsker, samt at det er lite poeng i å teste to relativt like filsystemer. Sammen med dette var det noe usikkert om ZFS on Linux er stabilt nok for Ubuntu Saucy(13.10) til at det kan settes ut i produksjon.

Et alternativ var å ofre noe av teknologien ZFS on Linux kunne gi oss, for så å kunne benytte et mer stabilt filsystem, nemlig btrfs. Etter å ha sett gjennom de nyeste oppdateringene som kom med ZFS on Linux, ble det funnet ut at ZFS on Linux ville bli stabilt nok til at det er akseptabelt for et produksjonsmiljø. Resultatet av dette ble at valget av filsystem falt på ZFS on Linux fremfor btrfs. Noe av det som også bidro til at valget falt på ZFS on Linux er at dersom man skal ta hensyn til erfaringer andre har hatt med de ulike systemene, er det tydelig at de som jobber innen fagfeltet foretrekker ZFS on Linux. Dette i seg selv er med på å bidra til å gi en mer detlajert oversikt over hvilke problemstillinger som kan forekomme når man setter opp dette filsystemet.

Her bør det også merkes at opprinnelig er ZFS on Linux en modifikasjon av ZFS, som er designet for Solaris systemet, men har i senere tid blitt gjort kompatibel med Linux kernel. Det som er en gjenganger i grunner til å velge ZFS on Linux over btrfs, er at btrfs gir mer støtte for RAID 5, noe som eksisterer den dag i dag, men brukes sjelden fordi det kan medføre problemer. Denne problematikken vil ikke bli drøftet i vår oppgave, da våre RAID skal være RAID0, RAID1 og RAID10.

Men det bør tas hensyn til at ved å benytte ZFS on Linux, eller bare ZFS generelt sett, kan det føre til at ytelsen kan bli noe redusert, fremfor bruken av btrfs. På den andre siden er btrfs mer ustabilt i dagens tilstand på filsystemet. Dette er et inntrykk som går igjen når man ser på helheten over de to ulike systemene. Det vil si at mens btrfs bruker lenger tid på å oppdatere og implementere nye funksjonaliteter, vil ZFS on Linux komme med flere funksjonaliteter hurtigere, noe som gir et inntrykk av at ZFS on Linux utvikles og reitereres hurtigere, og dermed pusher mot en helhetlig og stabil release. [44]

For videre å kunne øke ytelsen, ble det undersøkt ulike teknologier som kan gi størst utbytte å endre. Blant dette er drive write cache, DWC, som bør være slått på for SSD, og slått av for HDD, dette er igjen noe som ville øke ytelsen når testene skal startes. For at dette kunne utføres, ble det

tatt ibruk hpacucli, og disse kommandoene ble kjørt:

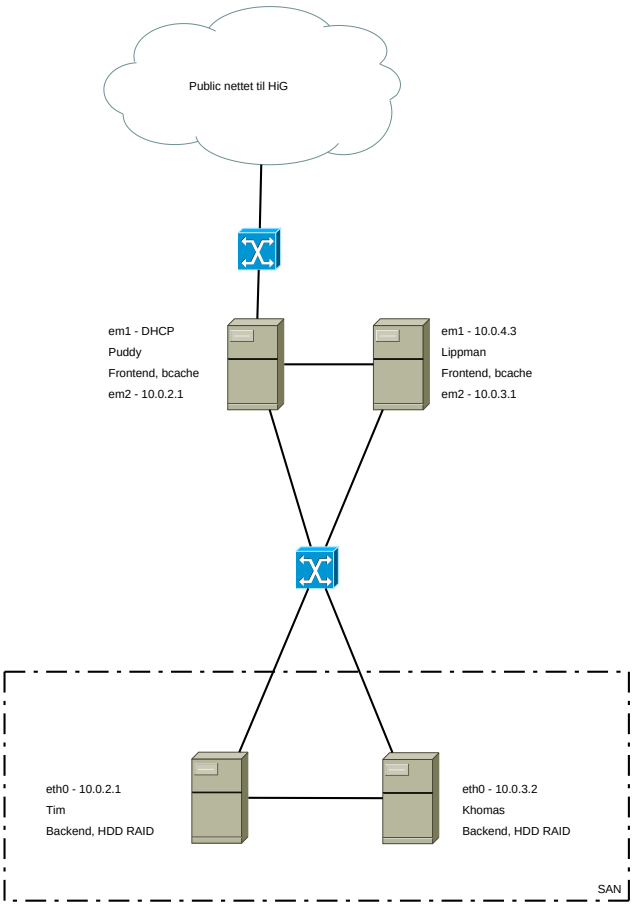
```
hpacucli controller all show detail
hpacucli ctrl slot=0 modify dwc=enable
```

Den første kommandoen vil liste de ulike diskene som er brukt i oppsettet, og den andre er konstruert på en slik måte at slot tar hensyn til hvor selve disken ligger. dwc=enable parameteren vil beskrive om drive write cache skal være på eller av. Kommandoene vil være like for alle serverene men slot og enable/disable vil være forskjellige i henhold til hvilke servere man tar hensyn til. For at man skal kunne sikre seg at RAID controller cachen er slått av, for igjen å få bedre ytelse, fjernes batteriet som står i serverene. Når dette batteriet er fjernet, antar gruppen at RAID controller cache er skrudd av. Det har ikke vært tid til å undersøke videre rundt dette, og påstanden kan derfor ikke underbygges. En annen mulighet er at cache mode for RAID controller cache endres til writetrough.

Majoriteten av data som skal til disk, går gjennom SSD cachen, og ikke RAM som ligger i serveren. Page cachen eller onboard cache er ikke noe man kommer unna, og denne vil bli benyttet, men i veldig liten grad. Som nevnt vil SSD ta imot random writes, gjøre det om til sekvensielle writes for så å sende det videre til den bakomforliggende HDD.

For å oppsummere litt vil man ha enablet DWC på frontend(puddy og lippman), og disablet denne for backend(tim og khomas). Deretter vil man også benytte onboard cache i minst mulig grad, siden dette kan skape ekstra overhead, og fordi denne cachen er volatile(mister data ved strømbrudd). Man vil disable RAID controller cachen på alle serverene ved å fjerne batteripakken som holder denne i live. Hovedgrunnen for å skru av DWC er at man kan miste data dersom RAID controller mister strømmen. Fordelene ved bruk av Intel SSD s3700 er forklart i kapittel 2.3 - Teori.

Dette er den arkitekturen som ville bli mest passende i henhold til hva oppdragsgiver spesifiserte, jfr. figur 3.1. Men ulike arkitekturer har også blitt drøftet, hvor alternativet til den endelige modellen er vist nedenfor.




Figur 3.4: Alternativ arkitektur

Denne arkitekturen vil være ineffektiv, samt at det generelt strider mot best practice i henhold til å ha et SAN på et eget nettverk for å unngå unødvendig støy fra andre kilder som er koblet til det samme nettverket. Man kan se dette ved at begge backendserverene er koblet opp mot samme switch, og de lett kan bli påvirket av støy fra andre kilder ettersom andre servere også kan være tilkoblet samme switch. Et eksempel på slik støy kan være at den ene backendserveren gjør en update mens den andre backendserveren er i produksjon, noe som vil medøre nedsatt ytelse i den perioden hvor update blir gjennomført.

Grunnen til at denne arkitekturen ble vurdert, var at man kunne få testet dette i et eget testnettverk, men dersom denne arkitekturen skulle bli satt ut i produksjon ville det trengs revidering av arkitekturen. Ved å følge den første arkitekturen(jfr figur 3.1) ville oppgaven bli mer passende for oss, i tillegg til å få en tilnærming til hvordan dette ville være når det eventuelt blir satt ut i produksjon.

For å ha et produksjonsklart element etter endt bacheloroppgave, ville det heller bli tatt hensyn til de ulike metodikkene som ville reflektere et oppriktig miljø, det vil si forholde oss til best practice som er høyt utbredt blant et virkelig produksjonsmiljø. Dermed ble oppsettet revidert, slik at gruppen endte på det førstnevnte(jfr. figur 3.1) oppsettet kontra det andre(jfr. figur 3.4).

Ved oppkoblingen av serverene har man egne farger på TCP/IP kablene for å klare å skille mellom de ulikene intranettene. Dette gjør at man lettere kan identifisere hvordan serverne er koblet sammen, samt at det er lett å holde oversikten over oppsettet ved å kun se på fargene på ledningene.

## 3.2 Maskinvare

| Modell: | DL380 G5 (Klient) | DL360 G6 (Frontend) | DL320s (Backend) |
|---|---|---|---|
| CPU(antall): | 4 | 8 | 2 |
| RAM(GB): | 22 | 12 | 4 |
| HDD/SDD (antall): | 2 stk HDD | 1stk 100GB Intel s3700 SSD, 2stk HDD | 12 HDD |
| RAID: | RAID1 | RAID1, RAID0 | RAID1, RAID10 |
| Linux kjerne: | 3.11.0 | 3.11.0 | 3.11.0 |
| Distribusjon: | Ubuntu Server 13.10 | Ubuntu Server 13.10 | Ubuntu Server 13.10 |
| Hypervisor: | KVM | – | – |

På klientmaskinen, vandalay, som kjører de virtuelle maskinene i oppsettet vil man ha operativsystemet på de lokale HDD'ene som kjører i RAID1 slik at selv om én disk ryker vil ikke serveren gå ned, eller miste integriteten til operativsystemet.

Frontendserverene, puddy og lippman, vil ha hver sin SSD som settes i RAID 0, og harddisker som settes i RAID1. SSD vil brukes som nevnt som en cache device opp mot den trege backend lagringen.

Backendserverene, tim og khomas, vil ha 2 HDD (Seagate Barracuda ES 250GB) satt i RAID1 for operativsystemet, og 10 HDD satt i RAID10 (1+0). RAID10 er den trege lagringsløsningen som benyttes på backendserverene.

## 3.3 Pakkeforklaringer

Under kommer en kort forklaring på hva de enkelte pakkene bidrar med og hvilken funksjonalitet de har. Det bør merkes at dersom det ønskes mer informasjon om hvordan disse brukes eller hvordan de er laget kan man anvende man-sidene som følger med disse pakkene. Der er det skrevet en detaljert informasjon.

attr:

Dersom man skal trenge å unmounte GlusterFS på en korrekt måte, kan man benytte seg av denne pakken. Den setter 0 på kritiske attributter slik at GlusterFS blir unmountet på en "ren" måte. [45]

bcache-tools:

For å kunne implementere selve bcache-funksjonaliteten, må denne pakken benyttes. Denne pakken gir støtte for make-bcache kommandoene som er kritisk for å kunne benytte en SSD som en cache-enhet. [46, 47, 48, 49]

cpu-checker:

Denne pakken gir støtte for kvm-ok kommandoen som er for å sjekke om hardware virtualisering er mulig.

fio:

Denne pakken gir tilgang til fio-kommandoene som blir kjørt på denne løsningen og brukes for å kunne utføre ytelsestester i de to oppsettene. [9, 50]

glusterfs-client:

Når klientmaskinene skal benytte seg av lagringen som GlusterFS gir, må man ta ibruk denne pakken. Den gir mulighet for å mounte gluster-volumet på en lokal mappe og er med på å skape redundans for data som skal lagres. [51, 52, 53]

glusterfs-server:

Denne pakken er viktig å ta ibruk på frontendserverene slik at man kan opprette, vedlikeholde og overvåke volum med bricks innenfor et GlusterFS-nettverk. Dette gjør det også mulig å skape redundant data mellom serverene, dvs. at GlusterFS i seg selv vil distribuere og replikere data over de oppgitte nodene. GlusterFS fremmer også high-availability storage, noe som gjør at man vil ha tilgang på data selv om en server går ned for vedlikehold eller kræsjer. [51, 54]

hpacucli:

Ved å benytte seg av denne pakken på HP-servere får man tilgang til å enable/disable drive write cache på serverene og muligheten til å sjekke om RAID controller cache er slått på eller av. Informasjon angående den logiske eller fysiske inndelingen av diskene kan bli listet her. Det bør også merkes at dette er en HP-spesifikk pakke, noe som igjen tilsier at den passer til oppsettene, grunnet brukt HP-servere i arkitekturen. [55]

isc-dhcp-server:

Denne pakken gjør det mulig å sette opp en av serverene som en dhcp-server. Herunder kan man spesifisere hvilken interface dhcp-protokollen skal lytte på og dele ut ip adresser til, i tillegg til at man kan gruppere enkelte ip adresser som skal benyttes i internnettverket som løsningen skal stå i. Det bør merkes at denne pakken blir brukt i samsvar med den generelle ip-konfigurasjonen som settes opp på de enkelte serverene, i tillegg til serverenes ip-ruter. [56, 57]

libvirt-bin:

Dette er pakken som gir mulighet for å benytte seg av libvirt API på servere. Dette gjør det enkelere når det skal opprettes, slettes og håndtere virtuelle maskiner. [27, 58]

lvm2:

For å kunne utføre logisk partisjonering av større lagringsmedier benyttes denne pakken, det gjør det mulig å dele RAID10(1+0) inn i mindre partisjoner som kan benyttes til forskjellige formål. [28]

open-iscsi:

Dette er pakken som trengs for å kunne benytte iSCSI over TCP/IP kabelen, og kjent som software iSCSI. Denne pakken vil bli benyttet av “initiatorene” i et iSCSI oppsett. Denne blir brukt hoved-

saklig på frontendserverene slik at disse kan benytte seg av lagringskapasiteten til backendserverene. [59, 60, 61]

qemu-kvm:

Dette er pakken som håndterer virtualisering gjennom KVM, denne er nødvendig for å kunne opprette og i det hele tatt kjøre virtuelle maskiner ved hjelp av KVM hypervisor. [62, 63, 64]

screen:

Screen er for å kunne samarbeide i et felles vindu, dette gjør det mulig at flere medlemmer kan jobbe sammen om et gitt emne i et kommandolinje-format. [65]

software-properties-commmon:

Denne pakken er nødvendig for å legge til nye ppa-lenker i repository-fila til apt-get, ubuntu sitt pakkeinnstalleringssystem. Dette måtte gjøres for å kunne få tilgang til den nyeste versjonen av GlusterFS og bcache-tools.

targetcli:

Blir brukt av iSCSI target for å kunne sette opp luns som skal fordeles til initiator'ene, dvs. luns som skal legges inn i lagringsbokser. [66, 67]

virtinst:

Denne pakken gir tilgang til virsh-kommandoene som er essensielle for å kunne opprette, håndtere og konfigurere virtuelle maskiner. Her får man også muligheten til å klone de gitte VM'ene. [68, 69]

xfsprogs:

For å kunne opprette et XFS filsystem på en gitt path, eller mer presist en block-device, må denne pakken benyttes. Denne gir muligheten til å benytte seg av *mkfs.xfs*-kommandoen som lager selve filsystemet på enheten. Det bør også merkes at her må man ta ibruk parameteren *-i size=512* for at man skal kunne benytte seg av GlusterFS. [70]

## 3.4 Fio-tester

For å finne ut hvor stor ytelsen på serverne er, må det benyttes et verktøy som kan gjennomføre ytelsestester. Testene designet for bruk i begge oppsettene er vedlagt i appendix B.6.

Det er viktig for innholdet i rapporten å vite at fio er designet slik at den kan kjøres både mot en block device likeså mot et filsystem. [71]

Den første testen kjøres for å sjekke cache, prossesor og minne til systemet slik at man får en baseline som senere kan sammenlignes mot. Det er valgt å ta med to parametere i denne kommandoen, forklart i appendix C.1.

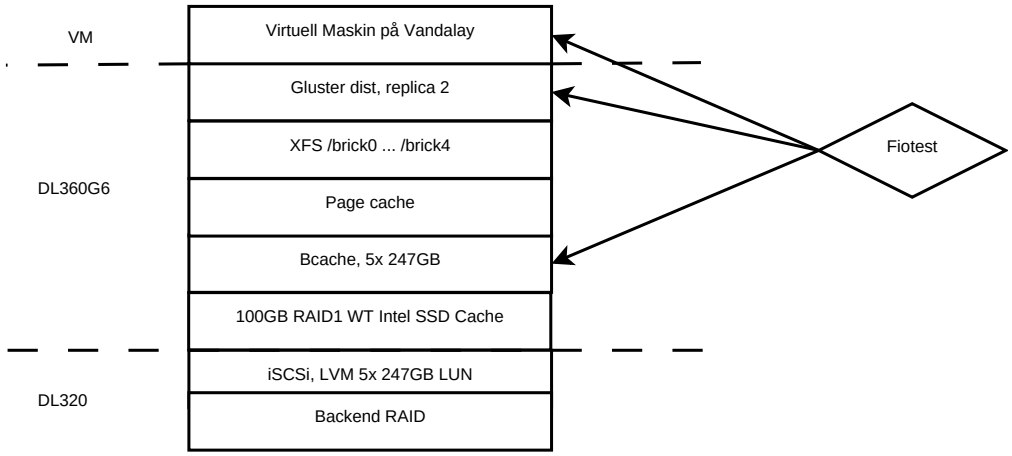
Denne testen skal kjøres på <*block-device*>, med andre ord den devicen som kobler SSD-disken mot backend storage.

De tre etterfølgende testene, nummerert med 2, 3 og 4, er alle hentet fra storagereview. *"Our Enterprise Synthetic Workload Analysis includes different profiles to reflect some real-world tasks. These profiles have been developed to make it easier to compare to our past benchmarks as well as widely-published values such as max 4k read and write speed and 8k 70/30, which is commonly used for enterprise hardware"*. Omformulert vil dette si at ved hjelp av disse testene kan man enkelt sammenligne ytelsen på sitt system med ytelsen andre har oppnådd ved bruk av de samme testene.

I test nummer 2 og nummer 3 er det kun én forskjell, hvor test nummer 2 tester randow writes, mens test nummer 3 tester random reads. Test nummer 4 har en block size på 8KB istedet for 4KB, og det gjøres 70% lesing og 30% skriving.

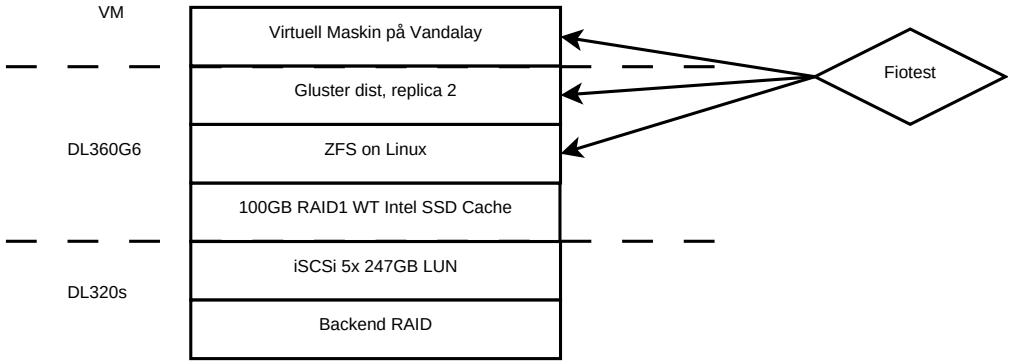
Det er i hovedsak de to siste testene som er viktigst å forklare, respektivt test nummer 5 og 6, ettersom det er disse testene som er satt sammen av gruppen. For forklaring av parameterne se appendix C.2.

Det er ønskelig å kjøre disse testene i tre forskjellige lag, som er vist nedenfor for oppsett 1:




Figur 3.5: Fio-tester oppsett 1

og for oppsett 2:




Figur 3.6: Fio-tester oppsett 2

Testene er planlagt å kjøre i hver av de tre lagene, jfr. figur 3.5 og figur 3.6, hvor det vil bli testet likt i begge oppsettene så langt dette er mulig. Det bør merkes av *hdparm*-testen kun skal kjøres i det første laget, ettersom den kun kan kjøres på block device.

De to testene gruppen har satt sammen har noen varierende parametere, men de samme parameterne går igjen i samtlige tester og vil derfor dekke over parameterne som er med i de tre testene fra storagereview også. Grunnen for valg av parameterne er skrevet nedenfor.

Med tanke på parameteren *–filname* er det valgt å bruke *<block-device>* i det første oppsettet siden det er denne block devicen som er mellomlaget mellom SSD og backend storage. *–filename* vil mest sannsynligvis få en annen verdi i oppsett 2, ettersom oppsettet blir noe annerledes. *–direct=1* og *–ioengine=libaio* henger sammen ved at buffered I/O på linux ikke er asynkron, og vil dermed ikke gi et riktig bilde av ytelsen.

Parameteren *–rw*, i de testene som er satt opp, vil det i nummer 5 bli gjort sekvensiell skriving, og i nummer 6 bli gjort sekvensiell lesing. Som nevnt ovenfor er sekvensiell lesing/skriving sjeldent når man har flere virtuelle maskiner som skriver mot samme SSD, men det er allikvel greit å ha en baseline å sammenligne mot.

*–refill_buffers* er ønskelig å ha med ettersom det å fylle buffer hver gang det gjøres en operasjon, vil gi et mer konkret bilde på systemets kapasitet. *–iodepth=16* og *–numjobs=16* tilsier antall jobber som startes i hver test, samt hvor mange jobber som kan jobbe samtidig mot filene. Det er i dette tillfellet interessant å teste hvordan systemet håndterer flere jobber samtidig, ettersom det er slik systemet må kunne behandle data når det er i et produksjonsmiljø.

Testene skulle være tidsbaserte, derfor er *–runtime=60* tatt med som parameter. Dersom *–runtime* ikke er definert, må *–size* være definert. Det var ønskelig å samle testresultatene for hver testkjøring, og dermed ble det valgt å ta med *–group_reporting*. Til slutt er testene navngitt, for å gi en ryddig oversikt over de ulike testene.

På grunnlag av hardware i arkitekturen, både med tanke på servere og koblingene mellom de, er det noen forventninger som kan stilles til testresultatene. Med tanke på testene som kjøres mot gluster-mount-foler på klientmaskinen, vil ytelsen ikke kunne overstige 100MB/s fordi dette er den maksimale kapasiteten over gigabit ethernet. Videre vet gruppen at en disk, enten det er HDD eller SSD, ikke har mulighet til å oppnå hastigheter på over 1 000MB/s, noe som er mulig i RAM.

Videre, som også forklart i kapittel 2.3, kan ytelsen på enn SSD være rundt 250MB/s for reads og 125MB/s for writes. Det bør her presiseres at dette er uten overhead, eller med andre ord testing direkte på disk, uten noen ovenforliggende software og hardware.

Diskene som er brukt i våre oppsett, hovedsaklig SSD, har en del ovenforliggende systemer i begge oppsettene. Dette vil medøre noe redusert ytelse. Hvor stor denne reduksjonen er, er det ingen i

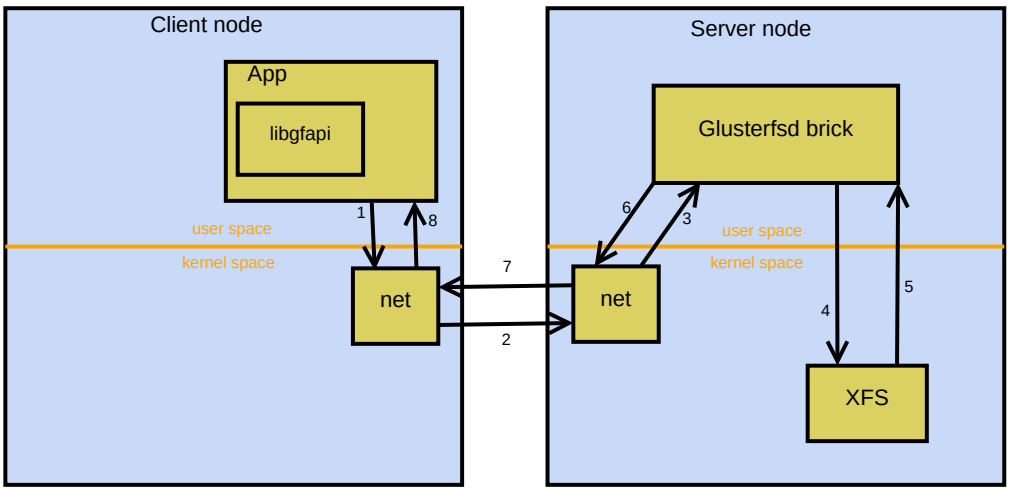
gruppen som kan si med sikkerhet, men det antas at ytelsen på writes kan komme til å ligge på opp mot 50MB/s og reads på opp mot 100MB/s. Disse antakelsene gjelder hovedsaklig testing mot laveste nivå, jfr. figur 3.5 og figur 3.6, henholdsvis mot XFS/bcache i oppsett 1 og ZFS on Linux i oppsett 2.

Når det kommer til ytelsen i de virtuelle maskinene, vil ikke den bli påvirket spesielt av noen ovenforliggende systemer. Grunnen for dette er at de virtuelle maskinene kjører direkte mot hardware på klientmaskinen, og derfor ikke har mye overhead. Ytelsen forventes derfor å være høyere i de virtuelle maskinene enn den er på både klientmaskinen og frontendserverne(XFS/bcache og ZFS on Linux).

## 3.5 Virtuelle maskiner

I løpet av dette prosjektet har det blitt studert to forskjellige måter å opprette virtuelle maskiner på. Her er noen av hovedtrekkene ved de to alternativene.

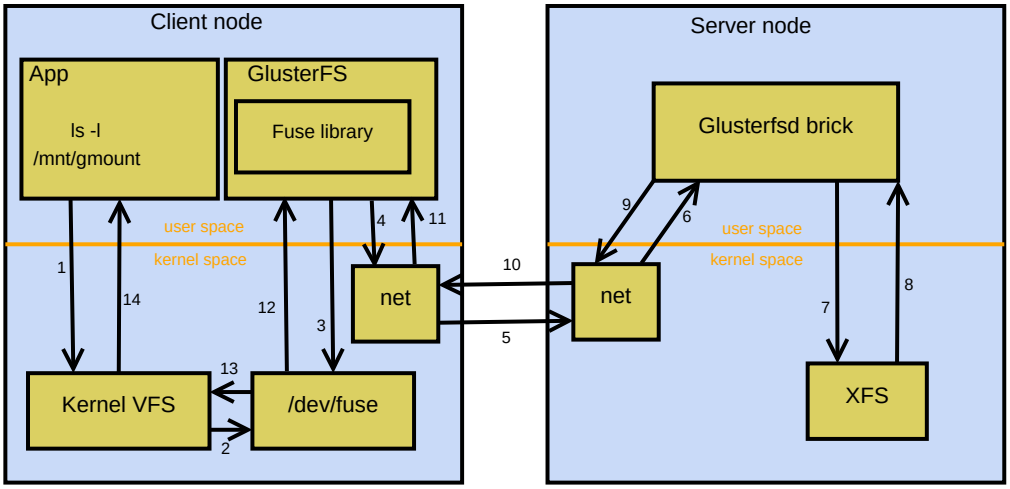
Den første metoden som er studert, bruker libgfapi for å aksessere GlusterFS. Den slipper å gå gjennom en rekke biblioteker, samt en GlusterFS mount, for å kunne kommunisere med GlusterFS-server.




Figur 3.7: Libgfapi

Man kan av dette se at ved bruk av libgfapi, så slipper den virtuelle maskinen å gå via en mount for å kunne kommunisere med GlusterFS-server, og går kun via nettet. Noe av fordelen med dette, er at det skaper en del mindre overhead på trafikken som går mellom den virtuelle maskinen og GlusterFS-server, noe som igjen vil føre til økt ytelse.

Det å bruke FUSE vil føre med seg en rekke oppslag på klient-maskinen, før den kan kommunisere over nettet med GlusterFS-server. Det totale antall trinn den virtuelle maskinen trenger for å nå GlusterFS, vil i libgfapi kun være åtte trinn for hver gang det gjøres en I/O-operasjon, mens ved bruk av FUSE vil den trenge hele fjorten trinn.




Figur 3.8: Fuse

Det synes klart av figurene at libgfapi vil være den ønskelige metoden å bruke ved kjøring av virtuelle maskiner. Disse figurene er hentet fra jp-redhat.com. [72]

# Kapittel 4

# Implementering

## 4.1 Oppsett 1

Herunder vil rapporten ta for seg oppsett 1, men for en mer detaljert og kommando spesefikt overblikk refereres det til appendix D.

**Overordnet sett vil oppsett 1 være slik:**

For den trege lagringsløsningen på backend vil man ha 10 HDD disker satt i et RAID10(1+0) som skal ta imot og lagre data fra frontendserverene. For å kunne utnytte dette på en oversiktlig måte har gruppen implementert LVM, slik at man får tilgang til logiske partisjoner som igjen kan deles ut ved en iSCSI forbindelse mellom backend og frontendserverne.

Den logiske inndelingen, bsrmgrSAN, er sammensatt av flere partisjoner som gruppen henholdvis har kalt *LUN0, LUN1, LUN2, LUN3, LUN4*. Dette ble gjort bevisst for å gjøre det enklere å forbinde de ulike logiske diskene med luns som i etterkant skal settes opp i targetcli, som er en interface for iSCSI targets.

Som nevnt benyttes targetcli som metoden man konfigurer iSCSI targets. De logiske partisjonene ble mappet til ulike luns som skal fordeles til iSCSI initiators. Her satt gruppen opp en iqn for selve lagringsboksen, altså RAID'et, samt at man må legge inn iqn for frontendserveren i targetcli for å tillate tilkobling fra den respektive frontendserveren. Backendserverene kan differensieres ved at iqn ID'en avsluttes med hdbox1:tim eller hdbox1:khomas.

Hver av frontendserverne kobles opp mot sine respektive backendservere ved å benytte iSCSI initiator. Puddy kobles opp mot Tim, og Lippman kobles opp mot Khomas. Dette gjøres ved å benytte open-iscsi, som gjør at frontendserverne kan finne iSCSI targets, samt å koble seg opp mot disse. For at initiator skal kunne kobles mot target må iqn for frontendserveren og den iqn som er listet som godtatt initiator på backendserveren matche hverandre.

I frontendserverne er det blitt implementert en SSD som skal brukes i samsvar med bcache. Dette er for å unngå noen av de ustabile elementene man kan støte på ved å benytte en page cache, samt at man får langt større cache med tanke på lagring. Hvilken type SSD dette er kan man se i kapittel 2.3 HDD vs. SSD.

For å kunne benytte SSD'ens fulle potensiale brukes bcache for å sette opp SSD som cache. Her vil bcache kobles sammen med luns som er tildelt fra iSCSI targets og cache mot disse. Dette gjør at man får 5x 247GB luns som lagring med en 100GB SSD cache.

Man vil benytte seg av denne formen for caching da page cache ikke kan håndtere data som er "in-flight", altså transporteres, på en forsvalig måte. Dettte i den forstand at data som går gjennom RAM er volatile. Det vil da si data som er i RAM forsvinner dersom serveren skrus av eller at strømmen går.

For luns vil man benytte seg av filsystemet XFS som er et journalførings filsystem. Dette gir en bedre mulighet til å gjennopprette data dersom noe kritiskt skulle skje.

For at oppsettet skal komme kjappere opp etter det er tatt ned, vil man mounte de ulike bricks innad i XFS, og kjent som luns når man ser på det i iSCSI sammenheng, med /etc/fstab.

For at man også skal skape redundans mellom frontendserverene vil disse settes i et cluster, som er bygget opp av GlusterFS. GlusterFS vil kjøre replikering og distribuering over begge serverene og dermed vil også bricks som er satt opp via bcache, og senere laget XFS på, bli med i denne prosessen.

Slik at dersom en av frontendserverene skulle gå ned vil ny data som eventuelt blir skrevet til disk ikke gå tapt siden GlusterFS kan fordele dette utover begge serverene med sin heal kommando.

Dette gjør også at all data finnes på begge frontendserverene slik at High Availability konseptet blir innført, nemlig at data er tilgjengelig uavhengig av tilstanden til en av frontendserverene.

Det er så satt opp en server til, som kalles for klientmaskin, eller etter hostnavnet Vandalay. Denne serveren er ansvarlig for å opprette virtuelle maskiner som da kobles opp mot frontend + backend løsningen, samlet sett som lagringsbokser.

Vandalay vil opprette et QEMU image i en mappe som ligger lokalt, men som er knyttet til GlusterFS, noe som gjør at data som lagres i denne mappen blir overført og fordelt utover begge frontendserverene. QEMU image vil operere som lagringsmediumet til én virtuell maskin.

Ved å opprette et slikt image i glustermappen på Vandalay vil man kunne ta i bruk det bakomforliggende oppsettet.

## 4.2 Oppsett 2

Nå som man har fått et innblikk for oppsett 1, kommer nå oppsett 2. For å få en mer detaljert og kommandospesefikt overblikk refereres det til appendix D. Her kan man se ledd for ledd når det kommer til oppbygningen av arkitekturen og oppsettet for ZFS on Linux.

For å få en mer detaljert og kommandospesefikt overblikk refereres det til appendix D. Her kan man se ledd for ledd når det kommer til oppbygningen av arkitekturen og oppsettet for zfs on linux.

Mye av det grunnleggende i begge arktitekturene er likt hverandre, og herunder vil rapporten hovedsaklig peke ut forskjellen mellom de to oppsettene. For mer informasjon om likhetene mellom de to oppsettene, kan det også henvendes til kapittel 4.1.

I oppsett 2 vil det forsatt bli benyttet en iSCSI-forbindelese til å fordele lagring fra backend-serverene til sine respektive frontendservere. Når det gjelder lagringsløsningen på backend, er denne den samme i begge oppsettene. Det vil si at man har satt opp LVM, og videre fordelt ut luns fra backend som en iSCSI target.

Frontendserverene vil forsatt inneholde SSD og denne vil også bli benyttet som en cache device. Forskjellen ligger her i at den skal cache for ZFS poolen i dette oppsettet kontra bcache i forrige oppsett. Når det gjelder frontendserverene er disse satt opp på en annen måte som spriker fra oppsett 1. I dette oppsettet har man en ZFS pool, som konsoliderer lagringen til en felles logisk ordning. Ut fra dette kan poolen splittets eller beholdes som et stort funksjonelt lagringsmedium.

Frontendserverene vil forsatt inneholde SSD og denne vil også bli benyttet som en cache device. Forskjellen ligger i at den skal cache for zfs poolen i dette oppsettet kontra bcache i forrige oppsett. SSD må partisjoneres slik at man kan får både ZIL og L2ARC på SSD devicen. Når det gjelder frontendserverene er disse satt opp på en annen måte som spriker fra oppsett 1. I dette oppsettet har man en zfs pool, som konsoliderer lagringen til en felles logisk ordning. Ut fra dette kan poolen splittets eller beholdes som et stort funksjonelt lagringsmedium.

Innenfor denne poolen ble det opprettet undermapper i ZFS-poolen som hver for seg skal holde styr på de ulike lagringsområdene for klientmaskinen. Disse mappene innad i poolen ble så mounted til lokale mapper som ligger på frontendserveren. Mappene som ligger lokalt på frontendserverene vil

så bli mounted i gluster, slik at man får delt lagring mellom begge frontendserverne.

Klientmaskinen er satt opp på samme måte som i oppsett 1, og vil være ansvarlig for å holde styr på de virtuelle maskinene, samt å opprette qemu-image i glustermappen som er mounted på klientmaskinen.

## 4.3 Virtuelle maskiner

Gruppen har studert to ulike metoder ved kjøring og opprettelse av virtuelle maskiner, jfr. kapittel 3.5. Her vil det bli tatt en nærmere kikk på hvordan man kan sette opp virtuelle maskiner ved bruk av de to forskjellige metodene, samt en forklaring på hva de ulike parameterne betyr og hvorfor det er valgt å ha de med. Viser her tilbake til kapittel 2.3 - Teori, hvor det er skrevet om QEMU.

Siden den første metodikken som ble forklart i kapittel 3.5 var libgfapi, så er det greit å starte også dette kapittelet med libgfapi. Viser her også mot Gantt-diagram vedlagt i kapittel 2.1, nærmere bestemt tidsperioden for oppsett 1, som ble avsluttet den 24.03.2014. Mot slutten av denne perioden kom oppdragsgiver over libgfapi som en alternativ metodikk for å opprette og kjøre virtuelle maskiner, og ønsket derfor at det ble gjort et forsøk på å sette opp og installere en virtuell maskin på denne måten.

Oppdragsgiver hadde sett på et par kommandoer som han mente trengtes for å installere virtuelle maskiner på denne måten, hvor disse kommandoene og kilder ble medsent per mail for å opplyse om denne metoden. Etter videre studering av eksempler, samt noen undersøkelser rundt libgfapi og kommandoene gruppen fikk tilsendt av oppdragsgiver, ble det avgjort at kun disse to kommandoene var nødvendig for å installere virtuelle maskiner.

Først må man opprette et image som skal brukes som lagringsområde for den virtuelle maskinen, etterfulgt av en kommando som skal sette igang installasjonen.

Følgende kommandoer kom gruppen frem til:
- *opprettelse av disk-image:*

```
qemu-img create -f qcow2
gluster://10.0.4.2:24007/bsrmgrSAN/libgfapi-test.img 5G
```

[64, 63]

*- installasjon av virtuell maskin:*

```
qemu-system-x86_64 -enable-kvm -nographic -smp 2 -m 2048
-cdrom ~/iso/ubuntu-server.iso
-drive file=gluster://10.0.4.2:24007/bsrmgrSAN/libgfapi-test.img,
if=virtio,cache=none
```

[62, 63]

For forklaring av parameterne se appendix C.3(qemu-img create, libgfapi) og C.4(qemu-system). Ved utførelse av den første kommandoen, nemlig:

```
qemu-img create -f qcow2
gluster://10.0.4.2:24007/bsrmgrSAN/libgfapi-test.img 5G
```

ble det vist en feilmelding om at gluster var en ukjent protokoll. Ved nærmere undersøkelse av denne feilmeldingen fant gruppen ut at dersom man ønsker å bruke libgfapi som metode, må qemu-kjernen bygges opp med støtte for GlusterFS. Denne prosessen ville innebære en god del arbeid, og grunnet lite tid til rådighet, samt at oppdragsgiver ønsket at det kun ble brukt to arbeidstimer, ble det ikke prøvd å gjennomføre en slik installasjon. [73]

Videre vil det bli beskrevet de kommandoene som ble brukt for å sette opp virtuelle maskiner ved bruk av FUSE og libvirt-bin. For å opprette virtuelle maskiner på denne måten, måtte det til noen flere kommandoer enn ved libgfapi. Her er de kommandoene gruppen har brukt i forhold til de virtuelle maskinene:

*- opprettelse av disk-image:*

```
qemu-img create -f qcow2 disk.qcow2 25G
```

[64]

*-installasjon av virtuell maskin:*

```
virt-install --connect qemu:///system -n ubuntu-server -r 2047
--disk path=/gluster-mount-folder/ disk.qcow2,format=qcow2
-c ~/iso/ubuntu-server.iso --noautoconsole --graphics vnc,
password= foobar,listen=0.0.0.0 --os-type linux
--os-variant ubuntusaucy --accelerate --network=network: default
--vcpus=2 --hvm
```

[69, 68]

*-snapshot av virtuell maskin:*

```
virsh snapshot-create ubuntu-server
```

[74]

*-"skru" den virtuelle maskinen tilbake til tilstanden den hadde når snashot ble tatt:*

```
virsh snapshot-revert --domain ubuntu-server 1395654043 --force
```

[74]

For en kort forklaring av parameterne brukt i disse forklaringene vises det til appendix C.3(qemu-img create, FUSE), C.5(Virt-install) og C.6(Snapshot).

Dette var den metoden gruppen ble enige om sammen med oppdragsgiver, og ville være en god måte for å opprette og administrere virtuelle maskiner. Nå som de forskjellige kommandoene er listet opp, vil det være greit å forklare de parameterne som er brukt i de fire ulike tilfellene.

Når man skal installere virtuelle maskiner kan det være greit å ha et forhåndsallokert lagrings-område ved starten av installasjonen. En av grunnene for dette er at man slipper å definere størrelse under installasjon av den virtuelle maskinen, noe som kan gjøre jobben enklere dersom man skal sette opp automatisert installering.

Ved valg av format er det to ulike valg man kan ta, "raw" eller "qcow2". Qcow2 var det formatet valget falt på for oss, ettersom det vil være dette formatet som ved størst sannsynlighet skal brukes

i et produksjonsmiljø, samt at det skal gi noe økt ytelse.

Til slutt står parameteren som gjelder størrelse på disk-image, som i vårt tilfelle ble satt til å være 25GB stor. Grunnen for at det ble valgt akkurat denne størrelsen var fordi det er stort nok til å kunne få kjørt de testene som var nødvendige å kjøre, samtidig som disk-image ikke ble for stort heller. Det ble gjennomført tester i den virtuelle maskinen med mindre størrelse på disk-image, noe som førte til diskfeil i den virtuelle maskinen på grunn av at all diskplassen ble brukt opp.

Som det tidligere er diskutert er libvirt et grensesnitt for å administrere ulike virtualiseringsplatformer. Tidlig i perioden hvor implementering av oppsett 1 var igang, var det oppdragsgivers ønske at det skulle bli brukt libvirt under administrasjon av virtuelle maskiner. Virt-install er en service som kan brukes for å kommunisere med libvirt, som igjen er et grensesnitt for kommunikasjon med KVM.

Kommandoen som er forklart ovenfor har en rekke parametere som må være slik de er satt for å få en virtuell maskin til å fungere på den ønskede måten. Blant disse er *–connect qemu:///system* og *–disk path*. Når det gjelder de andre parameterne, trenger disse en kort begrunnelse, og det er greit å begynne fra toppen. Som man kan se av kommandoen ovenfor, er minne satt til 2GB ved *-r 2047*. Grunnen for at det står 2047 og ikke 2048, som faktisk er 2GB, er at virt-install ikke støtter sifre høyere enn 2047.

Videre er,*format=qcow2*, som tilsvarer formatet som er brukt i *qemu-img create*, og disse formatene må stemme overens. Parameteren *–graphics* er ikke nødvendig, men er brukt i vårt tilfelle fordi det er ønskelig å bruke en vnc-client, samt å ha passord for console-tilgang.

De to neste parameterne, *–os-type* og *–os-variant*, henger sammen ved at de definerer hva slags OS som skal kjøre i den virtuelle maskinen. *–accelerate* er med i kommandoen fordi det er ønskelig å bruke KVM som virtualiseringsteknikk. Når det kommer til *–network*, er ikke den nødvendig å ha med som parameter ettersom *default* vil bli valgt dersom det ikke er spesifisert noe annet.

Til slutt gjenstår *–vcpus* og *–hvm*. Som beskrevet ovenfor er *–vcpus* den parameteren som sier hvor mange cpu'er den virtuelle maskinen blir tildelt, og *–hvm* er for å sørge for full hardware virtualisering. Begge disse parameterne er valgt i samråd med oppdragsgiver.

Da valget av virtualiseringsteknikk hadde falt på libvirt og KVM, var planen at formatet på disk-image til den ene virtuelle maskinen skulle være "raw" og den andre skulle være "qcow2". Etter videre diskusjon innad i gruppen ble det enighet om at en standard "qcow2", og en snapshot av denne, ville gi mer relevante resultater i forhold til det som var ønskelig.

Først ble det gjort en *qemu-img create* for å opprette disk-image, etterfulgt av en *virt-install* for å installere den virtuelle maskinen. Etter at installasjonen av den virtuelle maskinen var ferdig, ble det avgjort at all software som trengtes for å kjøre tester skulle installeres, samt å opprette de filene som trengtes for å kjøre tester og å lagre unna resultatene. Deretter ble det gjort en snapshot av den virtuelle maskinen, ved hjelp av *virsh snapshot-create ubuntu-server* på klient-maskinen, slik at tilstanden kunne settes tilbake til dette punktet når testene var ferdige. Ved endt testing ble det gjort en *virsh snapshot-revert –domain ubuntu-server 1395654043 –force* for å sette tilstanden tilbake slik den var før testen ble gjennomført.

Poenget med å gjøre snapshot av den virtuelle maskinen var at den første installasjonen ville oppfylle de samme kravene som en virtuell maskin med format "raw", og etter snapshot-revert ville den samme virtuelle maskinen oppfylle de kravene som ble stilt for "qcow2"-format.

## 4.4 Scenario-testing

Det er ønskelig å begynne dette delkapittelet med å forklare litt nærmere hva det innebærer å teste de to scenarioene på de to oppsettene som er valgt å bruke i denne oppgaven. Gruppen vil se over de to scenarioene som skal testes på oppsettene. Hovedsaklig går de to scenarioene ut på hva som vil skje med oppsettet dersom noe går galt.

Det er ønskelig å begynne dette delkapitlet med å forklare litt nærmere hva det innebærer å teste de to scenarioene på de to oppsettene som er valgt å bruke i denne oppgaven. Som nevnt tidligere er serverene satt opp på denn måten.

Her for oppsett 1:

- GlusterFS
- Xfs, frontend
- Bcache, frontend
- LVM, backend
- HWRAID10 (1+0), backend
- 32GB RAM, backend

Det første scenarioet går ut på å se hva som skjer dersom strømmen skulle gå. Et alternativ i forhold til at strømmen kan gå, er å gå til anskaffelse av en UPS. Etter en kort samtale med IT-leder Stian Husemoen ved IT-tjenesten på HiG, fikk gruppen vite at en UPS som kan håndtere strømkravet for hele serverskapet vil komme på 25 000 kroner. Prisen på en UPS kan variere noe, og kan anskaffes for kun ca 15 000 kroner for en UPS med støtte for 10 000W, som vil være på størrelse med det som trengs i OpenStack under produksjon.[75]

I vårt prosjekt kommer det ikke til å bli kjøpt inn en UPS, men det vil bli diskutert om det kan være greit å gå til innkjøp av en, avhengig av hvordan resultatet ved scenario-testene vil være.

Måten det er ønskelig å teste dette på er at serverne "mister" strømmen, ved å ta ut strømtilførselen, noe som er ønskelig at skal skje omtrent samtidig. Grunnen for dette er at ved strømbrudd vil alle serverne miste strømmen samtidig, og det er derfor mest nærliggende å gjøre det på denne måten.

Dette vil da fungere som et simulert strømbrudd, og det vil observeres hva som skjer når serverne blir skrudd på igjen.

Ved gjennomføring av denne testen er det planlagt at det kjøres en kommando i den virtuelle maskinen som skriver informasjon til disk. Den kommandoen som er valgt å bruke til dette er som følgende:

```
for i in {1..200}; do
dd if=/dev/zero of=./fil_$i bs=4k count=10000;
done
```

Det bør her merkes at denne kommandoen er kjørt direkte i et vindu med ssh-tilgang til den virtuelle maskinen. Resultatene av testingen er noe avhengig av metoden brukt for testingen.

Poenget med å kjøre en slik kommando er at man får testet hvor mye av skrivingen gjort mot disk som fortsatt er tilstede når serveren kommer opp igjen etter "strømbruddet". Gruppen kommer til å følge med på det som skjer i den virtuelle maskinen når strømmen blir kuttet, samt at det observeres hva som ble resultatene av testen.

Som et tillegg til dette er det valgt å teste scenario 1 ved hjelp av et skript lokalt på den virtuelle maskinen, for å se om dette gir noe annet resultat i forhold til kommandoen vist ovenfor. Dette skriptet er vedlagt i appendix B.5.

Det er noen ting gruppen regner med at kommer til å skje når serverne skal startes opp igjen etter at testen er gjennomført. Frontendserverene må kobles opp mot backend, for så å mounte backing-devicene slik at de kan bruke /dev/sdb(SSD) som cache-device igjen. Dette kan ordnes ved at man legger inn de gitte kommandoene for disse prosedyrene i oppstartscriptet til serveren. Dersom backendserverene ikke er i kjørende tilstand når tilkoblingen skjer, må den gjøres på nytt etter at backend er oppe igjen.

Videre er det nærliggende å anta at den kjørende virtuelle maskinen kommer til å kræsje på ett eller annet vis. Det skal kun være behov for å kjøre en reset på den virtuelle maskinen, for så å kunne starte den opp igjen, uten at noe i den virtuelle maskinen skal være "ødelagt". Siden transaksjonene

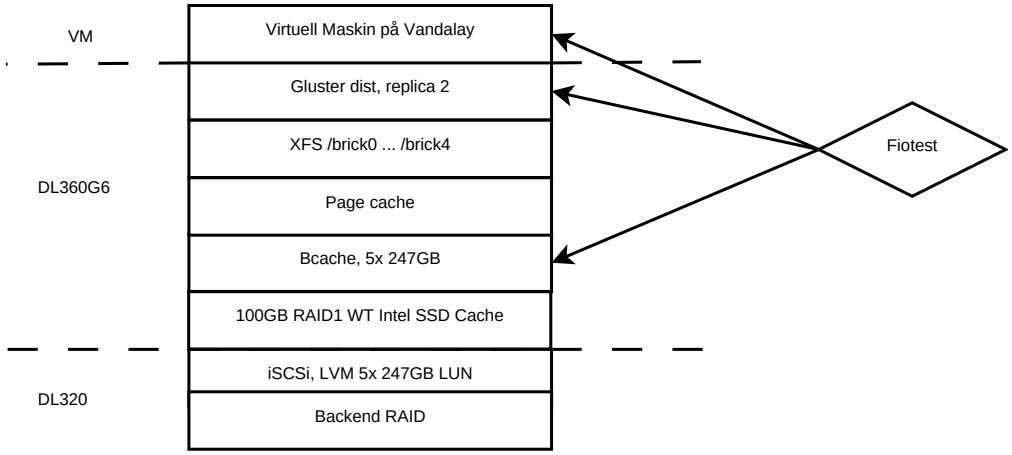
i ubuntu er atomiske, vil data gå tapt dersom den virtuelle maskinen var midt i en transaksjon, så ingen data blir korrupte. Det må presiseres at det er kun data som inngår i den pågående transasksjonen gruppen tror vil gå tapt ved "strømbruddet".

Når det kommer til den andre testen, altså scenario to, vil den ene frontendserveren bli skrudd av en stund. Etter at den ene frontendserveren er skrudd av, vil det bli gjort en god del endringer, som eksempelvis oppretting av filer i mappen som er mounted i GlusterFS. Grunnen for at dette gjøres er at gruppen ønsker å se hva som skjer når frontendserveren blir skrudd på igjen, etter at endringene er gjort.

Det regnes her med at endringene som ble gjort mens den ene frontendserveren var nede, skal ordnes automatisk ved at GlusterFS kopierer over fra den ene serveren til den andre, dersom data'ene på de ikke stemmer overens. Grunnen for at det forventes dette er at gruppen har satt opp replikering, eller med andre ord duplisering av data mellom bricks på serverne, slik at de skal kopiere data mellom seg. Dette er også et grep for å holde på integriteten til data'ene i tilfelle det skjer en feil som fører til kræsj, eller det kommer et strømbrudd.

## 4.5 Fio-testing

Som tidligere nevnt er *hdparm -tT <block-device>* en test som kjøres slik at det lages en baseline, som man kan sammenligne resultater fra de andre testene mot. Det er imidlertid ikke mulig å kjøre denne testen noe annet sted enn mot en block-device. Dette medfører at denne testen ikke kan kjøres på klient-maskinen, ettersom testene her skal kjøres mot et filsystem. Videre er det mulighet for å kjøre denne testen også i de virtuelle maskinene, og vil derfor bli kjørt både for testing mot bcache og i den virtuelle maskinen.




Figur 4.1: Fio-tester oppsett 1

For å kunne måle ytelsen på e mest nøyaktig måte, må man ha flere iterasjoner av testene slikk at man kan kartlegge hvorvidt data man får av testene er der reele målet eller om det er en abnormalitet, og dette medførte til at det ble en enighet om å kjøre hver av testene 50 itererasjoner, ettersom dette er etter best practice samt at 50 iterasjoner vil gi et bedre bilde av hvordan ytelsen til systemet faktisk er. Scriptet som er laget for testing mot bcache er vedlagt i kapittel 4.1.

Planen var å kjøre de samme testene i alle de tre ovenfornevnte lagene, slik at testresultatene ble mer sammenlignbare i forhold ytelsen. Det viste seg tidlig at dette ikke var mulig, ettersom parameteren *–filename=* kun er mulig å bruke dersom man tester direkte mot en block-device, og derfor ikke er mulig å bruke for testing i gluster i figurene ovenfor.

Dette medførte noe mer undersøkelser rundt hvilke parametere som kunne erstatte *–filename=*, og gruppen kom dermed fram til denne parameteren:

- **–directory=<folder>:** spesifikasjon på lokasjon hvor testen skal plassere test-filene, og hvor testene blir kjørt mot.

Som en følge av dette ble *–filename=* byttet ut med *–direcotry=* i fio-kommandoene ovenfor, og testene ble deretter kjørt på klient-maskinen.

Ved starting av testene dukket det opp en rekke feilmeldinger, som alle viste samme tekst. Denne feilmeldingen lyder som følgende:

- *... need to specify size ...*

Det ble lagt til denne parameteren i tillegg, ettersom den som vist ovenfor var nødvendig for testing mot et filsystem:

- **–size=:** den totale størrelse på fil-I/O for den spesifikke jobben. Hver av jobbene lager hver sin fil av denne størrelsen, så lenge filene ikke allerede finnes.

Hvordan disse to nevnte parameterne ble implemetert i fio-scriptet er vist i appendix B.1.2

Før det blir beskrevet noe videre i denne rapporten om testing i de virtuelle maskinene, er det relevant å henvise tilbake til kapittel 4.3 hvor det er skrevet om virtuelle maskiner. Som det er beskrevet i kapittel 4.3, skal det testes i to forskjellige typer virtuelle maskiner. Selv om det skal testes i to forskjellige typer virtuelle maskiner, vil det kun bli referert til den virtuelle maskinen som én, hvor kun det initielle oppsettet er gjort. Oppsettet innebærer innstallasjon av virtuell maskin, opprettelse av test-filer, test-script og snapshot av den virtuelle maskinen, jfr. appendix B.1.3 for mer informasjon.

Som det er nevnt ovenfor, var planen å kjøre de samme testene i alle de tre lagene. Siden det var ønskelig fra gruppens side å teste mot den virtuelle harddisken i den virtuelle maskinen, ble det først kjørt det samme scriptet som ble brukt i testing mot bcache. Disse testene ble derfor startet i den virtuelle maskinen, og kjørte til de skulle vært ferdig. Da testene skulle være ferdig, viste det seg at den virtuelle maskinen hadde kræsjet, uten at gruppen fant noen logisk grunn for at dette skjedde.

Siden det ikke ble funnet noen spesiell grunn for kræsjen, ble det satt opp en ny virtuell maskin, tatt snapshot og kjørt igang nye tester. Denne gangen skjedde det samme med den virtuelle maskinen, og det ble derfor gjort en *virsh snapshot-revert* for å få den virtuelle maskinen tilbake i en kjørende tilstand. Den virtuelle maskinen hadde nå kræsjet to ganger, på relativt like måter og tidspunkter,

som gjorde at det ble kjørt en enkelt test for å se hva som skjedde med den virtuelle maskinen under testen. Etter at denne testen var ferdig, så det ut for at denne ene testen hadde fyllt opp disken til den virtuelle maskinen.

Da den virtuelle maskinen nok en gang hadde fylt opp disken, noe som gruppen antok hendte i begge de to første tilfellene, måtte det settes opp en ny virtuell maskin. Som en følge av diskplassen fio-test-filene brukte, var den første tanken at man kunne legge til *size*= i scriptet slik at den maks produserte denne mengden data. Det ble derfor kjørt igang nye tester, men det samme skjedde også denne gangen.

Det viste seg at dersom man tester mot en block device, vil fio fortsette å skrive utover det som er spesifisert av *size*=, og derfor fylle opp disken allikevel. Gruppen vil her påpeke at det ikke er sikkerhet rundt kommentaren ovenfor, og den er derfor en ren antakelse. Grunnen for dette er at dersom man kjører slike tester, som vist ovenfor, direkte mot en block-device som ikke er designet for testing, kan ødelegge filsystemet på block-devicen. Gruppen fant ingen kilder som kunne underbygge denne påstanden, men gruppen har opplved lavere ytelse på frontendserverene etter det ble kjørt en fio-test mot block-device for OS'et.

Videre ble det avgjort innad i gruppen at testene kunne kjøres på samme måte som det ble gjort i gluster. Valget falt på å plassere *–directory*= i en mappe som ble opprettet i root-directory på den virtuelle maskinen. Som et tillegg til dette ble *hdparm -tT* ikke relevant å kjøre, ettersom det nå ikke ville bli testet direkte mot den virtuelle harddisken.

Grunnen for at gruppen valgte å kjøre testene på denne måten ligger hovedsaklig innenfor de følgende argumentene. For det første, som også for øvrig er nevnt ovenfor, kræsjet den virtuelle maskinen når testene ble gjennomført på samme måte som ved testing mot bcache. For det andre vil denne måten å både skrive til og lese fra filer foregå på en liknende form i et produksjonsmiljø. Det som er ment med dette er at brukerne kun opererer mot filer som opprettes i brukerens hjemmeområde på den virtuelle maskinen.

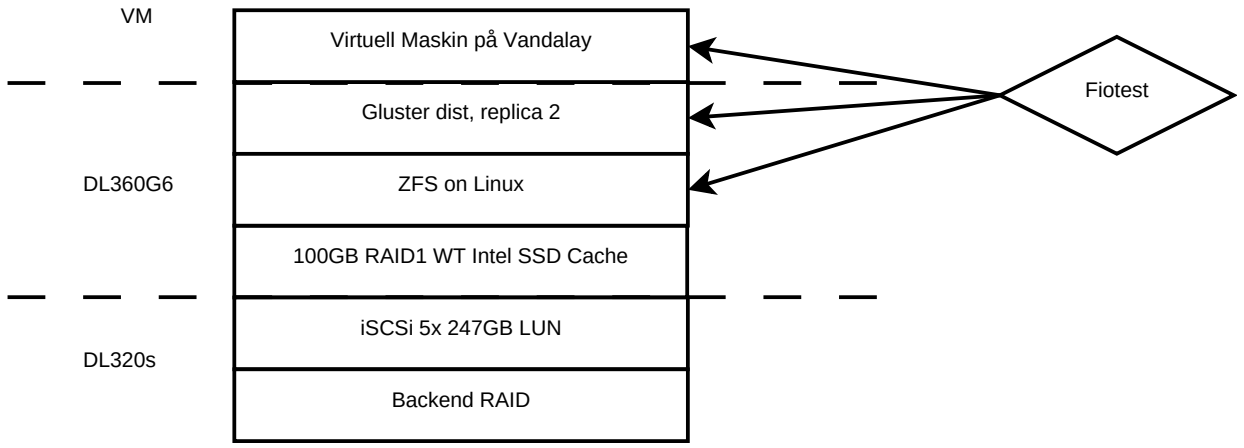
En tredje grunn for at dette vil gi de ønskede resultatene kan underbygges ved hjelp av disse kildene. [9, 50] Av disse kildene kan man se at *–filename* og *–directory* kun er omdirigeringer av fil-lokasjonen som blir brukt ved fio-testingen.

Testene gjort mot oppsett 1, XFS og bcache, er vedlagt i appendix B.1.

Etter endt testing i alle tre lagene, jfr. figur 4.1, viste det seg at gruppen hadde gjort en feil under oppsettet av bcache på frontendserverene. Det som innebar i feilen var at gruppen under sammenkobling mellom backend lagring og SSD-cache, ikke hadde kjørt de riktige kommandoene. Dette gjorde at SSD'en i frontendserverene ikke ble brukt som cache device mot backend lagring, som igjen ga mye lavere ytelse enn det som var ønsket.

Feilen medførte at alle testene måtte kjøres på nytt, noe som var lite ønskelig ettersom tiden begynte å bli knapp. Grunnen for dette var at gruppen i nærmeste fremtid hadde planlagt å begynne med oppsett 2, og hadde derfor lite tid til feil. Det ble derfor tatt et valg i samråd med veileder at antallet iterasjoner i testene måtte reduseres fra 50 til 30 per test. Kjøringen av testene denne gangen gikk heller ikke smertefritt når det gjaldt testing mot bcache. Grunnen for dette var at testene fylte opp cache-devicen, altså SSD'en, med dirty data, noe som igjen dro ned ytelsen en god del.

Siden gruppen skulle implementere to forskjellige oppsett, ble det også derfor to ulike måter å teste med fio på. Her kan man se laginndelingen for oppsett 2.




Figur 4.2: Fio-tester oppsett 2

Som man kan se av figur 4.1 og figur 4.2 har begge disse oppsettene mange av de samme elementene, men med en vesentlig forskjell. Forskjellen mellom de to oppsettene ligger i at ved oppsett 1 er det implementert XFS og bcache, mens ved oppsett 2 er det implementert ZFS on Linux istedet. Når det så kommer til testing med fio er det noen av parameterene som må endres ettersom det er brukt to forskjellige typer programvare i oppsettene.

Først vil det bli forklart litt generelt om endringene som ble gjort på fio-testene ved ytelsestesting av ZFS on Linux etterfulgt av innvirkningene dette hadde på fio-testene i de andre lagene vist i figur 4.2. Det er tre parametere i fio-testene som må endres når man gjør ytelsestesting med fio mot ZFS on Linux, hvor disse er *–direct*, *–fallocate* og *–ioengine*. Grunnen for at akkurat disse parameterene må endres er at ZFS on Linux ikke har støtte for forhåndsallokering vha. *–fallocate*, samt at ZFS on Linux ikke har støtte for asynkron I/O. Som en følge av at asynkron I/O ikke er støttet må *–direct* settes til å være 0 og *–ioengine* settes til sync. [9, 50]

Når gruppen skulle implementere ZFS on Linux sto valget mellom følgende to muligheter, hvor begge mulighetene skal gi et liknende oppsett som ved bruk av XFS og bcache:

- Sette opp fem ZFS mountpoints per frontendserver.
- Dele opp ZFS volum i fem block devices(zvols) og ha eget filsystem på disse.

Gruppen mente at det å dele opp ZFS volum i block devices var mest hensiktsmessig dersom man setter opp RAID ved å bruke ZFS sin mekanisme for software RAID. Denne mekanismen er forøvrig kalt RAIDZ.

Det var ønskelig fra gruppens side å ha oppsettet mest mulig likt ved bruk av både ZFS on Linux og XFS/bcache, noe som avgjorde at valget falt på å sette opp mountpoints fremfor block devices.

Siden valget falt på å bruke mountpoints for å aksessere ZFS volum måtte det endres en parameter til, ved å bytte ut *–filename* med *–directory*, samme som ble gjort på klient-maskinen. For utdypende informasjon om hvordan disse parameterne ble implementert i fio-test-scriptet, både på frontend, klient-maskin og i virtuelle maskine ved ZFS on Linux, er vedlagt i appendix B.2.

# Kapittel 5

# Testing og Resultater

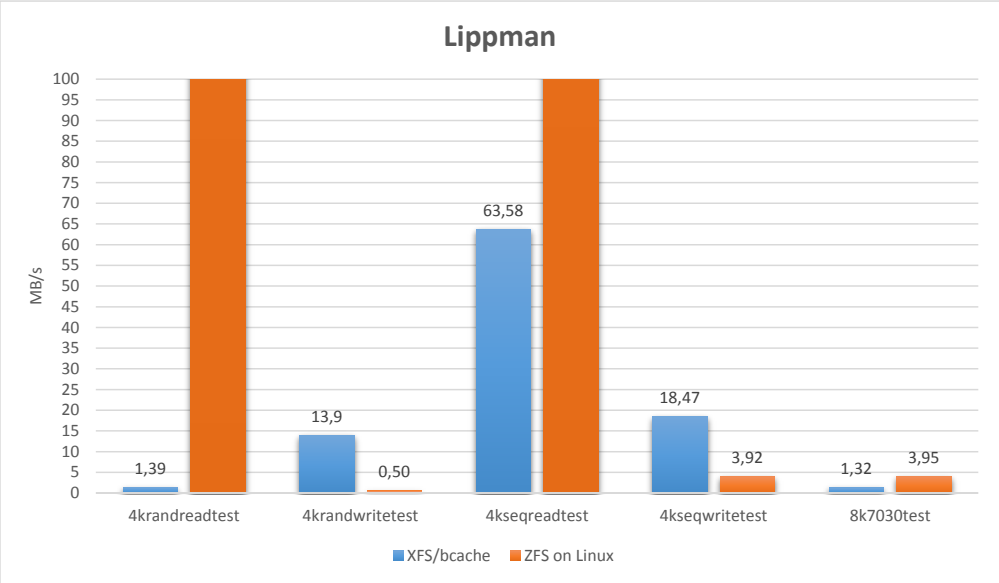
## 5.1 Fiotester

Resultatene gruppen fant ved ytelsestesting i oppsett 1, med XFS og bcache, hadde noen merkelige tendenser. Det viste seg at de testene vi gjorde, spesielt med tanke på skriving, hadde muligheter for å ødelegge filsystemet på block devices. På grunnlag av den rekkefølgen testene ble kjørt, altså fra nederst til øverst(1. XFS/bcache, 2. GlusterFS-mount-folder, 3. VM), fikk dette konsekvenser for alle resultatene. Testene er derfor kjørt på nytt, men i motsatt rekkefølge, samt at det er testet på to måter mot XFS/bcache. Resultatene fra disse testene vil komme i et eget kapittel etter konklusjon. Dette er gjort etter ønske fra oppdragsgiver å gjennomføre nye tester, og å legge resultatene i eget kapittel.

Gruppen har kjørt fio-tester i begge oppsettene for å sjekke ytelsen. Testene har blitt kjørt på frontendserverene, på klientmaskinen og på de virtuelle maskinene. Testene blir oppgitt i MB/s ($^{iops*4}/_{1024}$). Det ble kjørt seks forskjellige tester, hvorav fem er fio-tester og den sjette tester block device, for å se hva slags kjøringer som serveren håndterer bra og dårlig. Testene består av hdparm, random read, random write, sekvensiell read, sekvensiell write og til slutt en sammensatt test med både random read og write. Den sistnevnte testen er 70% read og 30% write. Istedet for at den er 4KB blocksize som de andre, så har denne 8KB blocksize.

Testene vil bli forklart i følgende rekkefølge, først en forklaring av testresultatene fra oppsett 1, XFS og bcache, og så for oppsett 2, ZFS on Linux. Dette går igjen for hver av serverene.

Første testen som vises er fra en frontendserveren Lippman.




Figur 5.1: Fiotest frontend Lippman

| Standardavvik Lippman i MB/s | | |
|---|---|---|
| | Oppsett 1 | Oppsett 2 |
| 4krandreadtest | 0,59 | 56,54 |
| 4krandwritetest | 36,35 | 0,04 |
| 4kseqreadtest | 39,72 | 51,17 |
| 4kseqwritetest | 27,17 | 0,74 |
| 8k7030test | 0,82 | 2,23 |

Når man ser på Lippman i oppsett 1, kommer det frem at reads har en høyere iops enn writes. Dette forekommer fordi det er vanlig at man benytter mindre ressurser når man utfører en read kontra en write. Dette gjelder for alle testene utført på Lippman, foruten 4krandreadtest. Grunnen til at denne er lav, kan ha en sammenheng med skriptet i seg selv, siden på dette tidspunktet er ikke cachen "varmet opp". Dette kan gi lav hit ratio, noe som trekker ned iopsen, siden cachen må foreta flere cache lookups.

Som man kan se fra figur 5.1 har testene et høyt standardavvik. Dette tilsier at svingingene i resultatene som ble observert er høye. Det vil si at det forekommer peaks og lows, og motsier at resultatene er normalfordelt. Skal dette oversettes til hva som faktisk skjer på serveren vil dette si at cachen fylles opp, og dumpes gjentatte ganger utover testforløpet. Dette er også noe som kan gjenspeiles i rå testdata, man ser at iops starter på ca 30 000, før den så dropper ned til ca 300, hvor den så vil ligge en stund, og gradvis bygge seg opp. Etter dette vil iopsen peake igjen på ca. 30 000 før det samme mønsteret gjentas. Dette gjelder hovedsaklig den høyest ytende testen, 4kseqread, men er en trend som går igjen når standardavviket er høyt.

Ulempen med disse svingingene er at de observerte testdata vil være upålitelig i den forstand at man ikke kan si noe definitivt om dem, i ren statistisk sammenheng.

Testene i seg selv ligger på 63,58 MB/s for 4kseqread, rundt 13,90 MB/s for 4krandwrite, og 4kseqwrite ligger litt høyere på 18,47 MB/S. 4krandread ligger på 1,39 MB/s, mens 8k7030 testen ligger på 1,32 MB/s.

I forhold til de resultatene hdparm gav, var det forventet høyere ytelse ved testing med fio, enn det som er vist her. Men igjen vil hdparm kun teste direkte på de ulike diskene, for å kunne gi en liten baseline. Fio vil gå gjennom de ulike lagene i arkitekturen på en annerledes måte og dermed skape mye mer overhead. Dette er med hovedtanke på opprettelsen av fio testfiler på disken som det testes mot.

Hdparm viste en hastighet på ca. 8 500 MB/s for cached reads og 36,7 MB/s for buffered disk reads. Men som nevnt er denne testen for å skape en røfflig baseline og en liten pekepinn på hvordan ytelsen er for hver enkelt disk. Denne testen ble ikke kjørt like mange ganger som fio-testene ble og kan dermed føre med seg noen unøyaktigheter. Senere tester viste en betydelig lavere MB/s verdi. Det vil si at cached reads lå på ca 1 000 MB/s og buffered reads på ca 3-5 MB/s. Men de sier noe om hvor mye data hver disk klarer å prosessere, teoretisk sett.

For å bedre noen av resultatene kunne man utført ulike operasjoner som er med på å legge data i cachen, slik at man har flere hits når man iverksetter testene.

Når man ser på Lippman i oppsett 2, kan man se at begge read testene er veldig høye. Begge to ligger på over 2 000 MB/s, noe som kun er mulig å oppnå fra RAM. Standardavviket på disse

testene er over 50 MB/s, men i forhold til hvor høy ytelse disse testene har, så er dette standardavviket normalt.

4krandwritetest ligger derimot på 0,50 MB/s og er lavere enn forventet. Grunnen til dette kan være mye forskjellig. Det kan være cachen som får mye page faults, og dette fører til at hastigheten blir såpass lav, eller det kan være kjørende programmer som krever en del ressurser. Testen holder seg stabilt lavt under hele kjøringen.

4kseqwrite- og 8k7030-testene ligger på 3,92 og 3,95 MB/s, som også er veldig lavt. Det kan være noe av det samme grunnen som på 4krandwrite, at cachen får mange page faults som får hastigheten til å droppe. Dette viser også at ZFS on Linux er optimaliisert for lesing og ikke skriving. Mønsteret til 4kseqwrite i forhold til iopsen, kan sammenlignes med oppsett 1, ved at den starter høyest på 1 300 iops, hvor den gradvis faller til 700, øker til 1 300 igjen, og deretter gjentas det samme mønsteret. Det kan et liknende mønster på 8k7030test, men her starter iopsen lavt på rundt 350 og 150, og øker utover kjøringen til ca 1 500 og deretter 600, hvor iopsen så synker til rund 300 igjen.

Grunnen til dette er at det kan være mye dirty data i cachen som forårsaker at iopsen dropper mye, men når den får flushet ut den dirty dataen, så vil iopsen øke igjen.

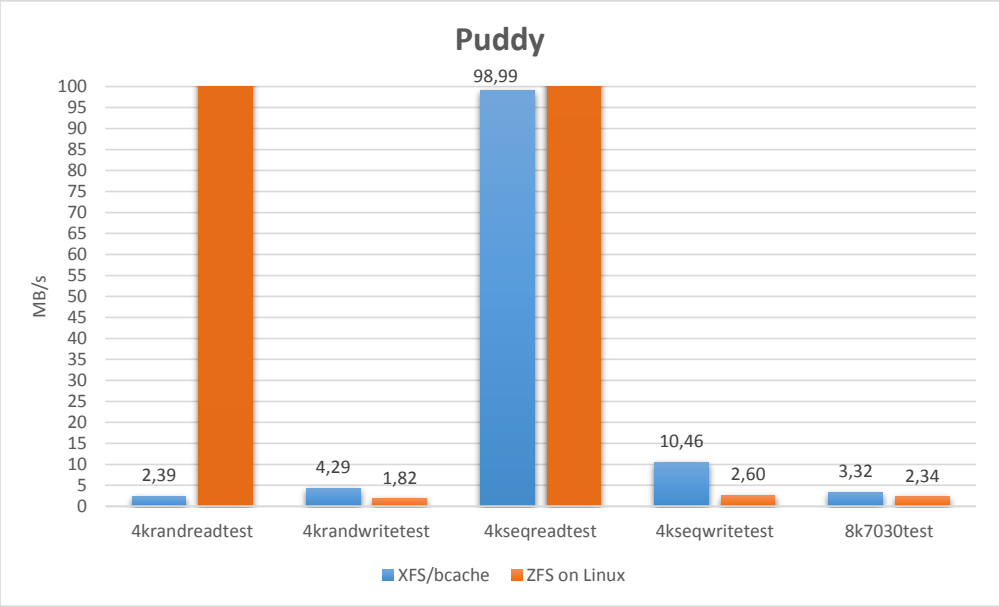
Når man sammenligner testresultatene, så kan man tydelig se at ZFS on Linux er overlegen på lesetestene i forhold til XFS. Begge lesehastighetene til ZFS on Linunx er på over 2000 MB/s, mens XFS sin er ikke på mer enn 1,39 MB/s(random) og 63,58 MB/s(sekvensiell). Man må huske at ZFS on Linux sine testresultater er lesing rett ifra RAM. Så under praksis så kan lese hastigheten til ZFS on Linux være mye lavere.

På 4krandwrite så ligger XFS langt over ZFS on Linux. ZFS on Linux ligger på rundt 0,5 MB/s, mens XFS ligger på 13,9 MB/s. Man kan se at XFS er mer tilpasset skriving enn det ZFS on linux er. På 4kseqwritetest ligger XFS over ZFS on Linux. Her ligger ZFS on Linux på rundt 3,92 MB/s, mens XFS ligger på rundt 18,47 MB/s. Lesetestene tilsier at ZFS on Linux i liten grad er optimalisert for skriving.

Siste testen så ligger ZFS on Linux på 3,95 MB/s, mens XFS ligger på 1,32 MB/s. Begge ligger svært lavt, men det er ZFS on Linux som kommer best ut i denne kjøringen. Dette kommer av at

her vil man ha en kombinasjon av skriving og lesing som blir utført under testforøpet.

Neste test kommer fra frontendserveren Puddy.




Figur 5.2: Fiotest frontend Puddy

| Standardavvik Puddy i MB/s | | |
|---|---|---|
| | Oppsett 1 | Oppsett 2 |
| 4krandreadtest | 0,08 | 72,82 |
| 4krandwritetest | 0,71 | 1,70 |
| 4kseqreadtest | 16,67 | 56,50 |
| 4kseqwritetest | 1,52 | 0,20 |
| 8k7030test | 1,33 | 2,58 |

Puddy har den samme konfigurasjonen som Lippman, som tilsier at resultatene vil være forholdsvis like, og som man se av figur 5.2 er dette til en viss grad sant. Det vil si at det er de samme testene som peaker på Puddy som på Lippman. Den vesentlige forskjellen er at man har 4kseqread som er ca 40 MB/s høyere enn hva Lippman-testene viser.

Når det kommer til standardavviket på denne serveren vil de høye avvikene ha den samme trenden som Lippman har. Det vil si at de peaker, før de så faller igjen. Dette tilsier, som på Lippman, at cachen fylles opp, før den så dumpes. Dette mønsteret forekommer gjennom hele testforløpet.

Hdparm på denne serveren viser de samme resultatene for cached reads, men gir en noe høyere buffered read-hastighet, det vil si 43,82 MB/s på Puddy kontra Lippmans 36,7 MB/s. Men de samme forbeholdene bør bli tatt her som på Lippman ved at hdparm ikke er kjørt så mange ganger som de andre testene.

Fio read testene er noe lavere her enn hva de var på Lippman, noe som kan være et resultat av at hardware er eldre eller har blitt mer brukt gjennom årene, slik at ytelsen er noe redusert. Som nevnt tidligere, er disse serverene lånt av Høgskolen i Gjøvik, og er ikke nye servere. Det kan også nevnes at selv om konfigurasjonen er lik kan testresultater variere noe, endringen i seg selv er ikke stor men betydelig nok til at dette legges merke til.

Det vil si i den forstand at brukeren ikke ville ha merket dette under bruken av oppsettet, men målingene differer nok til at dette bør bli lagt merke til siden det kan være et underliggende problem, og det er en mulighet for at forskjellen kan bli større dersom man oppskalerer dette oppsettet.

Foruten disse forskjellene vil Puddy i sin helhet være identisk med Lippman. Med dette menes at de samme trendene Lippman viser forekommer også her.

Som på oppsett 1 har også Puddy og Lippman lik konfigurasjon ved ZFS on Linux. Det er derfor naturlig at resultatene på Puddy vil være likt som på lippman. Det kan man også se ut fra test-resultatene. Begge read-testene har høy MB/s som er på over 2 000, og leser også her fra RAM.

Man kan også se likheten på 4krandwritetest for begge frontendserverene. På Puddy ligger testen lavt og er ikke på mer enn 2,39 MB/s, som er en forbedring fra å ligge under 1 MB/s i oppsett 1. Standardavviket på Puddy som er på 1,70 MB/s, er mye større enn på lippman sin 0,03 MB/s. Det kommer av at på Puddy så starter iopsen på rundt 2 500, hvor den i løpet av få iterasjoner dropper iopsen til rundt 200, så varierer iopsen imellom 200 og 800 utover kjøringen. Dette kan komme av

at cachen får for mye dirty data, som blir flushet iløpet kjøringen.

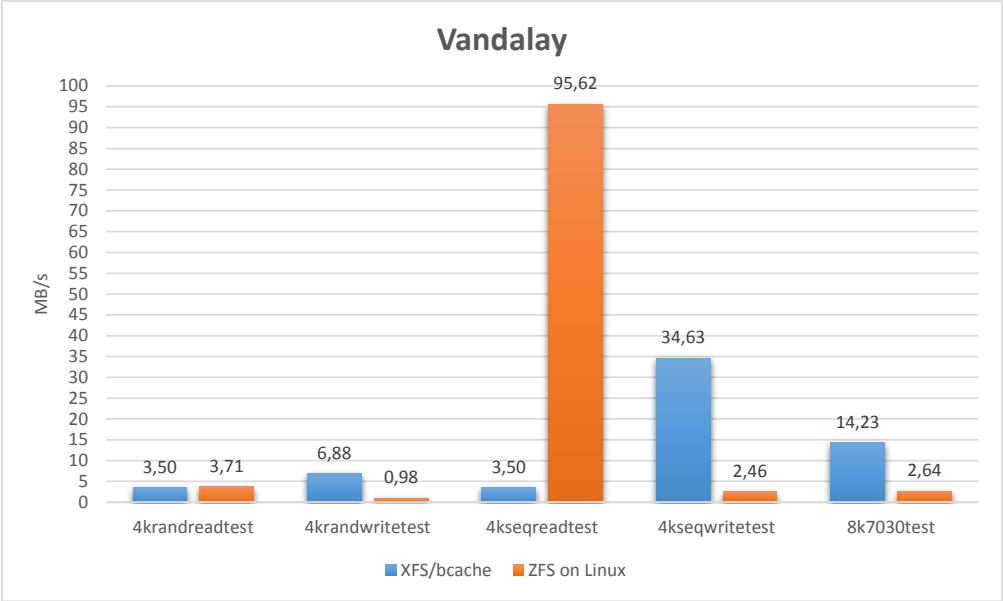
Den neste testen er 4kseqwritetest som i forhold til Lippman sin 3,90 MB/s ligger på 2,60 MB/s. Kjøringene på Puddy er også mye mer stabil og holder seg på rundt 700 iops.

På Puddy er fortsatt tilfellet at lesingen er mye høyere på ZFS on Linux en det det er på XFS. Begge lese testene i ZFS on Linux ligger begge over 2000 MB/s her også. XFS 4krandread ligger på 2,39 MB/s og 4kseqread ligger på 98,99 MB/s.

På begge write-testene ligger XFS over ZFS on Linux. XFS sin 4krandwrite ligger på 4,29 MB/s og 4kseqwrite ligger på 10,46 MB/s. ZFS on Linux sin 4krandwrite ligger på 1,82 MB/s og 4kseqwrite 2,60 MB/s.

Siste testen, 8k7030, er i forhold til resultatene på lippman, faktisk større i oppsett 1 enn det er i oppsett 2. XFS-resultatet ligger på 3,32 MB/s og ZFS on Linux resultatet ligger på 2,34 MB/s.

Neste test er fra serveren vandalay.




Figur 5.3: Fiotest klientmaskin Vandalay

| Standardavvik Vandalay i MB/s | | |
|---|---|---|
| | Oppsett 1 | Oppsett 2 |
| 4krandreadtest | 0,003 | 0,01 |
| 4krandwritetest | 0,64 | 1,44 |
| 4kseqreadtest | 0,005 | 0,57 |
| 4kseqwritetest | 1,07 | 0,69 |
| 8k7030test | 8,07 | 1,79 |

Denne maskinen er koblet opp mot Lippman og Puddy via GlusterFS. På testene kan man se at begge read testene ligger på 3,50 MB/s. Dette er det laveste MB/s på denne serveren. De har også nærmest ingen standardavvik, som tyder på at kjøringene er svært stabile. Grunnen til dette er at Lippman og Puddy ikke klarer å gi fra seg mer en dette. Flaskehalsen er at frontendserver ikke klarer å sende ifra seg mer enn 3,50 MB/s omgangen ifra klientmaskinen.

4krandwritetest er litt høyere enn begge read-testene og ligger på 6,88 MB/s. Teorien vil være at

denne testen burde være lavest av dem alle, siden write er generelt mer krevende enn read, og random er mye mer krevende en sekvensiell kjøring. Siden det lagres direkte over gluster så vil write testen ha bedre ytelse enn read testen. Standardavviket på denne testen ligger på 0,64 MB/s og dette er også svært lavt, som tilsier at testen er svært stabil, altså at testresultatene er gyldige.

Sekvensiell write-test er kjøringen som har høyest MB/s i oppsett 1. Den ligger på 34,63 MB/s, og standardavviket også her er svært lavt. Gruppen forventet at denne skulle være imellom 30 til 40 MB/s, og det vil si testen er sånn cirka som gruppen forventet at den skulle være.

Den siste testen er 8k7030-testen. Den ligger ganske høyt i forhold til at de andre testene siden read testene er såpass lave. Denne testen ligger på 14,23 MB/s, men med høyt standardavvik. Standardavviket er det høyeste blant disse testene, og grunnen til det er at denne testen består av en read og write som nevnt tidligere. Som begge har høy forskjell på ytelse. Dette forårsaker denne høye standardavviket.

Første testen på Vandalay i oppsett 2 som er 4krandread ligger på 3,70 MB/s. Standardavviket på denne testen er under 0,01 MB/s, som er veldig lavt, så testen er veldig stabil. Neste test er 4krandwrite, som ligger rundt 1 MB/s. Denne testen har et høyt standardavvik på 1,40 MB/s. Dette er også et tilfellet av at iopsen ligger svært høyt, rundt 2 700 i de første kjøringene, men dropper svært raskt etter flere iterasjoner av testen. Deretter holder den seg veldig lavt på rundt 100 iops i resten av kjøringen. Dette fører også med seg at fordelingen av observasjonene ikke er normalfordelte, som igjen gjør standardavviket uvirkelig høyt.

Neste kjøring er 4kseqread som er på 95,62 MB/s. Standardavviket på testen ligger på 0,57 MB/s, som er nærmest ingenting i forhold til den høye ytelsen på 95,62 MB/s. 4kseqwrite er i dette tilfellet veldig lav og ligger på rundt 2,45 MB/s, mens standardavviket på denne testen er på 0,68 MB/s. Denne testen burde ligge over 30 MB/s, jfr. kapittel 3.4. Grunnen til dette kan være ZFS on Linux er lite optimalisert for skriving.

Siste testen er 8k7030 og denne ligger på 2,63 MB/s. Standardavviket på denne testen ligger på 1,78 MB/s. Dette store standardavviket skyldes som nevnt at testen inneholder både lesing og skriving. Det kan begrunnes med at skriving og lesing har ulik ytelse, som man kan se av figur 5.3, og derfor skaper det høye standardavviket.

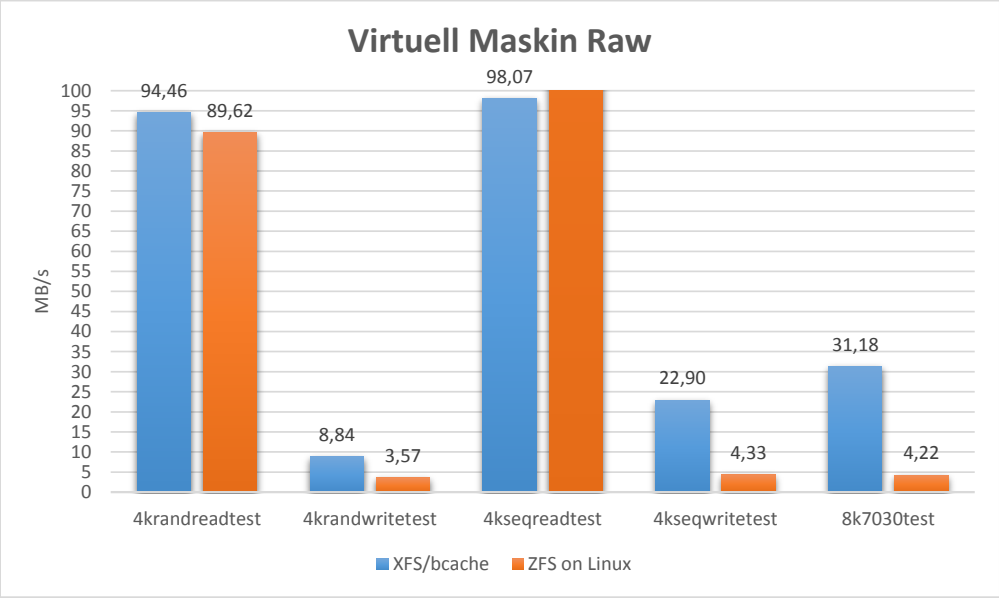
Når vi sammenligner XFS med ZFS on Linux, er det ikke mye ulikheter mellom 4krandread-testene. XFS ligger på 3,50 MB/s, mens ZFS on Linux sin ligger på 3,71 MB/s.

På 4krandwritetest så er det en av oppsettene som kommer merkbart bedre ut. Oppsett 1 ligger på 6,88 Mb/s kontra oppsett 2 som ligger på 0,98 MB/s.

4kseqread så ligger XFS på 3,50 MB/s. ZFS on Linux ligger på 95,62 MB/s så ligger ZFS on Linux bedre an her. Neste test er 4kseqwrite og nok engang kommer XFS sin write-kjøring bedre ut. XFS ligger på 34,63 MB/s, mens ZFS on Linux sin løsning ligger bare på 2,46 MB/s.

Siste kjøring, 8k7030, kommer XFS sin kjøring bedre ut en ZFS on Linux. XFS ligger på 14,23 MB/s, mens ZFS on Linux ligger på 2,64 MB/s.

Den neste testen er fra en RAW VM på klientmaskinen.




Figur 5.4: Fiotest VM RAW

| Standardavvik VM Raw i MB/s | | |
|---|---|---|
| | Oppsett 1 | Oppsett 2 |
| 4krandreadtest | 5,06 | 49,12 |
| 4krandwritetest | 5,35 | 2,20 |
| 4kseqreadtest | 3,70 | 40,72 |
| 4kseqwritetest | 1,10 | 1,62 |
| 8k7030test | 12,78 | 2,58 |

Her tester man VM RAW som er laget på vandalay. På denne så er begge read testene veldig høye. Sekvensiell read er litt større enn rand read, og seq read ligger på 98,07 MB/s og rand read er på 94,46 MB/s. Standardavviket på disse testene ligger på rundt 5 MB/s og dette er som vi forventet. Grunnen til at disse er så høye sammenlignet til testene på Vandalay, kan være at disse kjøringene blir kjørt lokalt på Vandalay.

4krandwrite ligger på 8,84 MB/s, men den er ikke forventet å ligge særlig mye høyere. Standardavviket på denne testen er på ca 6 MB/s og dette er svært høyt i forhold til kjøringen. Dette tyder på at det er mye variasjon i kjøringene. Denne testen starter svært lavt, men øker etter hver kjøring, flater ut og stabiliserer seg.

Sekvensiell skriving ligger på 22,90 MB/s, som er noe lavere en forventet. Denne testen har et lite standardavvik, som tyder på at testingen er svært stabil. 8k7030 ligger så høyt som forventet, som er på 31,18 MB/s. Standardavviket på testen er svært høyt. Det vil si litt over 10 MB/s.

Første testen, 4krandread fra oppsett nr 2 i RAW VM, så ligger denne på 89,62 MB/s. Standardavviket i denne testen er på 49,13 MB/s. Dette er et tilfellet av at testen starter lavt og kjøringene er på rundt 5 000 iops, hvor den øker og øker til den har nådd rundt 40 000 iops. Samme grunn her kan være at cachen blir varmet opp slik at iopsen har muliughet til å øke mer og mer.

Neste test er 4krandwrite som ligger lavt på 3,57 MB/s, mens standardavviket på denne testen ligger på 2,19 MB/s. Denne testen er ikke særlig stabil. Iopsen starter høyt på rundt 4 000, men faller fort nedover til 800, og går litt opp igjen til 1 200, hvor den til slutt ender på 500. Her kommer det frem at den ikke er optimalisert for skrive jobber. Det høye standardavviket kommer trolig ifra cachen flushes, og dermed skaper en del page faults.

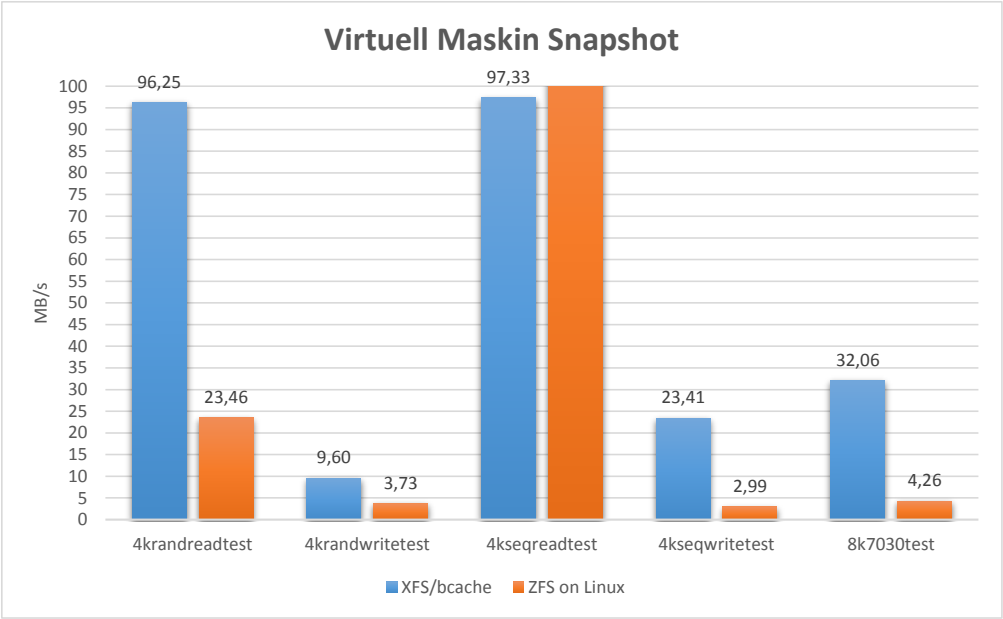
Neste test er 4kseqread som ligger på 500 MB/s. Dette er svært mye, og kan komme av at testen leser rett fra RAM. Standardavviket på denne testen er på 40,72 MB/s. Dette er greit standardavvik i forhold til at testen har såpass høy ytelse. Testen 4kseqwrite ligger på 4,33 MB/s. Dette igjen er svært lavt. Standardavviket på denne testen ligger på 1,61 MB/s, som er ganske høyt når testen ikke er større enn 4,33 MB/s. Iopsen på testen er også veldig ustabil og følger ikke et spesielt mønster. Denne testen er nok slik på grunn av at ZFS on Linux ikke er tilpasset write.

Siste testen er 8k7030 ligger på 4,22. I dette tilfellet er denne testen lavere enn 4kseqwrite. Mens den er høyere 4krandwrite testen. Som vanlig så har denne testen ett høyt standardavvik på 2,57 MB/s, som er en følge av at den har både lesing og skriving.

På den første testen så ligger ZFS on Linux og XFS med ganske så lik i hastighet. XFS ligger på 94,46 MB/s. Mens ZFS on Linux sin ligger på 89,62 MB/s. Neste test som er 4krandwrite så har XFS en hastighet på 8,83 MB/s, og ZFS on Linux har hastighet på 3,58 MB/s. Neste test som er 4kseqread så ligger ZFS on Linux langt over XFS. ZFS on Linux ligger på rundt 500 MB/s, mens XFS ligger på 98,07 MB/s.

4kseqwrite så ligger XFS på 22,9 MB/s og ZFS on Linux ligger på 4,33 MB/s. Siste test er 8k7030 så ligger XFS på 31,18 MB/s, mens ZFS on Linux ligger på 4,22 MB/s. XFS klarer å kjøre denne testen her svært bra. Mens ZFS on Linux virker som den sliter mer med disse kjøringene.

Neste test er snapshot fra VM på klientmaskinen.




Figur 5.5: Fiotest VM Snapshot

| Standardavvik VM Snapshot i MB/s | | |
|---|---|---|
| | Oppsett 1 | Oppsett 2 |
| 4krandreadtest | 5,31 | 4,21 |
| 4krandwritetest | 6,04 | 3,62 |
| 4kseqreadtest | 5,08 | 52,20 |
| 4kseqwritetest | 1,01 | 0,67 |
| 8k7030test | 13,12 | 2,82 |

På oppsett nr 1 så er testene her nesten nøyaktig lik som den forrige VM testen, noe som tyder på at det ikke har noe spesielt å si om du kjører en VM snap eller en VMraw, med tanke på ytelse. Testen som er mest ulik fra RAW, er 4krandreadtest, og de er bare 2 MB/s forskjell ifra hverandre.

Neste testene er kjørt på VM snapshot i oppsett nr 2. Den første testen er 4krandread og ligger på 23,46 MB/s. Dette er en bra MB/s til å være random read. Grunnen til dette kan være at man jobber

lokalt til å read fra VM'en. Standardavviket på testen ligger på 4,21 MB/s, som tilsier at testen ikke er normalfordelt, og testdata er mindre gyldige.

4krandwrite ligger på 3,73 MB/s og er dermed veldig lav. Her kan man se at ZFS on Linux ikke er optimalisert for skrive jobber. Standardavviket på denne testen ligger på 3,61 MB/s og dette kan anses som svært høyt i forhold til at testen ligger på 3,73 MB/s. Testen starter svært høyt på 2500 iops. Hvor den gradvis synker til rundt 300 iops. Tilslutt hvor den peaker opp igjen til rundt 2500. Dette kan skyldes cachen. Hvor den får for mye dirty data før den klarer å flushe vekk, og når cachen først blir flushet, så øker iopsen igjen.

Neste test er 4kseqread som ligger på hele 133,35 MB/s. Dette er mye selv til å være en sekvensiell lesing. Denne testen har ett høyt standardavvik, som vil si at den ikke er stabil eller spesielt gyldig siden dette indikerer på at testen ikke er normalfordelt iht. testobservasjonene.

Den starter på rundt 18 000 iops, men bare øker mer og mer utover kjøringen. Når kjøringen nærmer seg slutten så kan man se at iopsen varierer fra 30 000 til rundt 70 000. Disse høye resultatene kan komme av at cachen blir varmet opp til å gjøre disse kjøringene. Dette vil forklare hvorfor iopsen bare øker og øker.

4kseqwrite ligger på 2,99 Mb/s og er i dette tilfellet også svært lavt. Dette er lavere enn random write, noe som ikke er naturlig ift. teorien som ligger bak sekkvensiell skriving kontra random skriving. Men til gjengjeld så er denne testen mer stabil og standardavviket ligger på 0,67 MB/s.

Siste testen som er 8k7030, ligger på 4,25 MB/s og er større enn begge write testene. Denne testens standardavvik ligger på 2,81 MB/s. Denne testen er som nevnt litt spesiell, fordi den utfører både en skrive og lese jobb i samme test.

Første testen som kommer fram her i VM Snapshot er fra XFS og den ligger på 96,25 MB/s. Mens ZFS on Linux ligger på rundt 23,46 MB/s. Det faktum at ZFS on Linux ligger så lavt her, mens den ligger 89,82 MB/s på VM RAW, er overraskende. De resterende testene har veldig likt resultat som på VM RAW. Neste test av XFS ligger på 9,60 MB/s, mens ZFS on Linux ligger på 3,73 MB/s. 4kseqread så ligger XFS på 97,33 MB/s og ZFS on Linux ligger på ca 130 MB/s.

Den neste testen er 4kseqwrite hvor XFS har en hastighet på 23,41 MB/s og ZFS on Linux har en hastighet på 2,99 MB/s. Den resterende testen som er 8k7030 så ligger XFS på 32,06 MB/s, mens ZFS on Linux har en hastighet på 4,26 MB/s.

## 5.2 Scenariotester

**Scenario 1**

Som man kan se av kapittel 4.4, skulle det testes to ulike scenarioer på oppsettene som ble satt opp, hvor det første var å fjerne strømmen fra serverne. Det som ble gjort var at gruppen lagde et skript som opprettet filer, og skriptet ble kjørt i den virtuelle maskinen, dette er vedlagt i appendix B.5. Når skriptet kjørte ble strømmen fjernet for å se hvilken tilstand lagringsmediumene ville være i, og om det var forekommet noen form for korrupsjon av data som var i lagringsoppsettet. Gruppen observerte skriptets kjøring frem til strømmen ble fjernet, slik at man lettere kunne sammenligne resultatene mot det som ble vist på skjermen før kræsjen oppstod.

Dette resulterte i at tilstanden til den virtuelle maskinen var forsatt operativ, i tillegg til at data som skulle blitt skrevet var rullet tilbake og ikke skrevet til disk. Diskene i seg selv var intakte, og det var ikke forekommet noen form for korrupsjon av data. Disk-imaget som ble brukt av den virtuelle maskinen var også operativt, og det eneste som trengtes var å bare starte den virtuelle maskinen på ny.

Noe av det som kan være et problem med oppsettet var at data som var "in-flight" blir rullet tilbake, slik at dersom man mister strømmen må man utføre den forrige operasjonen på nytt. Dvs. at dersom du jobber med noe som ikke automatisk mellomlagrer data for gjennoppretting, må man utføre det man gjorde en gang til. Dette kan skape problemer dersom man utfører store jobber og strømmen ryker.

Ved kjøring av denne testen ble det noen forskjeller på de to oppsettene gruppen har satt opp, men også noen likheter. Samlet for begge oppsettene måtte gruppen koble frontend maskinene til backend maskinene, slik at volumene kan mountes via /etc/fstab, brukt i oppsett 1, og via zfs sin innebygde mount-funksjon, brukt i oppsett 2. Det var etter dette forskjellen mellom de to oppsettene kom, ved at i oppsett 1 ble backing devices mounted rett i gluster, mens i oppsett 2 var det nødvendig å vente en liten stund før man kunne fortsette med mounting.

Noe av grunnen for at denne forskjellen er tilstede, mener gruppen skyldes det høye ressurskravet zfs on linux har i forhold til xfs og bcache. Etter at backing devicene var mounted i GlusterFS, kunne gruppen mounte klientmaskinen i clusteret, og skru den virtuelle maskinen på igjen.
Kommandoene kjørt for å få opp igjen systemet i begge oppsettene er som følgende:

**Oppsett 1:**

```
./scrict/iscsiadm.sh
```

(koble til backend lagring, frontend).

```
mount -a
```

(blkid for bcache backing devices var uncommented i /etc/fstab, frontend).

```
mount -a
```

(tilkobling mot GlusterFS i /etc/fstab, klientmaskin).

```
virsh start ubuntu-server
```

(starte den virtuelle maskinen, klientmaskin).

**Oppsett 2:**

```
/scrict/iscsiadm.sh
```

.(koble til backend lagring, frontend).

```
zfs mount -a
```

(mounting av alle zfs pool/brick's opprettet, frontend).

```
mount -a
```

(tilkobling mot GlusterFS i /etc/fstab, klientmaskin).

```
virsh start ubuntu-server
```

(starte den virtuelle maskinen, klientmaskin).

De dataene som ble liggende på den virtuelle maskinen etter den var kommet opp igjen etter strømbruddet, differerte noe med det som stod i putty-vinduet da strømmen ble nappet. Det viste seg av de fem testene av scenario 1 som ble gjort mot oppsett 2 at den virtuelle maskinen rullet tilbake mellom 50 og 70 iterasjoner/skrivinger vist i skriptet, se appendix B.5.

Det er vesentlig at når denne lagringsboksen blir tatt i bruk, skal det være enkelt å få opp de virtuelle maskinene, og dataområdet til de ulike virtuelle maskinene skal være operativt, med andre ord ikke korrupt.

Resultatene våre viste at dette var tilfellet. Det eneste som trengtes å gjøre med den virtuelle maskinens, var å skru den på igjen. Den koblet opp mot sitt eget disk image, eller lagringsområde, og var operativ fra samme tidpunktet som før den siste transaksjonen mellom virtuell maskin og lagringsdisk ble gjennomført.

Dette medfører at den virtuelle maskinen ikke ble utsatt for noen skade foruten den transaskjonen som foregikk på det tidspunktet strømmen gikk.

Disk image til de virtuelle maskinen fungerte på samme måte slik som det gjorde før testen. Dette er med på å fremme integriteten på lagringsområdet.

For at dette skal ikke skal bli et problem, kan det være et alternativ å se på UPS alternativer som verner mot strømbruddet. Men dette er noe man kan ta hensyn til dersom man skal utføre kun store operasjoner. Dersom det er små operasjoner som kan lett replikeres, og kjøres på ny vil dette ikke være et problem, i og med at man ikke får korrupt data når strømmen går.

**Scenario 2**

Det andre av de to scenarioene beskrevet i kapittel 4.4, vil koblingen mellom de to frontendserverne fjernes. Det vil da si at Lippman, som er slave-maskin i vårt oppsett, ble fjernet fra GlusterFS i nettverket. Hensikten med dette var å se hva som ville skje med lagringen innad i GlusterFS dersom man fjernet en server. Det som teoretisk var tenkt var at Gluster ville detektere at serveren gikk ned, og når den kom tilbake ville GlusterFS automatisk kjøre en “heal” på glusteret slik at begge serverene ville ha data.

Det som ble gjort var at gruppen satte igang det samme skriptet som ble kjørt under testing av scenario 1, vedlagt i appendix B.5. Det som var tanken var at dersom man har et skript som skrive data til en fil, eller flere filer, vil man få de samme filene fordelt utover, eller replikert, utover den andre serveren som er en GlusterFS node.

Tilkoblingen til Lippman ble fjernet ved at serveren ble skrudd av, og skriptet ble kjørt for å opprette filer.

Etter at man koblet opp den andre serveren igjen, slik at den ble en del av GlusterFS i nettverket, sjekket vi tilstanden umiddelbart og registrerte at detmanglet filer. Dette var det gruppen i utgangspunktet regnet med, siden GlusterFS ikke hadde rukket å iverksette noen tiltak som skal være med på å få tilbake den ønskelige tilstanden, det vil si data fordelt utover begge serverene.

Etter 10 minutter ble det sjekket igjen om GlusterFS hadde klart å sette igang operasjoner for å kunne få tilstanden tilbake til "normalen", det vil si at data var replikert, var det forsatt ikke satt igang noen aksjon for å kunne reparere dette. Det ble da kjørt en kommando som skulle ordne opp i dette "problemet", hvor denne kommandoen er vist her:

```
gluster volume heal bsrmgrSAN full
```

Som en følge av kommandoen ble all data replikert mellom de to frontend maskinene i oppsettet vårt, og tilstanden var slik den skal være.

Ved testing av dette scenarioet i oppsett to fikk gruppen et noe annerledes resultat, og som stemte mer overens med det som ble antatt i kapittel 4.4. Det som var annerledes denne gangen var at etter oppstart av den ene frontend maskinen, den som hadde vært avskrudd under testen, klarte å gjøre en automatisk heal på volumet slik at all data ble replikert. Hovedsaklig er det slik det skal skje dersom en GlusterFS brick som er offline kommer online igjen. [76]

# Kapittel 6

# Konklusjon

## 6.1 Evaluering av Arbeid

**Orgnaisering**

Gruppen har gjennom hele bacheloroppgaven hatt tilfredsstillende kontakt med både veileder og oppdragsgiver. Dette har gjort fremgangen i prosjektet noe lettere, samt at det har hjulpet til med å forstå hva som forventes, gjort at planen har blitt fulgt, og gruppen har kommet frem til et tilfredsstillende resultat med rapporten.

**Gruppearbeid**

Det har gjennom denne perioden både vært jobbet sammen som gruppe, men også hver for seg. Begge formene for arbeid har fungert på en slik måte at de arbeidsoppgaver som er fordelt, er gjennomført av det bestemte gruppemedlemmet, samt at arbeidet holder forventet nivå.

Gruppen har tidligere samarbeidet sammen, noe som har fungert bra tidligere, og i løpet av dette prosjektet. Prosjektrollene satt opp i starten av perioden, har vist seg å være et godt valg, og rollene har hjulpet til at organiseringen av rapporten har blitt noe enklere.

**Hva kunne vært gjort bedre**

I starten av prosjektet ble det brukt en del tid på utvikling av webside, kanskje mer enn det var nødvendig, uten noen klar form for forventninger rundt resultatet av dette. Den tiden som ble brukt på utviklingen av webside kunne med fordel vært brukt på bedre formål, som for eksempel undersøkelser rundt oppsettene, spesielt med tanke på zfs on linux.

Med tanke på zfs on linux, som nevnt over, burde dette vært undersøkt en del mer før det ble implementert noe, siden dette ville spart gruppen for en del tid. Et problem som det ble støtt på underveis var at zfs on linux krevde et 64-bit system for å kunne kjøre, og funnet ut av dette før implementeringen startet ville spart gruppen for tid. Som en følge av at det ble benyttet 64-bit system på frontendserverne i oppsett 2, burde dette også vært testet i oppsett 1, for å se om dette kunne gi noen andre resultater.

Videre har det vært en del usikkerhet rundt testing med fio ettersom ingen av gruppemedlemmene hadde noen særlig kunnskap til denne programvaren. Før det ble begynt testing mot zfs on linux, ble testene dermed omorganisert. Det kunne derfor vært greit å kjøre disse testene også mot oppsett 1.

Med tanke på arbeidet gjennomført, generelt sett, kunne dette vært fordelt noe annerledes. Når det gjelder konfigurasjon av servere, burde dette arbeidet blitt fordelt mer mellom alle gruppedeltakerne. Et annet element som burde vært endret med tanke på arbeidsfordeling, var at mot slutten av prosjektarbeidet, ble en person satt til å skrive kapittel 5 alene. Dette burde vært gjort i samarbeid mellom minst to gruppemedlemmer, noe som ville ha gjort arbeidet noe hurtigere, og det hadde blitt høyere kvalitet ved første gjennomgang.

**Mulig videre arbeid**

Emner som kan jobbes videre med rundt de to oppsettene som er satt opp i løpet av denne bacheloroppgaven. Som en start kunne det vært greit å undersøke hvordan man kunne utvidet begge oppsettene horisontalt, altså ved å implementere flere lagringsbokser bestående av hver sin frontend- og backendserver.

Videre kunne det vært gjort noen endringer i oppsettene, ettersom det kan gjøres en del "tweaks" i begge oppsettene. Når det gjelder oppsett 1, vil hovedsaklig tweaks gjennomføres i forhold til GlusterFS og dens parametere, men også noen få ting rundt bcache. Endringer som kan gjøres i bcache vil være størrelse på cache, cache-metode(writeback/writethrough/writearound) og inndeling av cache, hver sin del for hver sin backing device.

Endringene i oppsett 2 vil være noe mer utfyllende siden zfs on linux har en del parametere, både for zpool mountpoints og zfs logical volumes. Endringene som kan gjøres i forhold til GlusterFS vil være de samme i begge oppsettene. Et eksempel av parametere ved zfs on linux som kan endres er; compression, deduplication, recordsize, med mer.

Siden det ikke ble tid i løpet av vår bacheloroppgave til å sette opp igjen oppsett 1, for å teste på nytt med de omdefinerte fio-testene, kan dette være et greit prosjektarbeid i for eksempel faget System Administration med Erik Hjelmås. Tidsperspektivet bør ikke være spesielt langt, siden fremgangsmåten for oppsettet allerede er kjent, og det kun gjenstår testing på nytt. Dette vil gi studentene en god bakgrunn med tanke på oppsett av servere, med tanke på arkitektur og basic konfigurasjon.

Det ble tidlig i prosjektet, i overgangen mellom februar og mars, prøvd å bytte ut RAID controller på backendserverne med en annen type RAID controller. Poenget med dette er å gi støtte for større

HDD i backendserverne, slik at det blir mer lagringsområde. En videre oppgave kan innebære å finne alternativer til RAID controller i backendserverne, siden den RAID controlleren gruppen prøvde å bytte, ikke passet i backendserveren.

## 6.2 Konklusjon

**Ytelsesbasert**

Ved studering av de oppnådde resultatene, jfr. figurer kapittel 6, ser man at det ofte er lite variasjon mellom de to oppsettene, men at det som regel er ZFS on Linux som kommer dårligst ut, foruten to tester, hvor begge tester read-ytelse. Grunnen for dette ligger i at ZFS on Linux er bygget for å fremme ytelsen ved read, og dermed få noe svakere write-hastighet. Dette kommer frem av dokumentasjonen rundt ZFS on Linux, og ZFS generelt.

Det også nevnes her at ZFS-testene som ble utført i henhold til read-estene er cache resultater, det vil da si at den verdien, MB/s, er veldig høy kontra om resultatene ble utført på en disk.

Det ble i etterkant testet med større testfiler for å undersøke om man kunne få et mer reellt resultat. Det som ble gjort for read testene, var at størrelsen ble satt først til 24GB, deretter 40GB. Dette resulterte i at 4krandread viste en ytelse på 770 MB/s og 4kseqread viste en ytelse på 850 MB/s.

Disse ytelsesmålingene var like for begge størrelsene. Men noe som man også bør ta hensyn til er at under 4krandread testen, er dette random og kan medføre flere cache hits/miss. Dette vil igjen resultere til at ytelsen går opp/ned.

Men det bør også merkes at disse endringene ble ikke kjørt like mange ganger som den opprinnelige testen, og kan medføre noen skjevheter. Det som også skjer er at zfs ikke har støtte for direct= parameteren, noe som igjen fører til at det er cache resultater som kommer frem. Dette er ifølge Jenx Axboe en zfs limitasjon fremfor en fio limitasjon.

*"Also the direct=0 is for the fact that in the documentation to fio the direct= parameter is not supported for zfs. And as a result the test will not complete(it does not even create the test files).*

*Clarification, this is not a fio limitation, it's zfs that doesn't support unbuffered IO."*
– Jens Axboe

Dette kommer frem fra en epost samtale med han.

**Scenariobasert**

Med tanke på de to scenarioene som er testet på oppsettene, er det ingen spesiell preferanse ut ifra de resultatene gruppen fant, for hvilket av oppsettene som er det beste. Begge oppsettene har oppført seg som forventet, hvor de begge håndterte scenarioene på en tilfredsstillende måte. Dette var forøvrig forventet, på grunnlag av tidligere kunnskaper. For valg av oppsett, eller nærmere bestemt cachingsmekanisme(XFS/bcache eller ZFS on Linux), vil ikke scenarioene ha stor betydning for utfallet.

**Andre aspekter**

Når man ser på aspektene rundt begge oppsettene, altså hva som må til for å sette opp de to oppsettene, jfr. appendix D.2, vil gruppen helt klart anbefale at det brukes zfs on linux. En av grunnene for dette er at zfs on linux har mindre komponenter man må forholde seg til samtidig, samt at det har vært mindre problemer ved implementasjon av zfs on linux fremfor xfs og bcache.

**Valg av oppsett**

Valg av oppsett vil hovedsaklig bli fattet på grunnlag av ytelsen oppnådd ved testing med fio, siden begge oppsettene håndterer kræsj på tilnærmet samme måte. De to oppsettene har hver sine styrker og svakheter, og en gjenganger er at styrkene til det ene oppsettet er svakheter i det andre oppsettet, og vice versa.

Det at begge oppsettene er såpass likestilte når det kommer til ytelse, sterke og svake sider, vil ikke gruppen fatte noen konkret beslutning på hvilket av de som er best. I løpet av prosjektperioden har det blitt undersøkt en del rundt ZFS on Linux, og hvordan man alternativt kan sette opp denne programvaren. Et alternativ til å velge enten ZFS on Linux eller XFS/bcache, kan være å kombinere XFS og ZFS on Linux, ved å bruke XFS som filsystem på ZFS on Linux sine logiske volum(ZVol). Dette kan være en fremtidig oppgave, med en rekke variabler som kan endres.

Valget av oppsettet vil derfor falle på hvilke egenskaper du trenger, dersom du har et oppsett som vil benytte seg hovedsaklig av reads fremfor writes, vil ZFS on Linux være det beste alternativet. Dersom det benyttes et oppsett som er mer basert på writes, vil det være mer naturlig å velge XFS. På denne måten må den som setter opp en lik eller tilnærmet lik arkitektur, ha kunnskap om hvilke operasjoner som skal gå gjennom oppsettet, før et valg om oppsett kan tas.

# Kapittel 7

# Fiotesting med signifikante endringer

I dette kapittelet vil det gjennomgås resultater som er forskjellige fra den opprinnelige XFS / ZFS on Linux sammenligningen. Gruppen bestemte seg for å gå tilbake til oppsett 1 og utføre endringer.

Dette var hovedsaklig fordi i etterkant ble det vist, som nevnt, at når det testes direkte på en block device vil dette muligens ødelegge filsystemet på den gitte devicen. Det i den forstand at superblockene til det gitte filsystemet og/eller i-nodene kan bli ødelagt siden disse skrives over av testen og dens testfiler.

Siden testingen fra oppsett 1 vil starte på det laveste nivå vil fundamentet i oppsettet være utsatt for skade, og kan gi feil resultater.

Under er en tabell over standardavviket resultatene produserte:

| Standardavvik | 4krandread | 4krandwrite | 4kseqread | 4kseqwrite | 8k7030 |
|---|---|---|---|---|---|
| Lippman(mappe) | 50,50 | 0,37 | 7,90 | 0,12 | 3,84 |
| Lippman(block device) | 0,85 | 0,37 | 0,65 | 0,52 | 1,64 |
| Puddy(mappe) | 49,97 | 0,23 | 5,52 | 0,11 | 3,88 |
| Puddy(block device) | 0,73 | 0,30 | 1,14 | 0,29 | 1,70 |
| Vandalay | 0,004 | 0,003 | 1,38 | 0,006 | 0,32 |

Dette vil bli brukt som en overordnet tabell som viser påliteligheten og gyldigheten til observert data, statistisk sett.

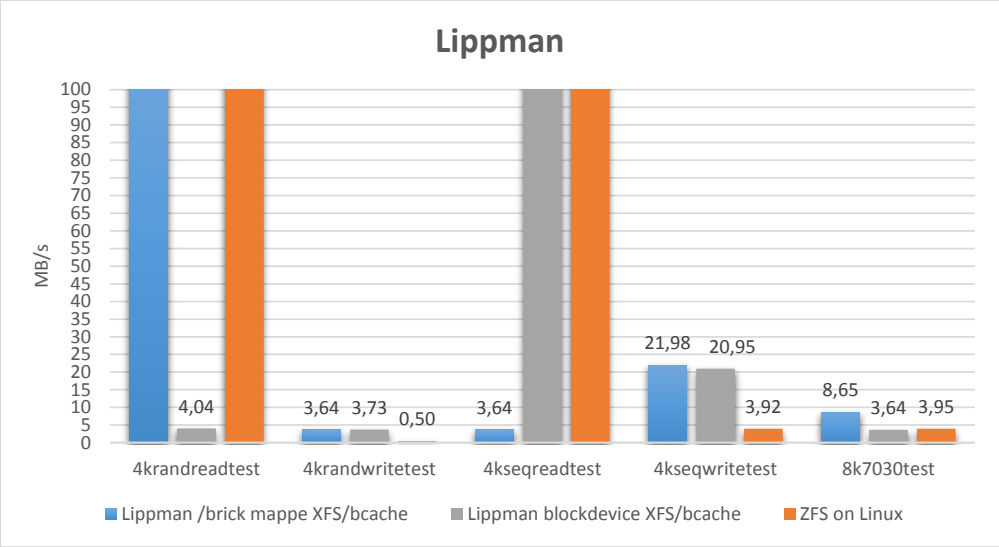
Men forklaringer rundt hvorfor disse standardavvikene er som de er, vil være de samme som i kap. 5.1

Dette tilsier også at dette kapittelet vil peke ut på endringer som har forekommet, og som gruppen mener er signifikant, i den forstand verdt å legge merke til.

Det første som bør påpekes er at testene ble utført fra toppen og ned, og ikke motsatt. Dette resulterer at filsystemet som ligger i bunn ikke vil bli ødelagt i prosessen. Noe som er med på å fremme ytelsen for testene som blir utført. Denne hypotesen blir også bevist gjennom resultatene som blir produsert gjennom hele testforløpet. Dette bli gjennomgått på samme måte som kap. 5.1 og serverene vil komme i samme rekkefølge for ordens skyld.

Det andre som bør merkes er at her har man tatt forbeholdet som ble nevnt tidligere, nemlig at man "varmet opp" cachen. Dette vil da teoretisk sett gi en høyere ytelse siden det finnes data allerede i cachen før testene kjøres.

Første test er Lippman:




Figur 7.1: Fiotest Lippman Ny

Det som er kommet som en addisjon til disse testene er at det også har blitt utført testing på både /brick mappene som bcache og dens bakomforliggede lagringsløsning er mountet til. Samt resultater fra block devicene, det vil si /dev/bcache.

Det første som merkes er at resultatene viser at Lippman når man tester i /brick mappene ligger på 116.96 MB/s. Og block devicen ligger på 4,04 MB/s. Det som bør legges merke til angående den første testen er at block device ytelsen har gått opp. Dette vil da si at flere cache hits ble gjennomført når det ble testet mot dette kontra testen fra kapittel 5.1.

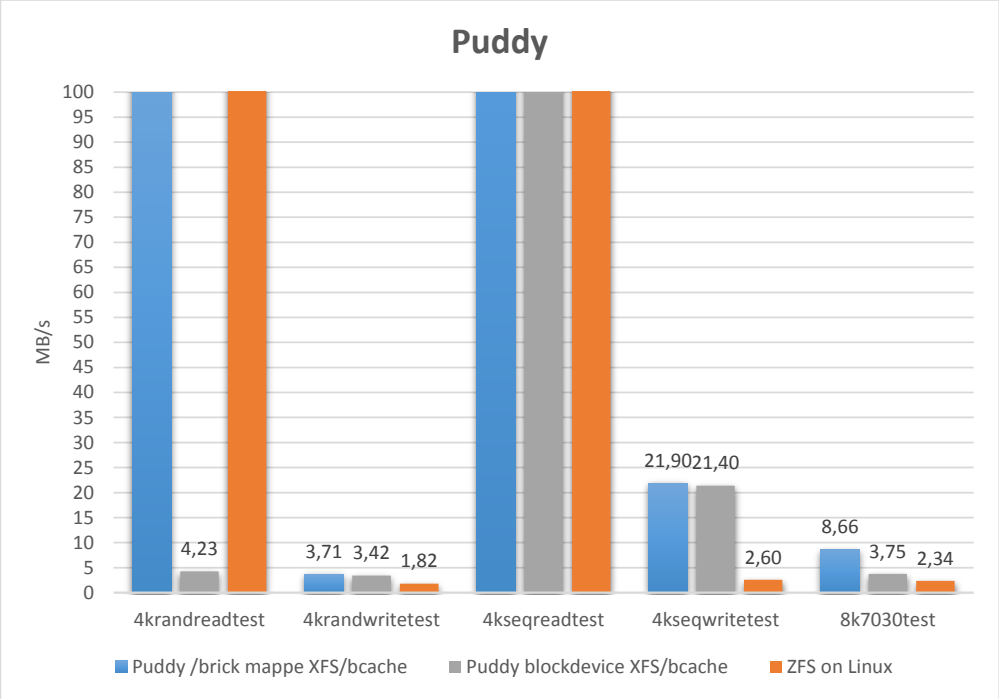
Ser man da på tabell 7.1 vil man se at standardavviket for /brickt mappe testingen vil være merkbart høyt. Mens for block devicen er dette akseptabelt lavt igjen. Det at block devicene hoder et lavt standardavvik vil da tilsi at resultatene er henholdvis normalfordelte.

Noe annet som bør merkes er at her vil man ha en reduksjon i ytelsen på 4krandwritetest. Dette vil være et resultat av at det forekommer mindre cache hits og flere oppslag vil bli gjort.

Videre ser man at det er endringer i 4kseqread test. Her vil block devicen ligge på 200 MB/s mens /brick mappe vil ligge på 3,64 MB/s. Grunnen til at /brick mappen har så lav ytelse vil da tilsi at det forekommer lite cache hits, samt at siden dette er en sekvensiell test vil det også ha betydning med mengden overhead (metadata) som blir produsert. Dette kombinert vil føre til at ytelsen blir betraktelig redusert.

Når det kommer til 8k7030test er /brick mappe resultatene som er de mest interessante. Her vil XFS ha en høyre ytelse enn ZFS on Linux. Standardavviket er lavt slik at man har en normalfordeling av observasjoner. Dette vil da tilsi at XFS vil yte bedre enn ZFS on Linux i en kombinasjons test (read og write) samt at blockstørrelsen ligger på 8k. Det tilsier også at testene fra kapittel 5.1 går noe ut. Med dette menes det at dersom man ikke tester direkte på block device først, og risikerer å ødelegge filsystemet, kan man får en høyere ytelese utav oppsettet. Det bør også merkes at alle ZFS on Linux data ble testet i mapper og ikke på block devices slik som XFS.

Den neste testen er fra serveren Puddy:

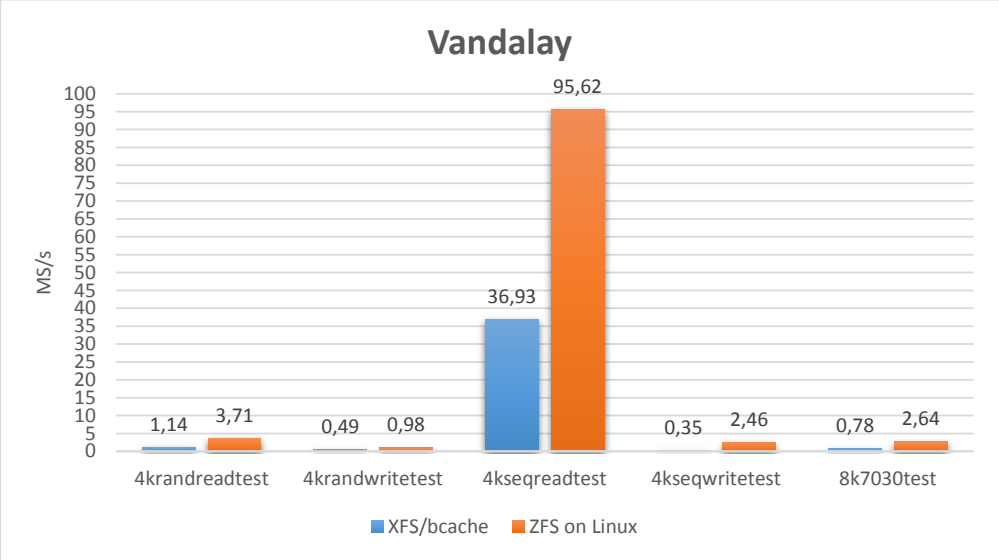



Figur 7.2: Fiotest Puddy Ny

Det merkbare her i denne testen vil være at testen 4kseqread vil ligge på 201,4 MB/s for /brick og 202,46 MB/s for block devicen. Dette vil være signifikant høyere enn Lippman, noe som ikke er forventet. Det som skjer her er at cachen vil ha cachet hele den sekvensielle blokken på en korrekt måte, og Lippman vil ikke ha gjort det.

Resten av testene har marginiale endringer slik at disse blir sett på som like med Lippmans tester.

Den neste testen er fra klientmaskinen Vandalay:



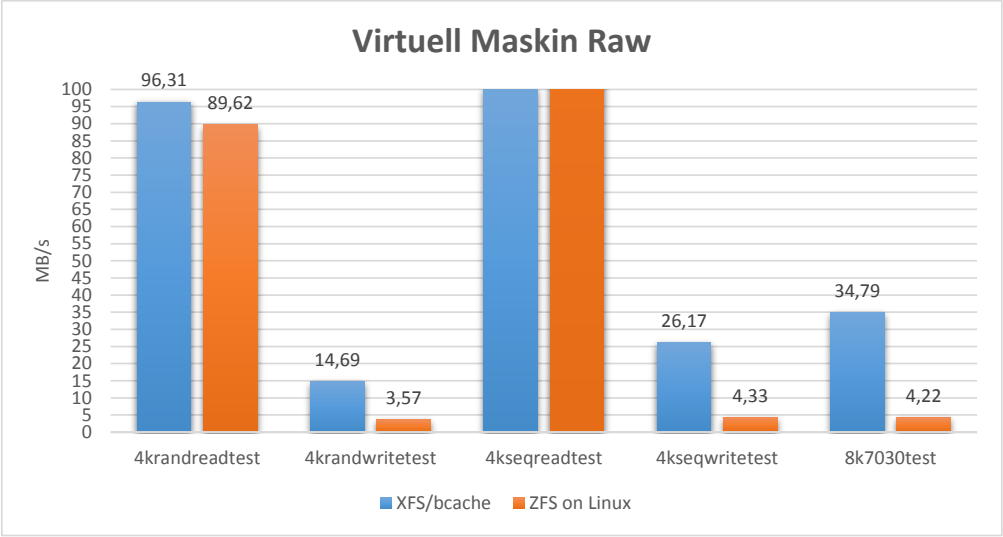

Figur 7.3: Fiotest Vandalay Ny

Det første som bør merkes her at man har en endring i 4krandwritetest, her vil man ha en reduksjon fra den forrige testen i kap 5.1. Men siden denne er en random test vil dette reflektere cache hits. Det vil da si at i denne testen er cache hit raten mindre enn ved den forrige.

Det neste som bør merkes at man har en økning i 4kseqreadtest. Dette støtter opp mot påstanden om at testing på block device kan ha ødelagt filsystemet som ligger på de. Som et resultat på at det ikke ødelagt vil man få en høyere ytelse.

Men det fører også med seg høyere overhead, og mindre cache hits. Dette kan sees i testene 4kseqwritetest og 8k7030test.

Den neste testen er fra den virtuelle maskinen:




Figur 7.4: Fiotest Virtuell Maskin Ny

Her vil det foreligge små endringer, men disse er henholdvis de samme som den testen som ble utfør i kapittel 5.1. Dette tilsier at den virtuelle maskinen er mer avhengig av sine egne ressurser enn hierarkiet av lag den må gå gjennom. Men over det hele vises en økning.

# Bibliografi


[1] Wikipedia. OpenStack — Wikipedia, The Free Encyclopedia, 2014. [Online; accessed 16-January-2014].

[2] Wikipedia. Logical Volume Manager (Linux) — Wikipedia, The Free Encyclopedia, 2014. [Online; accessed 16-January-2014].

[3] Wikipedia. RAID — Wikipedia, The Free Encyclopedia, 2014. [Online; accessed 16-January-2014].

[4] Wikipedia. Cache (computing) — Wikipedia, The Free Encyclopedia, 2014. [Online; accessed 17-January-2014].

[5] Wikipedia. bcache — Wikipedia, The Free Encyclopedia, 2014. [Online; accessed 11-February-2014].

[6] evilpiepirate. bcache, 2014. [Online; accessed 02-April-2014].

[7] Franklin. How to measure IOPS for Windows, 2013. [Online; accessed 19-February-2014].

[8] Inc StorageReview.com. Fio - Flexible I/O Tester Synthetic Benchmark, 2013. [Online; accessed 19-February-2014].

[9] Jens Axboe. Howto.txt, 2014. [Online; accessed 19-February-2014].

[10] Wikipedia. GlusterFS — Wikipedia, The Free Encyclopedia, 2014. [Online; accessed 13-March-2014].

[11] GlusterNews. About, 2009. [Online; accessed 13-March-2014].

[12] opprinnelig poster Ikcl Slashdot. Power-Loss-Protected SSDs Tested: Only Intel S3500 Passes, 2013. [Online; accessed 5-April-2014].

[13] Kevin OBrien. Western Digital Red NAS Hard Drive Review [WD30EFRX], 2012. [Online; accessed 13-March-2014].

[14] Kevin OBrien. Intel SSD DC S3700 Series Enterprise SSD Review, 2012. [Online; accessed 13-March-2014].

[15] Komplett.no. WD Red 2TB NAS Harddrive, 2014. [Online; accessed 13-March-2014].

[16] newegg.com. Intel DC S3700 Series Taylorsville SSDSC2BA100G301 2.5" 100GB SATA III MLC Internal Solid State Drive (SSD), 2014. [Online; accessed 13-March-2014].

[17] Joel Santo Domingo. SSD vs. HDD: What's the Difference?, 2014. [Online; accessed 13-March-2014].

[18] Wikipedia. Hdparm — Wikipedia, The Free Encyclopedia, 2014. [Online; accessed 1-April-2014].

[19] Bitpipe. High Availability, 2014. [Online; accessed 5-April-2014].

[20] Webopedia. High Availability, 2014. [Online; accessed 5-April-2014].

[21] Wikipedia. ISCSI — Wikipedia, The Free Encyclopedia, 2014. [Online; accessed 23-January-2014].

[22] Margaret Rouse. iSCSI (Internet Small Computer System Interface), 2011. [Online; accessed 23-January-2014].

[23] linux kvm. Kernel Based Virtual Machine, 2014. [Online; accessed 13-March-2014].

[24] Inc Red Hat. KVM – KERNEL BASED VIRTUAL MACHINE, 2009. [Online; accessed 13-March-2014].

[25] libvirt. The virtualization API, 2014. [Online; accessed 13-March-2014].

[26] Wikipedia. Libvirt — Wikipedia, The Free Encyclopedia, 2014. [Online; accessed 13-March-2014].

[27] archlinux. libvirt, 2014. [Online; accessed 13-March-2014].

[28] Inc. Rackable Systems. LVM HOWTO, 2006. [Online; accessed 16-January-2014].

[29] Wikipedia. Qcow — Wikipedia, The Free Encyclopedia, 2013. [Online; accessed 14-March-2014].

[30] Wikipedia. Storage area network — Wikipedia, The Free Encyclopedia, 2014. [Online; accessed 12-February-2014].

[31] QuinStreet Inc. SAN - Storage Area Network, 2014. [Online; accessed 12-February-2014].

[32] Margaret Rouse. storage area network (SAN), 2007. [Online; accessed 12-February-2014].

[33] Wikipedia. Write barrier — Wikipedia, The Free Encyclopedia, 2014. [Online; accessed 11-February-2014].

[34] Inc Red Hat. 22.2. Enabling/Disabling Write Barriers, 2014. [Online; accessed 11-February-2014].

[35] Wikipedia. Allocate-on-flush — Wikipedia, The Free Encyclopedia, 2011. [Online; accessed 11-February-2014].

[36] Jarret W. Buse. Allocation Methods, 2013. [Online; accessed 11-February-2014].

[37] Wikipedia. DMAPI — Wikipedia, The Free Encyclopedia, 2013. [Online; accessed 22-January-2014].

[38] Jarret W. Buse. DMAPI (Data Management API), 2013. [Online; accessed 22-January-2014].

[39] Wikipedia. XFS — Wikipedia, The Free Encyclopedia, 2014. [Online; accessed 22-January-2014].

[40] Thomas Graichen and Seth Mos. XFS FAQ, 2014. [Online; accessed 22-January-2014].

[41] Silicon Graphics International Corp. XFS: A high-performance journaling filesystem, 2012. [Online; accessed 22-January-2014].

[42] Wikipedia. ZFS — Wikipedia, The Free Encyclopedia, 2014. [Online; accessed 01-February-2014].

[43] Lawrence Livermore National Laboratory. ZFS on Linux. , 2013. [Online; accessed 01-February-2014].

[44] linux.slashdot.org. Facebook To Begin Deploying Btrfs, 2014. [Online; accessed 11-April-2014].

[45] die.net. attr(1) - Linux man page, 2014. [Online; accessed 02-April-2014].

[46] evilpiepirate. bcache, 2014. [Online; accessed 11-February-2014].

[47] GamezR2EZ. [Howto] bcache, 2013. [Online; accessed 06-February-2014].

[48] archlinux. Bcache, 2014. [Online; accessed 02-April-2014].

[49] RPM Pbone. make-bcache, 2014. [Online; accessed 11-February-2014].

[50] Jens Axboe and Aaron Carrol. fio(1) - Linux man page, 2014. [Online; accessed 19-February-2014].

[51] Lzone. GlusterFS Administration Commands, 2014. [Online; accessed 04-March-2014].

[52] James Coyle. HOW-TO MOUNT A GLUSTERFS VOLUME, 2013. [Online; accessed 20-February-2014].

[53] Gluster community. Gluster 3.1: Manually Mounting Volumes, 2011. [Online; accessed 20-February-2014].

[54] Inc Red Hat. 8.8. Creating Striped Replicated Volumes, 2014. [Online; accessed 23-February-2014].

[55] Hewlett Packard. Hpacucli Utility for Linux - All Commands Guide, 2014. [Online; accessed 6-April-2014].

[56] vita preskovsky. isc-dhcp-server, 2014. [Online; accessed 04-February-2014].

[57] chronospoon. How do I configure a DHCP server?, 2013. [Online; accessed 04-February-2014].

[58] Ubuntu documentation. libvirt, 2014.

[59] NIX CRAFT. Debian / Ubuntu Linux Connect to an iSCSI Volume, 2007. [Online; accessed 06-February-2014].

[60] Alex Aizman Open-iSCSI project and Dmitry Yusupov. iscsid(8) - Linux man page, 2014. [Online; accessed 06-February-2014].

[61] Alex Aizman Open-iSCSI project and Dmitry Yusupov. iscsiadm(8) - Linux man page, 2014. [Online; accessed 06-February-2014].

[62] Fabrice Bellard. qemu-system-x86_64, 2011. [Online; accessed 24-March-2014].

[63] Bharata B Rao. QEMU-GlusterFS native integration, 2014. [Online; accessed 18-March-2014].

[64] Fabrice Bellard. qemu-img(1) - Linux man page, 2014. [Online; accessed 18-March-2014].

[65] Juergen Weigert Wayne Davison and Michael Schroeder. Screen User's Manual, 2014. [Online; accessed 10-February-2014].

[66] Andy Grover. targetcli(8) - Linux man page, 2014. [Online; accessed 06-February-2014].

[67] Jerome Martin. targetcli, 2014. [Online; accessed 06-February-2014].

[68] IBM. The virt-install command options, 2014. [Online; accessed 20-February-2014].

[69] Inc Red Hat. virt-install(1) - Linux man page, 2011. [Online; accessed 20-February-2014].

[70] Dustin Kirkland. mkfs.xfs - construct an XFS filesystem, 2010. [Online; accessed 23-January-2014].

[71] DiceHoldings Inc. fio, 2014. [Online; accessed 12-April-2014].

[72] John Mark Walker. GlusterFS Cloud Storage, 2013.

[73] Jiri Hoogeveen. please add glusterfs support, 2013. [Online; accessed 26-March-2014].

[74] kashyapc. Snapshots - handout, 2014. [Online; accessed 24-March-2014].

[75] MPX.no. MPX.no - UPS, 2014. [Online; accessed 18-April-2014].

[76] RedHat Inc. Triggering Self-Healing on Replicated Volumes, 2014. [Online; accessed 24-April-2014].

[77] archlinux. hdparm, 2014. [Online; accessed 01-April-2014].

[78] Mark Lord. hdparm(8) - Linux man page, 2005. [Online; accessed ].

# Appendices

# Tillegg A

# Stikkordliste

| | |
|---|---|
| API | Application Programming Interface, grensesnitt for et program. |
| Applikasjon | En type software som er innstallert på serverenen. |
| Asynkron(ikke synkron) | Motsatt av synkron. |
| B | Byte (8 bit, eller $2^3$). |
| Backend | Maskin som ligger i bakkant på lagringsboksene. |
| Backup | Ta kopi av data i tilfelle systemet går ned. |
| Baseline | Oversatt til norsk, referansepunkt. |
| Bcache | Software brukt på frontend i løsning 1. |
| Block device | Annet navn for hard disk. |
| Block size | En sekvens med bits(blokker) i en gitt størrelse. |
| Btrfs | Alternativt software for oppsett 1, på frontend. |
| Buffered I/O | Laster all data før det skrives / leses. |
| Cache | En enhet for at nylig brukte data senere kan hentes ut raskere. |
| CPU | Central Processing Unit - prosessor, enhet som håndterer prosessering. |
| C-bibliotek | Samling C-kode filer, organisert i et bibliotek. |
| Data | Et annet begrep for informasjon. |
| Directory | Mappe, katalogisering av data. |
| Disk | Hardware for lagring av data. |
| Disk-image | Et “bilde” av hvordan man ønsker at den virtuelle maskinen skal se ut. |
| Disk read | Lesing av data fra disk. |
| Frontend | Maskin som ligger i forkant på lagringsboksene. |

| | |
|---|---|
| FUSE(Filesystem in User SpacE) | Tillater opprettelse av user space filsystemer, uten endring i kernel. |
| GB | Giga byte (1024 MB, eller $2^30$). |
| Hardware | Maskinens fysiske komponenter. |
| Hardware virtualisering | Virtualisering ved bruk av hardware. |
| Hot swapping | Bytting av disker når serveren er i kjørende tilstand. |
| Hypervisor | Software/hardware som lager og kjører virtuelle maskiner. |
| Input | Informasjon som blir send til et program. |
| Intranet | En bedrift sitt interne “internett”. |
| IOPS | I/O-operasjoner per sekund. |
| I/O(input/output) | Se input og output. |
| KB | Kilo byte (1024 B, eller $2^10$). |
| Kernel | En modus i linux for håndtering av I/O mot CPU og annet hardware. |
| Komprimering | Reduksjon av størresle, her i forhold til filer. |
| Kommandolinje(cli) | Tekstbasert grensesnitt mot hardware og software. |
| LAN | Local Area Network, sammenkobling av servere internt i en bedrift. |
| Lagringsboks | To servere koblet sammen som en, håndterer cache og backup. |
| Libgfapi | C-bibliotek for å kunne kommunisere med GlusterFS uten å bruke FUSE. |
| Libvirt-bin | Software brukt på klientmaskin. |
| Lun | Logical Unit Number, brukt av SCSI for å holde orden på lagringsmedier. |
| LVM(Logical Volume Manager) | Software brukt på backend server. |
| MB | Mega byte (1024 KB, eller $2^20$). |

| | |
|---|---|
| Minne | Hardware som gir tilgang til all lagrede data, også kalt RAM. |
| Native | Bruker direkte kommandoer mot kernel, uten noe ekstra form for grensesnitt. |
| NIC(Network Interface Card) | Hardware som brukes for nettverktilgang for serveren. |
| OS(OperativSystem) | En samling av all software som styrer hardware på en server. |
| Output | Resultat ved kjøring av et program, for eksempel tekst vist i nettleser. |
| Parameter | Argument på tekstform, egendefinerte valg. |
| Paravirtualisering | Virtualisering ved presentasjon av hardware som software. |
| RAID(Redundant Array of Inexpensive Disks) | Sammenkobling av flere disker. |
| Random read | Tilfeldig lesing av data fra disk. |
| Random write | Tilfeldig skriving av data til disk. |
| Recovery | Gjenoppretting, ofte brukt på data ved en kræsj. |
| Service | Ord brukt som alternativ til software. |
| Software | Programvare. |
| Storage | Lagring. |
| Synkron | Sender og mottaker av data bruker samme referanse. |
| UPS(Uninterruptable Power Supply) | Enhet som skal håndtere sikker avslutning ved strømbrudd. |
| Vnc-client | En server som har støtte for vnc, tilkoblet klient-maskin i oppsettet. |
| Vertsserver | Server som holder tilstanden til de virtuelle maskinene. |

| | |
|---|---|
| Virtuell maskin | Server som kjører på vertsserver, finnes ikke fysisk. |
| WAN | Wide Area Network, sammenkobling av servere på geografisk splittede lokasjoner. |
| XFS | Filsystem brukt i oppsett 1. |
| ZFS | Kombinert filsystem og logical volume manager(se LVM) brukt i oppsett 2. |
| zlib | Software-bibliotek for komprimering av data. |

# Tillegg B

# Script

## B.1 Fiotester XFS

### B.1.1 Frontend

```
#!/bin/bash

for x in {0..4}; do
hdparm -tT /dev/bcache$x >>
~/test-resultater/hdparm_baseline.txt

for i in {1..50}; do
fio --filename=/dev/bcache$x --direct=1
--rw=randrw --refill_buffers --norandommap
--randrepeat=0 --ioengine=libaio --bs=4k
--rwmixwrite=100 --iodepth=16 --numjobs=16
--runtime=60 --group_reporting --name=4krandwritetest >>
~/test-resultater/fio-4krandwritetest.txt
sleep 30;
done

for i in {1..50}; do
fio --filename=/dev/bcache$x --direct=1
--rw=randrw --refill_buffers --norandommap
--randrepeat=0 --ioengine=libaio --bs=4k
--rwmixread=100 --iodepth=16 --numjobs=16
--runtime=60 --group_reporting --name=4krandreadtest >>
~/test-resultater/fio-4krandreadtest.txt
sleep 30;
done

for i in {1..50}; do
fio --filename=/dev/bcache$x --direct=1
--rw=randrw --refill_buffers --norandommap
--randrepeat=0 --ioengine=libaio --bs=8k
--rwmixread=70 --iodepth=16 --numjobs=16
--runtime=60 --group_reporting --name=8k7030test
```

```
>> ~/test-resultater/fio-8k7030test.txt
sleep 30;
done

for i in {1..50}; do
fio --filename=/dev/bcache$x --direct=1 --rw=write
--refill_buffers --ioengine=libaio --bs=4k
--iodepth=16 --numjobs=16 --runtime=60
--group_reporting --name=4kseqwritetest >>
~/test-resultater/fio-4kseqwritetest.txt
sleep 30;
done

for i in {1..50}; do
fio --filename=/dev/bcache$x --direct=1 --rw=read
--refill_buffers --ioengine=libaio  --bs=4k
--iodepth=16 --numjobs=16 --runtime=60
--group_reporting --name=4kseqreadtest >>
~/test-resultater/fio-4kseqreadtest.txt
sleep 30;
done

done;
```

### B.1.2 Klient-maskin

```
#!/bin/bash

for i in {1..50}; do
fio --directory=/gluster-mount-folder --size=128M
--direct=1 --rw=randrw --refill_buffers
--norandommap --randrepeat=0 --ioengine=libaio
--bs=4k --rwmixwrite=100 --iodepth=16
--numjobs=16 --runtime=60 --group_reporting
--name=4krandwritetest >>
~/test-resultater/fio-4krandwritetest.txt
sleep 30;
done

for i in {1..50}; do
fio --directory=/gluster-mount-folder --size=128M
--direct=1 --rw=randrw --refill_buffers
--norandommap --randrepeat=0 --ioengine=libaio
--bs=4k --rwmixread=100 --iodepth=16
--numjobs=16 --runtime=60 --group_reporting
--name=4krandreadtest >> ~/test-resultater/fio-4krandreadtest.txt
sleep 30;
done





for i in {1..50}; do
fio --directory=/gluster-mount-folder --size=128M
--direct=1 --rw=randrw --refill_buffers
--norandommap --randrepeat=0 --ioengine=libaio
--bs=8k --rwmixread=70 --iodepth=16 --numjobs=16
--runtime=60 --group_reporting --name=8k7030test
```

```
>> ~/test-resultater/fio-8k7030test.txt
sleep 30;
done

for i in {1..50}; do
fio --directory=/gluster-mount-folder --size=128M
--direct=1 --rw=write --refill_buffers
--ioengine=libaio --bs=4k --iodepth=16
--numjobs=16 --runtime=60 --group_reporting
--name=4kseqwritetest >>
~/test-resultater/fio-4kseqwritetest.txt
sleep 30;
done

for i in {1..50}; do
fio --directory=/gluster-mount-folder --size=128M
--direct=1 --rw=read --refill_buffers
--ioengine=libaio --bs=4k --iodepth=16
--numjobs=16 --runtime=60 --group_reporting
--name=4kseqreadtest >>
~/test-resultater/fio-4kseqreadtest.txt
sleep 30;
done
```

### B.1.3 Virtuelle maskiner

```
#!/bin/bash

for i in {1..50}; do
fio --directory=/test-fio --size=128M --direct=1
--rw=randrw --refill_buffers --norandommap
--randrepeat=0 --ioengine=libaio --bs=4k
--rwmixwrite=100 --iodepth=16 --numjobs=16
--runtime=60 --group_reporting
--name=4krandwritetest >>
~/test-resultater/4krandwritetest.txt
sleep 30;
done

for i in {1..50}; do
fio --directory=/test-fio --size=128M --direct=1
--rw=randrw --refill_buffers --norandommap
--randrepeat=0 --ioengine=libaio --bs=4k
--rwmixread=100 --iodepth=16 --numjobs=16
--runtime=60 --group_reporting
--name=4krandreadtest >>
~/test-resultater/4krandreadtest.txt
sleep 30;
done




for i in {1..50}; do
fio --directory=/test-fio --size=128M --direct=1
--rw=randrw --refill_buffers --norandommap
--randrepeat=0 --ioengine=libaio --bs=8k
--rwmixread=70 --iodepth=16 --numjobs=16
--runtime=60 --group_reporting --name=8k7030test
```

```
>> ~/test-resultater/8k7030test.txt
sleep 30;
done

for i in {1..50}; do
fio --directory=/test-fio --size=128M --direct=1
--rw=write --refill_buffers --ioengine=libaio
--bs=4k --iodepth=16 --numjobs=16 --runtime=60
--group_reporting --name=4kseqwritetest >>
~/test-resultater/4kseqwritetest.txt
sleep 30;
done

for i in {1..50}; do
fio --directory=/test-fio --size=128M --direct=1
--rw=read --refill_buffers --ioengine=libaio
--bs=4k --iodepth=16 --numjobs=16 --runtime=60
--group_reporting --name=4kseqreadtest >>
~/test-resultater/4kseqreadtest.txt
sleep 30;
done
```

## B.2 Fiotester ZFS on Linux

### B.2.1 Frontend

```
#!/bin/bash

for i in {1..50}; do
fio --directory=/pool --size=128M --direct=0
--fallocate=none --rw=randrw --refill_buffers
--norandommap --randrepeat=0 --ioengine=sync
--bs=4k --rwmixwrite=100 --iodepth=16 --numjobs=16
--runtime=60 --group_reporting
--name=4krandwritetest >>
~/test-resultater/4krandwritetest.txt
sleep 30;
done

for i in {1..50}; do
fio --directory=/pool --size=128M --direct=0
--fallocate=none --rw=randrw --refill_buffers
--norandommap --randrepeat=0 --ioengine=sync
--bs=4k --rwmixread=100 --iodepth=16 --numjobs=16
--runtime=60 --group_reporting
--name=4krandreadtest >>
~/test-resultater/4krandreadtest.txt
sleep 30;
done



for i in {1..50}; do
fio --directory=/pool --size=128M --direct=0
--fallocate=none --rw=randrw --refill_buffers
--norandommap --randrepeat=0 --ioengine=sync
--bs=8k --rwmixread=70 --iodepth=16 --numjobs=16
--runtime=60 --group_reporting --name=8k7030test
```

```
>> ~/test-resultater/8k7030test.txt
sleep 30;
done

for i in {1..50}; do
fio --directory=/pool --size=128M --direct=0
--fallocate=none --rw=write --refill_buffers
--ioengine=sync --bs=4k --iodepth=16 --numjobs=16
--runtime=60 --group_reporting
--name=4kseqwritetest >>
~/test-resultater/4kseqwritetest.txt
sleep 30;
done

for i in {1..50}; do
fio --directory=/pool --size=128M --direct=0
--fallocate=none --rw=read --refill_buffers
--ioengine=sync --bs=4k --iodepth=16 --numjobs=16
--runtime=60 --group_reporting
--name=4kseqreadtest >>
~/test-resultater/4kseqreadtest.txt
sleep 30;
done
```

### B.2.2 Klient-maskin

```
#!/bin/bash

for i in {1..50}; do
fio --directory=/gluster-mount-folder --size=128M
--direct=0 --fallocate=none --rw=randrw
--refill_buffers --norandommap --randrepeat=0
--ioengine=sync --bs=4k --rwmixwrite=100
--iodepth=16 --numjobs=16 --runtime=60
--group_reporting --name=4krandwritetest >>
~/test-resultater/4krandwritetest.txt
sleep 30;
done

for i in {1..50}; do
fio --directory=/gluster-mount-folder --size=128M
--direct=0 --fallocate=none --rw=randrw
--refill_buffers --norandommap --randrepeat=0
--ioengine=sync --bs=4k --rwmixread=100
--iodepth=16 --numjobs=16 --runtime=60
--group_reporting --name=4krandreadtest >>
~/test-resultater/4krandreadtest.txt
sleep 30;
done




for i in {1..50}; do
fio --directory=/gluster-mount-folder --size=128M
--direct=0 --fallocate=none --rw=randrw
--refill_buffers --norandommap --randrepeat=0
--ioengine=sync --bs=8k --rwmixread=70
--iodepth=16 --numjobs=16 --runtime=60
```

```
--group_reporting --name=8k7030test >>
~/test-resultater/8k7030test.txt
sleep 30;
done

for i in {1..50}; do
fio --directory=/gluster-mount-folder --size=128M
--direct=0 --fallocate=none --rw=write
--refill_buffers --ioengine=sync --bs=4k
--iodepth=16 --numjobs=16 --runtime=60
--group_reporting --name=4kseqwritetest >>
~/test-resultater/4kseqwritetest.txt
sleep 30;
done

for i in {1..50}; do
fio --directory=/gluster-mount-folder --size=128M
--direct=0 --fallocate=none --rw=read
--refill_buffers --ioengine=sync --bs=4k
--iodepth=16 --numjobs=16 --runtime=60
--group_reporting --name=4kseqreadtest >>
~/test-resultater/4kseqreadtest.txt
sleep 30;
done
```

### B.2.3 Virtuelle maskiner

```
#!/bin/bash

for i in {1..50}; do
fio --directory=/test-fio --size=128M --direct=0
--fallocate=none --rw=randrw --refill_buffers
--norandommap --randrepeat=0 --ioengine=sync
--bs=4k --rwmixwrite=100 --iodepth=16 --numjobs=16
--runtime=60 --group_reporting
--name=4krandwritetest >>
~/test-resultater/4krandwritetest.txt
sleep 30;
done

for i in {1..50}; do
fio --directory=/test-fio --size=128M --direct=0
--fallocate=none --rw=randrw --refill_buffers
--norandommap --randrepeat=0 --ioengine=sync
--bs=4k --rwmixread=100 --iodepth=16 --numjobs=16
--runtime=60 --group_reporting
--name=4krandreadtest >>
~/test-resultater/4krandreadtest.txt
sleep 30;
done




for i in {1..50}; do
fio --directory=/test-fio --size=128M --direct=0
--fallocate=none --rw=randrw --refill_buffers
--norandommap --randrepeat=0 --ioengine=sync
--bs=8k --rwmixread=70 --iodepth=16 --numjobs=16
--runtime=60 --group_reporting --name=8k7030test
```

```
>> ~/test-resultater/8k7030test.txt
sleep 30;
done

for i in {1..50}; do
fio --directory=/test-fio --size=128M --direct=0
--fallocate=none --rw=write --refill_buffers
--ioengine=sync --bs=4k --iodepth=16 --numjobs=16
--runtime=60 --group_reporting
--name=4kseqwritetest >>
~/test-resultater/4kseqwritetest.txt
sleep 30;
done

for i in {1..50}; do
fio --directory=/test-fio --size=128M --direct=0
--fallocate=none --rw=read --refill_buffers
--ioengine=sync --bs=4k --iodepth=16 --numjobs=16
--runtime=60 --group_reporting
--name=4kseqreadtest >>
~/test-resultater/4kseqreadtest.txt
sleep 30;
done
```

## B.3 Nettverksscript

```
#!/bin/bash

# Setter paa routing paa server
echo 1 > /proc/sys/net/ipv4/ip_forward;

# Flushe iptables.rules
iptables --flush;
iptables -t nat -D POSTROUTING -o em2 -j MASQUERADE;
iptables -t nat -D POSTROUTING -o em1 -j MASQUERADE;

# Legge til nye regler i iptables.rules
iptables -t nat -A POSTROUTING -o em2 -j MASQUERADE;
iptables -t nat -A POSTROUTING -o em1 -j MASQUERADE;
iptables -A FORWARD -i em2 -o em1 -m state --state RELATED,
ESTABLISHED -j ACCEPT;
iptables -A FORWARD -i em1 -o em2 -j ACCEPT;

# Lagre reglene
sysctl -p /etc/sysctl.conf;
iptables-save > /etc/iptables.rules;

# Skru nettverkskortene av og paa
ifdown -a;
ifup -a --force;
```

## B.4 ISCSI, kobling mot backend

```
#!/bin/bash

iscsiadm --mode discovery --type sendtargets --portal <ip backend>
iscsiadm --mode node --targetname
iqn.2014-02.no.hig.hdbox1:<hostname backend>
--portal <ip backend>:3260 --login
```

## B.5 Scenariotesting

Script for scenario 1, kjørt i vm:

```
for i in {1..50}; do
for x in {1..50}; do
echo "skriver fil$i$x"
dd if=/dev/zero of=./fil$i$x bs=4k count=1000
done
done;
```

Script for scenario 2, kjørt på klientmaskin:

```
for i in {1..10}; do
for x in {1..10}; do
echo "skriver fil$i$x"
dd if=/dev/zero of=./fil$i$x bs=4k count=1000
done
done;
```

## B.6 Fio-kommandoer

1. `hdparm -tT <$block-device$> >> ~/test-resultater/hdparm_baseline.txt`

2. 
```
fio --filename=<block-device> --direct=1 --rw=randrw
--refill_buffers --norandommap --randrepeat=0
--ioengine=libaio --bs=4k --rwmixwrite=100 --iodepth=16
--numjobs=16 --runtime=60 --group_reporting
--name=4krandwritetest >> ~/test-resultater/4krandwritetest.txt
```

3. 
```
fio --filename=<block-device> --direct=1 --rw=randrw
--refill_buffers --norandommap --randrepeat=0
--ioengine=libaio --bs=4k --rwmixread=100 --iodepth=16
--numjobs=16 --runtime=60 --group_reporting
--name=4krandreadtest >> ~/test-resultater/4krandreadtest.txt
```

4. 
```
fio --filename=<block-device> --direct=1 --rw=randrw
--refill_buffers --norandommap --randrepeat=0
--ioengine=libaio --bs=8k --rwmixread=70 --iodepth=16
--numjobs=16 --runtime=60 --group_reporting
--name=8k7030test >> ~/test-resultater/8k7030test.txt
```

5. 
```
fio --filename=<block-device> --direct=1 --rw=write
--refill_buffers --ioengine=libaio --bs=4k
--iodepth=16 --numjobs=16 --runtime=60 --group_reporting
--name=4kseqwritetest >> ~/test-resultater/4kseqwritetest.txt
```

6. 
```
fio --filename=<block-device> --direct=1 --rw=read
--refill_buffers --ioengine=libaio --bs=4k
--iodepth=16 --numjobs=16 --runtime=60 --group_reporting
--name=4kseqreadtest >> ~/test-resultater/4kseqreadtest.txt
```

# Tillegg C

# Brukte parametere

## C.1 Hdparm

Dette er en kort forklaring av de parameterne brukt i hdparm-testen. [77, 78]

- **-t:** gjennomfører tidsbaserte tester mot disken for å måle hvor stor den maksimale ytelsen er på, med tanke på lesing(disk read). Det bør merkes at lesingene skjer sekvensielt.
- **-T:** gjennomfører tidsbaserte tester mot cache og buffer, som igjen tilsier at testingen ikke går innom disk.

## C.2 Fio

Samlet sett er disse parameterne brukt: [9, 50]

- **fio:** er en service som brukes for å kjøre tester mot en disk eller directory.
- **–filename=<block-device>:** dette tilsvarer den block devicen testene skal kjøres mot.
- **–direct=1:** sier at det ikke skal brukes buffered I/O, altså unngå cache'en til filsystemet.
- **–rw:** denne parameteren brukes for å definere om det skal gjøres skriving, lesing eller begge deler, samt om lesing/skriving skal skje random eller sekvensielt.
- **–refill_buffers:** gjør at fio vil fylle I/O buffer'et hver gang det kjøres en test.
- **–ioengine=libaio:** spesifiserer hvordan I/O skal gjøres, her ved bruk av linux sin native asynkrone I/O.
- **–bs=4k:** definerer block size som skal brukes under testing, default er 4k så denne parameteren er ikke nødvendig, men grei å ha med for å vise hva block size er.
- **–iodepth=16:** viser til hvor mange jobber som står i kø for å jobbe på fio-filene.
- **–numjobs=16:** dette er antallet jobber som startes samtidig når fio-testen kjøres.
- **–runtime=60:** spesifierer hvor lenge testen skal kjøre.
- **–group_reporting:** samler sammen resultatene for alle jobbene som ble kjørt i fio-testen, og viser en gjennomsnittlig oversikt over ytelsen.
- **–name:** er kun for å gi et navn til testen, så det er lettere å holde oversikten dersom man kjører flere tester etterhverandre.

Til slutt i kommandoene kommer >> *[navn på fil hvor test-resultatene lagres]*, slik at det kan tas vare på resultatene til senere, og dermed enklere sammenligne resultater opp mot hverandre.

## C.3 Qemu-img

Når det kommer til den første kommandoen, brukt ved installasjon med libgfapi vil de ulike parameterene bety følgende: [64, 63]

- **qemu-img create:** er en service som kan brukes for å opprette et image.
- **-f qcow2:** sier at formatet på image skal være qcow2, jfr. kapittel 2.3.
- **gluster:** er protokollen som skal brukes for kommunikasjonen med GlusterFS-server.
- **10.0.4.2:24007:** er ip-addressen til serveren hvor gluster volume kjører, med tilhørende port for glusterd på denne serveren.
- **bsrmgrSAN:** dette er navnet på gluster volume.
- **libgfapi-test.img:** vil være navnet på disk-image som lages.
- **5G:** er maks størrelsen på disk-image, noe som igjen betyr at 5GB er maks størrelse den virtuelle maskinen kan bruke.

Dette er forklaring for paramterne brukt for å installere ved bruk av FUSE: [64]

- **qemu-img create:** er en service som kan brukes for å opprette et image.
- **-f qcow2:** sier at formatet på image skal være qcow2, jfr. kapittel 2.3.
- **disk.qcow2:** vil være navnet på disk-image som lages.
- **25G:** er maks størrelsen på disk-image, noe som igjen betyr at 25GB er maks størrelse den virtuelle maskinen kan bruke.

## C.4 Qemu-system

Her er parameterne med en kort forklaring: [62, 63]

- **qemu-system-x86_64:** er en emuleringsteknikk for å installere og organisere virtuelle maskiner.
- **-enable-kvm:** gjør slik at den virtuelle maskinen kjører med full KVM-støttet virtualisering.
- **-nographic:** er en parameter som gjør qemu til en enkel kommandolinje applikasjon.
- **-smp 2:** tilsier hvor mange virtuelle cpu'er den virtuelle maskinen har lov til å bruke.
- **-m 2048:** tilsier hvor mye virtuelt minne den virtuelle maskinen blir tildelt.
- **-cdrom /iso/ubuntu-server.iso:** dette er fila som blir brukt som cdrom image ved installasjon av den virtuelle maskinen, altså et alternativ til en fysisk cd-plate.
- **-drive:** definerer en disk, og har flere alternativer:
  - **file=gluster://10.0.4.2:24007/bsrmgrSAN/libgfapi-test.img:** viser til hvilket disk-image som den virtuelle maskinen skal bruke som lagringsenhet. De forskjellige delene er forklart ovenfor.
  - **if=virtio:** dette forklarer hvilken type grensesnitt disk-image er tilkoblet med.
  - **cache=none:** sier hva slags metode caching i den virtuelle maskinen skal være, i vårt tilfelle avskrudd.

## C.5 Virt-install

Her er forklaringen av parameterne brukt i kommandoen virt-install: [69, 68]

- **virt-install:** er en service som brukes for å installere virtuelle maskiner.
- **–connect qemu:///system:** dette sier hvilken hypervisor som den virtuelle maskinen skal bruke. qemu:///system er det alternativet som skal brukes ved full hardware virtualisering gjennom KVM.
- **-n ubuntu-server:** er navnet på den virtuelle maskinen.
- **-r 2047:** tilsvarer hvor mye minne den virtuelle maskinen får tildelt, her 2GB.
- **–disk:** spesifiserer hvilket media som skal brukes til lagring for den virtuelle maskinen, og har en rekke alternativer:
  - **path=/gluster-mount-folder/disk.qcow2:** lokasjon til lagringsmedia som skal brukes. Kan allerede være allokert, men trenger ikke.
  - **format=qcow2:** spesifisering av type disk-image som skal brukes.
- **-c /iso/ubuntu-server.iso:** dette er fila som blir brukt som cdrom image ved installasjon av den virtuelle maskinen, altså et alternativ til en fysisk cd-plate.
- **–noautoconsole:** gjør at vnc-clienten ikke prøver å koble til den virtuelle maskinen automatisk.
- **–graphics vnc,password=foobar,listen=0.0.0.0:** spesifisering av det grafiske grensesnittet, passordet for tilkobling via vnc-client er foobar, og den lytter etter tilkoblinger fra alle ipv4-adresser.
- **–os-type linux:** sier at os'et på den virtuelle maskinen skal være linux.
- **–os-variant ubuntusaucy:** sier hvilken versjon av linux som benyttes på den virtuelle maskinen, ubuntusaucy er versjon 13.10, også kalt Saucy Salamander.
- **–accelerate:** gjør at kvm er foretrukket hypervisor ved installasjon av virtuelle maskiner.
- **–network=network:default:** gjør at det kun settes opp en NIC på den virtuelle maskinen.
- **–vcpus=2:** tilsier at den virtuelle maskinen får tildelt 2 virtuelle CPU'er.
- **–hvm:** spesifiserer at den virtuelle maskinen bruker hardware virtualisering.

## C.6 Snapshot

Her er en forklaring av parameterne brukt for å ta snapshot av den virtuelle maskinen: [74]

- **virsh:** en service for å håndtere virtuelle maskiner.
- **snapshot-create:** parameter som tilsier at det skal tas snapshot.
- **ubuntu-server:** navnet på den virtuelle maskinen det skal tas snapshot av.
- **snapshot-revert:** parameter som tilsier at vi skal "skru" den virtuelle maskinen tilbake til en tidligere tilstand.
- **–domain:** brukes for å kunne spesifisere hvilken virtuell maskin det skal kjøre revert på.
- **1395654043:** dette er navnet på snapshot'en som er tatt.
- **–force:** dersom den virtuelle maskinen er busy(opptatt), må denne parameteren være med for å overstyre brukeren av den virtuelle maskinen. Brukes ikke dersom den virtuelle maskinen er avskrudd da kommandoen kjøres.

# Tillegg D

# Serverbasert oppsett

## D.1 Backend

**Pakker**:

- targetcli (iscsi target software, tilbyr logiske volum over iscsi)
- lvm2 (opprettelse og administrasjon av storage)

**Trinnvis oppsett**:

1. Sette OS-disker i RAID1(tray 1 og 2) og lagrings-disker i RAID10(tray 3-12) i ROM-based utilities.
2. Gjøre network boot.
3. Sette root passord(sudo su, passwd).
4. Endre hostname.
5. Endre keyboard layout:
   a. `dpkg-reconfigure keyboard-configuration`
   b. Velg Dell.
   c. Velg Norsk.
   d. Trykk Enter for resten av alternativene.
6. Reboot.
7. Logge inn igjen som root.
8. Sette riktige parametere for network interfaces:
   a. Interface 1 går mot frontend server, følgende config:

```
# The primary network interface
auto eth0
iface eth0 inet static
    address 10.0.2.2
    network 10.0.2.0
    netmask 255.255.255.0
    gateway 10.0.2.1
```

```
        broadcast 10.0.2.255
        dns-nameservers 128.39.32.2
```

9. `apt-get update`

   (passe på at alle pakkene er up to date ift. repo).

10. `apt-get install -y targetcli lvm2.`

11. Sett opp logiske partisjoner.

    a. `pvcreate /dev/cciss/c0d1`

    b. `vgcreate san /dev/cciss/c0d1`

    c. `lvcreate -n LUN0 -L 230GB san`

    d. `lvcreate -n LUN1 -L 230GB san`

    e. `lvcreate -n LUN2 -L 230GB san`

    f. `lvcreate -n LUN3 -L 230GB san`

    g. `lvcreate -n LUN4 -L 230GB san`

12. `targetcli`

    a. `backstores/iblock create LUN0 /dev/san/LUN0 (cd /)`

    b. `backstores/iblock create LUN1 /dev/san/LUN1 (cd /)`

## D.2 Frontend

Siden dette er et overordnet oppsett for hvordan frontend serverene er satt opp i begge oppsettene, vil forskjellene markeres med *XFS/bcache* for oppsett 1 og *ZFS on Linux* for oppsett 2.

**Pakker**:

- software-properties-common (add-apt-repository)
- open-iscsi (tilkobling mot backend lagring)
- ubuntu-zfs (filsystem og lvm for frontend cache og lagring, ZFS on Linux)
- bcache-tools (lage bcache og backing-devices, koble de sammen, XFS/bcache)
- xfsprogs (lage XFS filsystem på backing-devices, XFS/bcache)
- glusterfs-server (for å mounte mapper i glusterfs)
- attr (i tilfelle mounted directories i glusterfs må unmountes)
- fio (teste ytelsen på lagringsløsningen)
- screen (samarbeidsjobbing)

**Trinnvis oppsett**:

1. Sette OS-disker i RAID 1 og SSD-disk(er) i RAID1(XFS/bcache)/RAID0(ZFS on Linux) i ROM-based utilities.
2. Gjøre network boot.
3. Sette root passord(sudo su, passwd).
4. Endre hostname.
5. Endre keyboard layout:
   a. `dpkg-reconfigure keyboard-configuration`
   b. Velg Dell.
   c. Velg Norsk.
   d. Trykk Enter på resten av alternativene.
6. Reboot.
7. Logge på igjen som root.
8. Sette riktige parametere for network interfaces:
   a. Interface 1 går mot klientmaskinen, skal ha dhcp.
   b. Interface 2 går mot backend server, følgende config (network vil være forskjellig på hver av frontend serverene, 10.0.2.x på en frontend og 10.0.3.x på den andre):

```
# The secondary network interface
auto em2
iface em2 inet static
    address 10.0.2.1
    network 10.0.2.0
    netmask 255.255.255.0
    broadcast 10.0.2.255
    dns-nameservers 128.39.32.2
```

9. Sett opp routing:
   a. `echo 1 > /proc/sys/net/ipv4/ip_forward`
   b. `iptables -t nat -A POSTROUTING -o em2 -j MASQUERADE`

c. `iptables -t nat -A POSTROUTING -o em1 -j MASQUERADE`

d. `iptables -A FORWARD -i em2 -o em1 -m state --state RELATED,ESTABLISHED -j ACCEPT`

e. `iptables -A FORWARD -i em1 -o em2 -j ACCEPT`

f. `sysctl -p /etc/sysctl.conf`

g. `iptables-save > /etc/iptables.rules`

h. `nano /etc/sysctl.conf`

i. Uncomment

   `net.ipv4.ip_forward = 1`

   (save and exit).

j. `sysctl -p /etc/sysctl.conf`

k. `service networking restart`

10. `apt-get update`

    (passe på at alle pakkene er up to date ift. repo).

11. `apt-get install -y software-properties-common`

12. Pakker som trengs å legge til fra andre steder enn det som står listet i */apt/sources.list*.

    a. `add-apt-repository ppa:g2p/storage`

       (XFS/bcache).

    b. `add-apt-repository ppa:zfs-native/stable`

       (ZFS on Linux).

    c. `add-apt-repository ppa:semiosis/ubuntu-glusterfs-3.4`

    d. `apt-get update`

13. `apt-get install -y open-iscsi bcache-tools`

    (XFS/bcache)

    `xfsprogs`

    (XFS/bcache)

`ubuntu-zfs`

(ZFS on Linux)

`glusterfs-server attr fio screen`

14. Ta notat av initiatorname på frontend, kan finnes i */etc/iscsi/initiatorname.iscsi*.
15. Koble til backend lagring, skript vedlagt i appendix B.4(kjøres etter hver reboot).
16. Opprette 5 bricks(/brick0 . . . /brick4).
17. Kommandoer for XFS/bcache:
    a. `make-bcache -C /dev/sdb`

       (pass på at uuid listes ved ls /sys/fs/bcache, bcache cache device).

    b. `make-bcache -B /dev/sdc`

       (bcache backing device)

    c. `make-bcache -B /dev/sdd`

       (bcache backing device).

    d. `make-bcache -B /dev/sde`

       (bcache backing device)

    e. `make-bcache -B /dev/sdf`

       (bcache backing device).

    f. `make-bcache -B /dev/sdg`

       (bcache backing device).

    g. 
       ```
       for i in 0 1 2 3 4; do echo
        559d4576-5ba6-492e-9594-2828b17c8ada >
        /sys/block/bcache$i/bcache/attach;
        done
       ```

       (koble bcache backing devices mot bcache cache device).

    h. 
       ```
       for i in 0 1 2 3 4; do echo writeback >
       /sys/block/bcache$i/bcache/cache_mode;
       done
       ```

(sette cache mode til writeback).

i. Lage XFS filsystem på backing-devices.

- `mkfs.xfs -i size=512 /dev/bcache0`
- `mkfs.xfs -i size=512 /dev/bcache1`
- `mkfs.xfs -i size=512 /dev/bcache2`
- `mkfs.xfs -i size=512 /dev/bcache3`
- `mkfs.xfs -i size=512 /dev/bcache4`

j. Legge til diskene i /etc/fstab.

- `blkid /dev/bcache* >> /etc/fstab`

k. Endre /etc/fstab slik at hver bcache block device har "hver sin linje", skal se ut som følger:

```
# /dev/bcache0
UUID=<uuid skrevet inn av blkid-kommandoen over>
/brick0 xfs defaults 1 2
```

(mountpoint, skal stemme overens med riktig bcache block device, /dev/bcache0 til /brick0 etc)

l. `mount -a`

18. Kommandoer for ZFS on Linux:

a. `zpool create -f -o ashift=12 pool /dev/sdc /dev/sdd /dev/sde /dev/sdf /dev/sdg cache /dev/sdb`

(opprette pool).

b. Dele opp pool i bricks:

- `zfs create pool/brick0`
- `zfs create pool/brick1`
- `zfs create pool/brick2`
- `zfs create pool/brick3`
- `zfs create pool/brick4`

c. Endre mountpoint for bricks

- `zfs set mountpoint=/brick0 pool/brick0`
- `zfs set mountpoint=/brick1 pool/brick1`
- `zfs set mountpoint=/brick2 pool/brick2`
- `zfs set mountpoint=/brick3 pool/brick3`
- `zfs set mountpoint=/brick4 pool/brick4`

d. Skru av deduplication

- `zfs set dedup=off pool`
- `zfs set dedup=off pool/brick0`
- `zfs set dedup=off pool/brick1`
- `zfs set dedup=off pool/brick2`
- `zfs set dedup=off pool/brick3`
- `zfs set dedup=off pool/brick4`

e. Sette på compression

- `zfs set compression=lz4 pool`
- `zfs set compression=lz4 pool/brick0`
- `zfs set compression=lz4 pool/brick1`
- `zfs set compression=lz4 pool/brick2`
- `zfs set compression=lz4 pool/brick3`
- `zfs set compression=lz4 pool/brick4`

f. Skru av access time updates

- `zfs set atime=off pool`
- `zfs set atime=off pool/brick0`
- `zfs set atime=off pool/brick1`
- `zfs set atime=off pool/brick2`
- `zfs set atime=off pool/brick3`

- `zfs set atime=off pool/brick4`

g. Sette 4K block size

- `zfs set recordsize=4K pool`
- `zfs set recordsize=4K pool/brick0`
- `zfs set recordsize=4K pool/brick1`
- `zfs set recordsize=4K pool/brick2`
- `zfs set recordsize=4K pool/brick3`
- `zfs set recordsize=4K pool/brick4`

19. Gjøres på én av frontend serverene(10.0.4.2):

    a. `gluster peer probe <ip til andre frontend server>`

    b. `gluster volume create bsrmgrSAN replica 2 transport tcp 10.0.4.2:/brick0 10.0.4.3:/brick0 10.0.4.2:/brick1 10.0.4.3:/brick1 10.0.4.2:/brick2 10.0.4.3:/brick2 10.0.4.2:/brick3 10.0.4.3:/brick3 10.0.4.2:/brick4 10.0.4.3:/brick4 force`

    (trengte å bruke force fordi /brick-mappene er mount-folders)
    (setter opp distribuering mellom serverne, replica mellom like bricks på serverne, slik som (10.0.4.2)puddy:/brick0 = (10.0.4.3)lippman:/brick0 osv).

20. Testing med fio:

    a. Lage mappe og filer for lagring av resultater ved gjennomføring av fio-tester.

    b. Lage og kjøre skript for fio-testing, vedlagt i appendix B.1.1(XFS/bcache) og B.2.1(ZFS on Linux).

## D.3 Klientmaskin

**Pakker**:

- isc-dhcp-server (dhcp)
- software-properties-common (add-apt-repository)
- screen (samarbeidsjobbing)
- cpu-checker (kvm-ok)
- qemu-kvm (kjøre vm)
- libvirt-bin (administrering av vm)
- virtinst (install vm)
- glusterfs-client (mounte gluster volume på puddy)
- fio (teste ytelsen på lagringsløsningen)

**Trinnvis oppsett**:

1. Sette opp RAID1 på OS-disker i ROM-based utilities.
2. Gjøre network boot.
3. Sette root passord(sudo su, passwd).
4. Endre hostname.
5. Endre keyboard layout:
   a. `dpkg-reconfigure keyboard-configuration`
   b. Dell.
   c. Norsk.
   d. Trykk enter resten.
6. Reboot.
7. Logge inn igjen som root.
8. Sette riktige parametere for network interfaces:
   a. Interface 1 går mot public nett, skal ha dhcp.
   b. Interface 2 går mot lagringsboksene, følgende config:

```
# The secondary network interface
auto eth1
iface eth1 inet static
    address 10.0.4.1
    network 10.0.4.0
    netmask 255.255.255.0
    broadcast 10.0.4.255
    dns-nameservers 128.39.32.2
```

9. `apt-get update`

   (passe på at alle pakkene er up to date ift. repo).
10. `apt-get install -y isc-dhcp-server software-properties-common`
11. `nano /etc/default/isc-dhcp-server`

```
INTERFACES="eth1"
```

12. `nano /etc/dhcp/dhcpd.conf`

```
option routers 10.0.4.1

subnet 10.0.4.0 netmask 255.255.255.0 {
    range 10.0.4.2 10.0.4.3;
    option broadcast-address 10.0.4.255;
    option domain-name-servers 128.39.32.2;

    default-lease-time 3600;
    max-lease-time 7200;

    host puddy {
        hardware ethernet 18:a9:05:3b:3d:68;
        fixed-address 10.0.4.2;
    }

    host lippman {
        hardware ethernet 00:26:55:82:77:24;
        fixed-address 10.0.4.3;
    }
}
```

13. `/etc/init.d/isc-dhcp-server restart`
14. Sett opp routing:
    a. `echo 1 > /proc/sys/net/ipv4/ip_forward`
    b. `iptables -t nat -A POSTROUTING -o em2 -j MASQUERADE`
    c. `iptables -t nat -A POSTROUTING -o em1 -j MASQUERADE`
    d. `iptables -A FORWARD -i em2 -o em1 -m state --state RELATED,ESTABLISHED -j ACCEPT`
    e. `iptables -A FORWARD -i em1 -o em2 -j ACCEPT`
    f. `sysctl -p /etc/sysctl.conf`
    g. `iptables-save > /etc/iptables.rules`
    h. `nano /etc/sysctl.conf`
    i. Uncomment

       `net.ipv4.ip_forward = 1`

       (save and exit).
    j. `sysctl -p /etc/sysctl.conf`
    k. `service networking restart`
15. Pakker som trengs å legge til fra andre steder enn det som står listet i */apt/sources.list*.
    a. `add-apt-repository ppa:semiosis/ubuntu-glusterfs-3.4`
    b. `apt-get update`
16. Innstallere pakker på server.
    a. `apt-get install -y screen nmap cpu-checker qemu-kvm libvirt-bin virtinst glusterfs-client fio`
17. Lage mount folder til GlusterFS.
18. Mounte Vandalay til GlusterFS:
    a. `10.0.4.2:/bsrmgrSAN /gluster-mount-folder glusterfs defaults 0 0`

       (legges til i /etc/fstab, mountes med mount -a).

19. Klargjøre for installering av vm.
    a. `kvm ok`

       (sjekker om hardware-virtualisering er mulig)

    b. `adduser root libvirtd`

       (legger root bruker inn med libvirt rettigheter)

    c. `virsh -c qemu:///system list`

       (evt. virsh list –all, lister alle vm)

20. Testing med fio:
    a. Lage mappe og filer for lagring av resultater ved testingen.
    b. Lage og kjøre skript for fio-testing, vedlagt i appendix B.1.2(xfs/bcache) og B.2.2(zfs on linux).

21. Installere en virtuell maskin:
    a. Laste ned iso for ubuntu server 13.10.
    b. `qemu-img create -f qcow2 disk.qcow2 25G`

       (kjøres i /gluster-mount-folder, oppretter disk image for den virtuelle maskinen).

c. `virt-install --connect qemu:///system -n ubuntu-server -r 2047 --disk path=/gluster-mount-folder/disk.qcow2,format=qcow2 -c ~/iso/ubuntu-server.iso --noautoconsole --graphics vnc,password=foobar,listen=0.0.0.0 --os-type linux --os-variant ubuntusaucy --accelerate --network=network:default --vcpus=2 --hvm`

   (setter igang installasjon av en virtuell maskin).

22. Kjøre fio-tester i vm:

    a. `apt-get update`

       (passe på at alle pakkene er up to date ift. repo).

    b. `apt-get install -y fio screen`

    c. Lage mappe og filer for lagring av resultater ved testingen.

    d. Lage og kjøre skript for fio-testing, vedlagt i appendix B.1.3(xfs/bcache) og B.2.3(zfs on linux).

23. Ta snapshot av vm(husk å holde vm avskrudd når snapshot tas).

    `virsh snapshot-create ubuntu-server`

24. Kjøre tester i "raw" vm(kopier ut test-resultater etter gjennomført test).

25. Gjøre en revert av vm.

    `virsh snapshot-revert --domain ubuntu-server <navn på snapshot>`

26. Kjøre tester i "qcow" vm(kopier ut test-resultater etter gjennomført test).

# Tillegg E

# Møtelogg, gruppemøter

## E.1 07.01.2014

Alle deltakere tilstede.

*Hva er gjort:*
Til nå har gruppedeltakerne lest noe om de ulike løsningene vi skal implementere.

*Hva skal gjøres:*
Grupperegler, Begynne med SLA/REQSPEC. Komme opp med kortnavn.
Tilrettelegge for timeføring. Få opprettet en subversion. Få opp template til Latex. Få opp en prosjektplan / forprosjektrapport / forskningsskisse. Tenke på webside.

*Andre tanker:*
Må komme i kontakt med oppdragsgiver og veileder for å klargjøre hvordan vi skal jobbe i tiden frem til neste møte. SLA og latex template er sett på som mindre viktige for dagens arbeid.

## E.2 09.01.2014

Alle deltakere tilstede.

*Hva er gjort:*
Utkast av del 1 er ferdig. Utkast av del 2 er ferdig. Første del av utkast for 4 er ferdig(4.1). Regnark for timeføring satt opp. Grupperegler satt opp. Kortnavn laget. Har opprettet subversion repo og alle medlemmer har lastet ned TeXStudio.

*Hva skal gjøres:*
Signere Grupperegler. Signere Prosjektplan. Fullføre del 4. Fortsette med de resterende deler av forprosjektrapporten(3, 5 og 6). Avtale med veileder når det passer med ukentlige møter mellom gruppe. veileder og oppdragsgiver.

*Andre tanker:*
Thomas skal fullføre del 4 i utkast til forprosjektrapporten før han går videre. Kim avtaler med veileder, og kan etterpå gjøre del 5 i utkast til forprosjektrapporten. Kent-Marius skal gjøre del 3 i utkast til forprosjektrapporten.

## E.3 10.01.2014

Alle deltakere tilstede.

*Hva er gjort:*
Ukentlig møter er satt opp med Øivind Kolloen (Onsdag kl.11:00). Utkast til rapport kapittel 3 er ferdig. Utkast til rapport kapittel 4 er ferdig. Utkast til rapport kapittel 5 er ferdig.

*Hva skal gjøres:*
Gruppa som helhet skal gå gjennom det som er skrevet som utkast til nå. Etter dette skal det begynnes med kapittel 6 i utkast til forprosjektrapport. Kent-Marius skal skrive om 1.2 og 1.3 i utkast til forprosjektrapport til mandagens møte.

*Andre tanker:*
Hovedtyngden av arbeidet vil gå med på å se over arbeidet som er gjort med forprosjektrapporten, mens det er ingen nødvendighet at de som har skrevet de respektive delene leser over sitt eget arbeid. Tekst som føles feil for leser, markeres med rødt og rettes i plenum i gruppa. Generelle skrivefeil retter den enkelte leser.
Møte mandag den 13.01. 2014 er flyttet fra 14:00 til 09:30.

## E.4 13.01.2014

Alle deltakere tilstede.

*Hva er gjort:*
Forkast til forporsjektrapport er sett gjennom, noe måtte skrives om. Gantt-diagram er påbegynt, men ikke fullført. Del 1.2 og 1.3 er jobbet noe med, men ikke komplett.

*Hva skal gjøres:*
Gjøre ferdig del 1.2 og 1.3 som samlet gruppe, evt. uten Kent-Marius grunnet timeplan. Begynne å sette opp en skisse over hvordan vi ønsker det fysiske miljøet satt opp.

*Andre tanker:*
Arbeid skal ikke gjennomføres etter klokka 20:00, med mindre dette er nødvendig for å få gjennomført arbeidet til avtalt tid. Siden det kommer ikke noe "klokt" ut av vedkommende på den tiden. Gantt-diagram vil forhåpentligvis ferdigstilles etter møtet med oppdragsgiver og veileder onsdag 15.01.
Møt til møtene god tid i forveien, slik at du er klar når møte skal begynne.
Kent-Marius står til ansvar for å gå gjennom del 1.2 og 1.3 senere denne dagen for å se hvordan det er ment at dette skal settes opp.

## E.5 14.01.2014

Alle deltakere tilstede.

*Hva er gjort:*
1.3 i forprosjektrapporten er skrevet ferdig. 1.2 i forprosjektrapporten er skrevet ferdig. Satt opp disker i HWRAID10. Sett etter UPS.

*Hva skal gjøres:*
Sette opp et eksempel på UseCase diagram. Sette opp et eksempel på Gantt-diagram. Dersom det blir ekstra tid, kan innledning til rapporten begynnes med. Beregne strømforbruk på nytt, brukt litt feil server type. Åpne nytt serverskap. Sette inn skinner i det nye skapet(og det gamle?, legge av 4 skinner til det gamle).

*Andre tanker:*
1.2 i forprosjektrapporten må endres noe, usikkert når det gjelder del 1.3.
Erik lurte på om vi kunne flytte møtet fra kl 11:00 til 10:30 er dette aktuelt? Dette går bra.

## E.6 15.01.2014

Alle deltakere tilstede.

*Hva er gjort:*
use case diagram, gantt-diagram, 1.2 i forprosjektrapporten er rettet opp, satt inn i serverskap.

*Hva skal gjøres:*
kravspek skal skrives. forprosjektrapport skal sendes til Øivind når den er ferdig, etter den er lagt i latex. de resterende skinner kan festes i skapet.

*Andre tanker:*
Timeføring skal føres ærlig. Når man jobber med oppgaven skal man prøve å holde fokus på det man skriver, slik at ting "henger på greip". I forhold til UPS, kan være et alternativ å snakke med Jon om det går an å få aggregat til å føre strøm bort til D001, samt alternative UPS-er.

## E.7 16.01.2014

Alle deltakere tilstede.

*Hva er gjort:*
Montering i serverskap. Kravspek er ferdig. Forprosjektrapport er snart ferdig lagt inn i latex. Lagt opp maler for sluttrapport, samt forklaring på de ulike delene.

*Hva skal gjøres:*
Forprosjektrapport skal settes sammen og leveres Øivind. Skinner for de 2 siste serverne skal monteres i skapet. Innledning for sluttrapport kan begynnes på. (Kim tar denne!) Finne ut om PDU / strømfordeler i serverskap kan benyttes. Sette opp mal i Latex for loggføring av møter, da også velge ansvarlig for å føre disse møtene inn i den. Oppdatere firmware på serverne.

*Andre tanker:*
Når en person føler at et "del-kapittel" er komplett kan de da starte å føre dette inn i LateX.

## E.8 17.01.2014

Alle deltakere tilstede.

*Hva er gjort:*
Forprosjekt er sendt til Øyvind. Innledning på sluttrapport er påbegynt. Undersøkt om PDU i skap kan brukes. Mal for loggføring av møter er satt opp i latex. Thomas er valgt som ansvarlig for loggføring av møtene. Det er oppdatert firmware på samtlige servere. Webside er påbegynt.

*Hva skal gjøres:*
Skinner skal monteres i skap(hadde ikke verktøy igår, og de to resterende skinnene var ikke tatt med til HIG). Fortsette med webside. Fortsette med innledning til sluttrapport. Montere de to siste serverne i skapet.

*Andre tanker:*
Med tanke på webside, adresse for lokasjon av siden er tildelt, og websiden bør dermed legges der.

## E.9 20.01.2014

Alle deltakere tilstede.

*Andre tanker:*
Vi som gruppe bestemte oss for å avlyse møtene på mandager, for å kunne jobbe med andre fag.

## E.10 21.01.2014

Alle deltakere tilstede.

*Hva er gjort:*
Skinner for den ene lagringsserveren er montert, de andre er ikke tilgjengelig. Utkast til innledning sluttrapport er nærmest ferdig. Utkast til forprosjektrapport er ferdig. Webside er påbegynt.

*Hva skal gjøres:*
Webside skal fortsettes med. Begynne å lese om zfs, btrfs, bcache, lvm, storage som backend. Hente SSD hos Erik, Partisjonere den ene for OS, Installere OS.

*Andre tanker:*
Hovedtyngden av lesinga vil bli på siste del, altså storage som backend. Dette fordi det er denne delen som gruppa tror det vil være mest informasjon om, og det er dette hele systemet avhenger av.

## E.11 22.01.2014

Alle deltakere tilstede.

*Hva er gjort:*
Webside er jobbet videre med. Sluttrapport er endret noe på.

*Hva skal gjøres:*
Websiden skal fortsettes med. Lese om storage som backend. Lese om optimalt filsystem på backend.

*Andre tanker:*
OS er ikke installert enda, disker var ikke tilgjengelig igår. Det er heller ingen nødvendighet for å få installert OS idag, men kan gjøres hvis det blir lite annet å gjøre.
Samtidig som en er oppe i serverrommet kan den siste lagringsserveren settes inn i skapet.

## E.12 23.01.2014

Alle deltakere tilstede.

*Hva er gjort:*
Webside er jobbet videre med. Siste lagringsserver(HP ProLiant DL 320s) er satt inn i skap. OS er rullet ut på puddy og lippman via den andre bachelorgruppa i drift sin automatiske utrulling. Lest og notert om SAN og xfs.

*Hva skal gjøres:*
Lese om LVM. Jobbe videre med nettside. Lese om SAN, hovedsaklig OS for SAN. Lese om iSCSI protokollen.

*Andre tanker:*
Planen er å jobbe med det som er satt opp så langt, men gjøre mer hvis det blir tid til det.

## E.13 24.01.2014

Alle deltakere tilstede.

*Hva er gjort:*
Lest om LVM og iSCSI protokoll. Lest om SAN OS. Webside er videreutviklet.

*Hva skal gjøres:*
Jobbe videre med webside. Scanne inn prosjektavtale og levere i fronter. Se etter at forprosjekt/prosjektplan er ferdig, deretter lever i fronter. Lese videre på SAN OS.

*Andre tanker:*
Skriv noe om statusrapporter i prosjektplanen før den leveres.

## E.14 28.01.2014

Alle deltakere tilstede.

*Hva er gjort:*
Lest videre om SAN OS. Scannet inn prosjektavtale og levert i fronter. Levert prosjektplan/forprosjekt i fronter. Sjekket om RAID controller i de 2 DL320s'ene kan skiftes med de i DL380G6'ene. Webside er nesten ferdig for nå, trenger bare oppdateringer.

*Hva skal gjøres:*
Switchen skal flyttes. Webside kan optimaliseres. Linke navn på webside til linkedin. Sette opp CSS til webside. Sette av ansvarlig for å skrive statusrapport i slutten av måneden.

*Andre tanker:*
Switchen må flyttes til midt i racket. SAN OS er blitt avgjort til å bli Ubuntu Server 13.10. Vi mangler 1 disk til RAID i den ene 1U serveren (er dette nødvendig?). Webside GUI kan endres, så det blir "finere".

## E.15 29.01.2014

Alle deltakere tilstede.

*Hva er gjort:*
Thomas er ansvarlig for å skrive statusrapporter, disse skal også legges inn i loggen. Kent-Marius har lagd mal for hvordan index sida skal se ut.

*Hva skal gjøres:*
Flytte switch(ble ikke gjort i går, ikke riktig verktøy). Begynne fysisik oppkobling av servere. Installere Ubuntu server 13.10 på begge backend servere. Begynne oppsett av SAN.

*Andre tanker:*
Vi må få flyttet switchen før vi kan begynne å koble i skapet.

## E.16 30.01.2014

Alle deltakere tilstede.

*Hva er gjort:*
Ekstra switch er satt inn i skap for SAN. Webside er jobbet videre med. Ubuntu er installert på backend lagring. Koblet sammen serverne.

*Hva skal gjøres:*
Snakke med it-tjenesten om å få patchet opp et punkt til. Begynne med iSCSI på backend. Fortsette med websiden.

*Andre tanker:*
Det var ikke tid til å begynne med SAN i går, da tiden ble brukt på noe som kom opp under installasjon. Med tanke på iSCSI, serverne må settes inn i internt nettverk først.

## E.17 31.01.2014

Alle deltakere tilstede.

*Hva er gjort:*
Begynt så vidt med config av servere. Kontaktet it-tjenesten for nytt uttak i veggen. Jobbet med webside.

*Hva skal gjøres:*
Lage en skisse av hvordan del 2 i sluttrapporten skal settes opp. Skrive noe på del 2 i sluttrapporten. Webside skal være ferdig i løpet av helga.

*Andre tanker:*
Foreløpig er det ikke noe svar fra IT-tjenesten ang TCP/IP porten på K202, dette medfører noen forsinkelser av config. Hvis ikke noen port er åpnet innen tirsdag, må gruppa se på alternativer ved config.

## E.18 04.02.2014

Alle deltakere tilstede.

*Hva er gjort:*
Laget skisse av hvordan del 2 i sluttrapporten skal skrives. Skrevet litt om noen av de teoretiske begrepene vi har. Satt serverene inn i et nettverk. Webside er jobbet litt videre med.

*Hva skal gjøres:*
Begynne med iSCSI på serverne. Få nettverkstilgang til alle serverne. Presisere en grupperegel.

*Andre tanker:*
Loggføring av møter skjer rett etter møtet er hevet. Kunne hatt bilde av den enkelte gruppedeltaker når musepeker holdes over navn?

## E.19 06.02.2014

Alle deltakere tilstede.

*Hva er gjort:*

Ordnet statisk ip på serverne. Satt opp routing på frontend. Måtte kjøre fresh install på tim. Reorganisert arkitektur. Fjernet puppet fra tim og puddy. Lagt inn bilder på webside.

*Hva skal gjøres:*
Sette opp LVM. Sette opp iSCSI target / initiator. Koble flere lun's mot samme (b)cache.

*Andre tanker:*
Det som ikke er ferdig på webside til nå, har webansvarlig ansvar for å få inn på fritiden. Møter skal loggføres så snart som mulig etter gjennomført møte.

## E.20 07.02.2014

Alle deltakere tilstede.

*Hva er gjort:*
Satt opp lvm på backend. Satt opp iSCSI target på backend (targetcli). Satt opp iSCSI initiator på frontend (open-iscsi).

*Hva skal gjøres:*
Ordne puddy og tim i forhold til iSCSI. Snakke med Jon om iSCSI LUN. Snakke med Erik om en eventuell ruter. Gjøre klart "pakkelister" for serverne vi har. Lage et oppsett hvor vi selv er ansvarlige for å føre inn ukentlige timeantall (mandag-søndag).

*Andre tanker:*
Med tanke på ruter, hvis det blir tilgjengelig, får Kent-Marius ansvar for å ordne config på den (i henhold til avtalt oppsett). Etter samtale med Jon er det nødvendig med reconfigurering av minst 1 av backend storage serverne. Timeføring for uker kan gjøres i "Ark 2" på dokument "Timeføring". Dette kan gjøres som første ting hver tirsdag.

## E.21 11.02.2014

Alle deltakere tilstede.

*Hva er gjort:*
Lastet ned bcache-tools og software-repository-common på puddy og lippman. Skrevet videre på sluttrapport.

*Hva skal gjøres:*
Ordne puddy og tim i forhold til iSCSI. Lippman skal kjøres fresh install på.

*Andre tanker:*
Fresh install på lippman: installer OS, fjern puppet, bytt hostname, sett opp NAT og routing, installer pakker, koble til backend. Verifisere at alt virker etterpå. Gjøres av Kent-Marius.

## E.22 12.02.2014

Alle deltakere tilstede.

*Hva er gjort:*
Fresh install på lippman er nesten ferdig. Lest tilnærmet ferdig om oppsett av xfs og bcache.

*Hva skal gjøres:*
Lese om oppsett av zfs. Skrive mer på sluttrapport.

*Andre tanker:*
Kan hende install på Lippman må gjøres på nytt. Skriving på sluttrapport gjelder SAN, og kanskje få til noe ekstra om btrfs under info om zfs(likheter og ulikheter mellom zfs / btrfs).

## E.23 13.02.2014

Alle deltakere tilstede.

*Hva er gjort:*
Lest litt om KVM. Lest om zfs on linux (setup og teori). Lest om xfs, btrfs, bcache, SAN. Skrevet litt på sluttrapport

*Hva skal gjøres:*
Lese opp igjen teorien som er lest til nå. Les mer om KVM.

*Andre tanker:*
Lesing: vi skal "sirkulere" lesinga, slik at det Kent-Marius har lest skal Kim lese, det Thomas har lest skal Kent-Marius lese, og det Kim har lest skal Thomas lese. Hovedsaklig lese linkene som ligger i "notater lesing"-mappa.

- Leseliste Kim: LUN(størrelse, antall), KVM(qemu, libvirt).

- Leseliste Kent-Marius: KVM(qemu, libvirt), iSCSI, LVM, SAN, UPS, xfs, zfs, btrfs.
- Leseliste Thomas: SAN, bcache, btrfs, FC, FCoE, NAS, LVM.

## E.24 14.02.2014

Alle deltakere tilstede.

*Hva er gjort:*
Leselista fra igår er gått gjennom.

*Hva skal gjøres:*
Kim skal ordne opp i dokumentasjon. Kent-Marius kan begynne å sette opp bibtex og sluttrapport. Thomas skal lese og skrive mer om zfs on linux.

*Andre tanker:*
Sluttrapport er det kun del 1 og 2 som er aktuelt å legge inn. Fremtidige møter blir bare to dager i uka, respektivt tirsdag og torsdag, pluss eventuelle statusmøter på onsdag.

## E.25 23.02.2014

Alle deltakere tilstede.

*Hva er gjort:*
Gluster er satt opp, dette ble feil. Bcache er satt opp, dette ble feil. Xfs er satt opp, må endres siden bcache må settes opp på nytt. KVM er satt opp.

*Hva skal gjøres:*
Sette på hardware virtualisering på vandalay. Glusterfs på ny versjon. bcache på ny, remount. Oppdatere Kent-Marius på arbeidet som er utført iht bcache, gluster og xfs. fio kommandoer bør bli ferdig.

*Andre tanker:*
Kan hende vi slipper å endre noe med xfs, men det ser vi etterhvert.
Fio kommando: fio –name=random-writers –rw=randwrite –size=32m –numjobs=1 –direct=1

## E.26 25.02.2014

Alle deltakere tilstede.

*Hva er gjort:*
Testet mer rundt å opprette VM'er med virt-inst. Fio kommandoene er klare for å testes. Glusterfs satt opp ordentlig. bcache er satt opp ordentlig. Kent-Marius er oppdatert på arbeidet.

*Hva skal gjøres:*
Klare å installere VM'er uten å måtte bruke Xming for å gjøre førstegangsinstallasjon. Må lage et skript som kan håndtere fio-kommandoene. Begynne å teste med fio. Kent-Marius må skjønne det som er gjort idag!

*Andre tanker:*
Skriptet som tar seg av fio kommandoene skal dumpe data til fil, det blir deretter opprettet et skript som henter ut nødvendig data og plotter dette med Gnuplot. Xming må foreløpig brukes for å fullføre installasjon. Img filene er tomme uten å gjøre dette... Ta hensyn til qcow og raw img'er som skal opprettes.

## E.27 27.02.2014

Alle deltakere tilstede.

*Hva er gjort:*
Script til fio-test på xfs er laget og kjørt. Skrevet litt om fio-kommandoene.

*Hva skal gjøres:*
Få vm'ene til å kjøre ordentlig, uten gui installering. Skrive om fio-kommandoene. Ordne opp i dokumentasjon. Skrive statusrapport #2.

*Andre tanker:*
Vi prøver å få til slik at scenario 1(nappe ut strøm) på løsning 1 kan testes imorgen, men da må VM'ene være oppe å kjøre først.

## E.28 04.03.2014

Alle deltakere tilstede.

*Hva er gjort:*
Satt opp igjen logisk volum på tim. Satt opp igjen bcache på puddy. Startet tester på lippman. Satt i 2x 2TB disker i tim.

*Hva skal gjøres:*
Koble opp glusterFS når tester på lippman er ferdig. Teste på puddy så fort alt er oppe igjen. Få opp BibTeX. Bli enige om felles format for skriving i LaTeX.

*Andre tanker:*
Brukt litt lang tid på fio, snakket om i møtet.

## E.29 07.03.2014

Alle deltakere tilstede.

*Hva er gjort:*
Kjørt fio-kommandoene en gang til. Feilsøkt på puddy for å prøve å finne årsaken til feilen. Bibtex. Fio, skrive om kommandoene. Undersøkt hvordan watchdog fungerer. Undersøkt litt på feilmeldingene som watchdog genererte. Skrudd av x11 på puddy og tim. Innstallert vm ferdig ved hjelp av virt-install og tightvnc-client på lokal maskin. Fio-tester på vandalay er kjørt.

*Hva skal gjøres:*
Skru av x11 på de resterende 3 serverne. Sende mail til fio mailing lista. Redefinere fio tester til å håndtere directories. Sette sammen rapporten med målinger for levering til Erik og Øivind. Teste med fio på vandalay, to ganger i vm, og forhåpentligvis på både puddy og lippman en gang til.

*Andre tanker:*
Det ble holdt møte idag i stedet for igår grunnet at igår kunne ikke alle gruppemedlemmene delta. Fio-kommandoene kræsja på samme sted(etter 8 kjøringer på test #2 i scriptet fiotester.sh). Ser ut som grunnen kan være CPU stall på flere av CPU-kjernene i puddy. Pass på at websida oppdateres hver torsdag fremover. Det å teste iops på RAW og qcow2 er endret, slik at dette skal ikke testes på. Man skal heller teste på "live" VM med qcow2 og en snapshot av samme maskin.

## E.30 11.03.2014

Alle deltakere tilstede.

*Hva er gjort:*
Lippman er oppe igjen og vi har fått den ut av recursive loopen som hindret startup. Det er blitt gjort undersøkelser på muligheter på hvorfor iops dropper. Det er blitt lest litt mer på zfs for å være klar for det nye setupet når den tid kommer. Midlertidige test resultater er levert til Erik. Det er tatt valg om å beholde backend sin config slik den er nå, med targetcli.

*Hva skal gjøres:*
Se om det kan være en spesiell grunn for at iops-en droppa så veldig. Kent-Marius skal lese zfsonlinux/faq. Gjøre eventuelle endringer på fio-kommandoene hvis det trengs. Teste videre med fio, gjelder VM-er og Vandalay. Gjøre endringer i sluttrapporten, eller kanskje legge til mer. Se om det finnes noen løsning for problemet som har oppstått på lippman.

*Andre tanker:*
Kent-Marius har en del timer å ta igjen, så det er viktig at han står på fremover. Med tanke på iops-en, kan disken være problemet? Hvis ikke, hvilke parametere i kommandoen kan være problemet? Ellers kan det være noe i bcache som er problemet? Har også tenkt at backend maskinene skal tilkobles slik som idag, med open-iscsi på frontend.

## E.31 13.03.2014

Alle deltakere tilstede.

*Hva er gjort:*
Kjørt tester på nytt på puddy. Gjort feilsøking på lippman.

*Hva skal gjøres:*
Fortsette feilsøking på lippman, må opp igjen (kent og kim). Kjøre tester på vandalay så fort lippman er oppe i "riktig" tilstand. Skrive en del mer i teoridelen av sluttrapporten (thomas). Plotte de nye test-dataene til puddy (kent).

*Andre tanker:*
Mangla noen elementer på teoridelen av sluttrapporten, så det skal gjøres. Forklaring av fio-kommandoene skal legges inn i del to av sluttrapporten sammen med fio. Kan sikkert også prøve å forklare mer utdypende hvis det trengs. Snart også på tide å begynne med del tre av sluttrapporten, og forklare rundt oppsettet vi har brukt for løsning 1. Tenker at resultatene fra første fio-test også kan tas med i sluttrapport(grafer), men forklare hvorfor ytelsen kunne være så dårlig, og hvilke endringer som ble gjort for å bedre ytelsen. Dette kommer ikke før del fire av sluttrapporten.

## E.32 18.03.2014

Alle deltakere tilstede.

*Hva er gjort:*
Det er gjort mer feilsøking på lippman. Tester er kjørt på lippman, puddy og vandalay. Skrevet noe mer teori.

*Hva skal gjøres:*
Sette opp et diagram som viser de ulike "lagene" som testene er kjørt på. Gjøre løpende endringer på gantt-diagram. Kjøre tester på vm. Skrive mer teori i sluttrapporten.

*Andre tanker:*
Fant ingen feil på lippman, så den er oppe å kjører igjen. I forhold til teori, det er begynt å skrive på del 4 i sluttrapporten. Gantt-diagram endres i forhold til hvordan vi faktisk jobber med prosjektet.

## E.33 20.03.2014

Alle deltakere tilstede.

*Hva er gjort:*
Det er kjørt nye tester på puddy og lippmann. Det er rettet opp et cachingsproblem. Grafer er lagt for testdata.

*Hva skal gjøres:*
Nye tester på vandaly - Glustermappa. Nye tester på vandalay VM.

*Andre tanker:*
Test data har forandret seg lite fra første gang, men vi får nå dirty data i cachen, noe som tilsier at den brukes. Og iops spiker mer nå enn hva den gjorde før, dette resulterer også i høyere spikes. Er det noe galt med cachen forsatt ? Fylles den opp og gir lav iops i perioder ?

## E.34 25.03.2014

Alle deltakere tilstede.

*Hva er gjort:*
Fresh install på vandalay, lippman, tim og khomas. Os-disker på lippman er satt i raid1, med de to "gamle" 500GB diskene. Basic setup på vandalay, lippman og khomas. Gjort tester av "raw" vm, snapshot, og nye tester på vandalay. Sluttrapporten er delt inn i flere underdokumenter. Satt opp en to-do liste for lettere å se hva som må/kan gjøres til enhver tid.

*Hva skal gjøres:*
Fresh install på puddy. Basic setup på puddy og tim. Plotte de siste testene som er gjort. Ellers er det bare å følge to-do lista som er laget. Ordne rekkefølgen i to-do lista slik at det er organisert i sammenheng og viktighet.

*Andre tanker:*
Kim jobber remote fra Stavanger grunnet personlige årsaker, han vil jobbe med sluttrapporten og en del teoretiske elementer.
Før fresh install kan gjøres på puddy, må Kent ta med de to 1TB diskene, disk-trays og skruer ned til IT-tjenesten for å få skrudd diskene inn i trays. Deretter må det koordineres med den andre gruppa, slik at det er klart for installering. Gå inn i [F8]-meny før du setter inn disker, og fjern de RAID'ene som ligger der. Pass på at diskene for OS er i tray 1 og 2, og SSD disk er i tray 3.

For basic setup, følg punktene som er listet i filene for de respektive serverne under serverbasert oppsett –> løsning 2 - zfs –> <server-navn>
Det er ikke lenger nødvendig med 2 møter i uka, derfor blir det kun møter på tirsdager fremover, grunnet to-do lista som er satt opp og at medlemmene daglig snakker med hverandre allikevel.
Ser ut for at timeantallet kanskje er litt lavere enn det burde, både totalt og for hver enkelt. Det er derfor ønskelig at fra nå skal timeantallet per uke overstige 30 timer, med mindre det finnes gyldige grunner for at det ikke blir slik.
Fremover bør medlemmene være konsekvente med at de skriver start-tidspunktet på dagsarbeidet før de begynner å arbeide.

## E.35 01.04.2014

Alle deltakere tilstede.

*Hva er gjort:*
Skrevet videre på sluttrapporten. Levert del 1 til Ida og Elisabeth. Levert del 3 til oppdragsgiver og veileder.

*Hva skal gjøres:*

Få opp serverene på en slik måte at gluster kan mountes over ZoL. Ellers følge to-do lista.

*Andre tanker:*
Som nevnt sist uke, må alle stå på fremover, minst 30 timer per uke. Husk på å følge "reglene" som er satt opp for to-do lista.

## E.36 08.04.2014

Alle deltakere tilstede.

*Hva er gjort:*
Zol er satt opp. Skrevet mer på sluttrapport. Retta feil på sluttrapport(fjerna 'da' på overflødige steder).

*Hva skal gjøres:*
Lese gjennom rapport, ordne orddeling. Ordne på zol, jfr. antall bricks. Ellers følg to-do liste.

*Andre tanker:*
Logo som vi skal bruk er logo nummer: 1. Husk å skriv på riktige steder, altså endringer skjer i latex og ny tekst skrives i google docs. Formatet på bilder og figurer skal være .eps.

## E.37 15.04.2014

Alle deltakere tilstede.

*Hva er gjort:*
Zol er nesten satt opp ferdig, mangler kun siste touch.

*Hva skal gjøres:*
Få ferdig zol og begynne tester. Skrive videre om testing. Følg to-do liste.

*Andre tanker:*
$(paaske() != fri());$

## E.38 22.04.2014

Alle deltakere tilstede.

*Hva er gjort:*
Kjørt fio-tester på alle lagene. Skrevet noe på sluttrapporten. Levert del 3 av rapporten til retting.

*Hva skal gjøres:*
Skrive videre i kapittel 5 om testing. Endre kapittel 4 i forhold til det som har blitt gjort tilbakemelding på. Ellers følge to-do liste.

*Andre tanker:*
Det er fortsatt viktig at hver gruppemedlem jobber minst 30 timer per uke. Totalt antall timer per deltaker bør være opp mot 570 timer på hele oppgaven(regner inkludert den tida som skal til for muntlig presentasjon), helst jobbe noe mer enn 30 timer også.

## E.39 29.04.2014

Alle deltakere tilstede.

*Hva er gjort:*
Skrevet en del mer på sluttrapporten. Fio test-resultat-filer er gjort om til pdf og kjørt crop på. Rettet kapittel 2.

*Hva skal gjøres:*
Legge figurene inn i kapittel 5. Rette kapittel 3. Gjøre endringer fra Erik når det kommer. Ellers følge to-do lista.

*Andre tanker:*
Fra nå skal sluttrapporten gjøres ferdig, og dersom vi har tid skal oppsett 1 settes opp igjen slik at vi får testet på nytt (som nevnt i to-do list). Erik skal snakke med zol-ekspert i UiOA for å se hva som er det beste å gjøre ift. zol.

## E.40 06.05.2014

Alle deltakere tilstede.

*Hva er gjort:*
Endret noe på oppsettet i zol, for å se om dette kan endre resultatene som gruppa fikk ved forrige oppsett. Skrevet en del mer på sluttrapporten. Levert foreløpig sluttrapport til oppdragsgiver og

veileder for tilbakemelding. Kjørt tester mot det nye oppsettet i zol. Laget figurer for testresultater.

*Hva skal gjøres:*
Få opp igjen gluster på oppsett 2. Teste på nytt i gluster-mount-folder. Skrive om kapittel 5 slik som erik ønsker. Ellers følge to-do lista.

*Andre tanker:*
Ingen av gruppemedlemmene har oppnådd 30 timer forrige uke av diverse årsaker. Dette er imidlertidig godkjent av lederen, da det har vært lite oppgaver som kan samarbeides om, og to av gruppemedlemmene har vært på ekskursjon til en mulig fremtidig arbeidsgiver.

## E.41 13.05.2014

Alle deltakere tilstede.

*Hva er gjort:*
Skrevet ferdig stort sett hele rapporten, mangler noe småting. Levert rapport til veileder og oppdragsgiver for tilbakemelding. Levert kapittel 4 til retting.

*Hva skal gjøres:*
Rette kapittel 4 når det kommer tilbakemelding på det. Rette kapittel 5 når det kommer tilbakemelding på det. Legge kapittel 6/7 inn i latex, levere til veileder og oppdragsgiver. Levere sluttrapport til Sven Erik Espeland(Evry).

*Andre tanker:*
Ettersom gruppa ligger bra ann, kommer ikke timeantallet stort høyere enn 500-510 timer før rapporten leveres inn.

# Tillegg F

# Møtelogg, statusmøter

## F.1 08.01.2014

Alle deltakere tilstede.

hva skjer hvis noe kræsjer?

To hoveddeler i prosjektet:

- Hva trengs, og hva koster det. Gjelder da UPS i forhold til hvor mye servere som er oppe og kjører. Hvilken UPS trengs.
- Lage gode lagringsbokser. Trenger en datalagringsløsning som er skalerbar, GlusterFS. Redundant, GlusterFS speiler, hjelper litt. Setter opp RAID(10) også. Effektiv lagringskapasitet på 1/4 av total.

Vi må ha 3.5” disker. Det blir et problem med at på de nye så har de bare 2.5”. Vi kan bruke de gamle serverne som benytter seg av 3.5”. Men de er smertefullt trege.

Sette opp SSD-cache mot de gamle 3.5” serverne som backend, og de nye serverne som frontend for bedre ytelse.

Erik har satt opp bcache, se også dokument fått på mail.
- GlusterFS
- xfs
- bcache
- LVM
- HardWare RAID10
- 32 GB RAM

Kræsje qcow-baserte VM-er i ulike scenarier.

## F.2 15.01.2014

Alle deltakere tilstede.

Use Case ser grei ut. ikke følg utviklingsmalen slavisk, bruk heller ingeniør-malen. gantt-diagram må forklares litt, gjelder løsninger og scenarioer. får hjelp av erik på ytelsestester. finne ut av både teoretisk og eksperimentelt.

- bruke de gamle serverne med 3.5” med 2 4TB disker, RAID 1.
- hvor mye SSD cache trengs det for å få til dette effektivt.
- hvordan virker dette eksakt.
- Hvor mye SSD cache i forhold til en treg backend.

## F.3 22.01.2014

Alle deltakere tilstede.

Mottakerene er andre studenter etc fagpersoner.

Os blir på vanlig HDD

Load: har data neste onsdag.

fopen(...,O_SYNC,O_DIRECT), denne typen vil være den værste for scenarioet når strømmen går. her vil data lagres til lagringsbufferen som er sensetiv til strømbrudd.

for å finne ut om cachen er på disk eller på raidet kan man benytte

hdparm -W plassering (/dev/sda)

dersom write cache er på vil man få =1.

fio –name=random-writers –rw=randwrite –size=32m –numjobs=1 –direct=1 — grep iops

write-cache=0 er hva RAID kontrolleren gjør

vi må nesten teste dette . . .

intel s3500 er powerprotected. dette er en mulig vei rundt problemet med cache.

fio kommandoen for å teste reell ytelse vil man få av Erik.

## F.4 29.01.2014

Alle deltakere tilstede, men Kent-Marius var forsinket.

Trenger ikke å gå i detalj om OpenStack. Kjøre ubuntu over en KVM.

teste på ytterpunkter med random read/write og noe midt i mellom.

Se på skyHiG io oppgaven. Denne er den "beste" oppgaven.

Se på storage basics: Part 1. vmtoday

Ingen direkte mal for websiden.

Kommentarer fra oss: Vi vil begynne å sette opp "løsning 1" denne uken fremfor neste uke.

## F.5 05.02.2014

Hvordan er det tenkt å kjøre glusterFS?

- distribusjon
- replica 2

Litt mer informasjon på oppsettet vårt.

- Er det noen spesiell rekkefølge vi bør følge ang. implementeringen av de ulike komponentene ?
- Bør sette opp LVM før iSCSI
- libgfapi
- LVM er bare backend

## F.6 19.02.2014

finne utav tester.
test 100 ganger. tommelfingerregel 30 ganger.
plot dette, middelverdi og standardavvik.

10-15 tester, jfr fio test

3 typer tester:

- VM
  - RAW
  - qcow
- xfs native frontend
- glusterfs frontend

test på VM 2x en med RAW og en med qcow, xfs native frontend , glusterfs frontend

mulig at vi må ha et LVM lag over bcache.

## F.7 05.03.2014

Sende sluttrapport til Øivind.
Bør ha et par som leser gjennom som har peiling på området.
Pass på skrivefeil.
Write cache må passes på.

## F.8 19.03.2014

hpacucli ctrl slot=0 modify dwc=enable
Write caching not supported.
hdparm -W 1 /dev…

lav iops, andre forslag ?

mulig at man må redusere for løkken.
møte uka før påske.

## F.9 07.05.2014

Vi har tidligere spurt om totalt antall timer per deltaker som er forventet ut av antall studiepoeng bacheloroppgaven innebærer(les: ca 570 timer per pers).
Er det medregnet i dette den tiden det vil brukes for å:
forberede muntlig presentasjon
skrive individuelt refleksjonsnotat
lage ferdig plakat og levere inn i fronter

Videre er det spørsmål rundt oppsettet av rapporten.
Er det noe mer som skal være med i rapporten?
(spesielt spørsmål om prosjektavtale skal være inkludert i sluttrapporten, gjelder forøvrig andre aspekter også).

Spørre om det var noe spesielt i sluttrapporten levert sist gang(26.04.14), både ved oppsett, formuleringer og som nevnt ovenfor, eventuelle påkrevde elementer som mangler.

Generelt:
Språk er fortellerstil.

Skriv mer konkrete resultater

Inndeling:
litt kunstig å splitte resultater og diskusjon.
Slå dette sammen til et kapittel.

Sammendrag:
Gå gjennom denne igjen og rette på språket.

Gå gjennom starten på rapporten, og endre på dette slik at dette reflekterer mer mot ytelse, og scenario blir mer underliggende.

1.3 “til ulike fag” : Skriv om denne slik at det klarere kommer frem hva som menes.

diskusjon er karakteravgjørende:
si noe om automatisering når det kommer til self heal kommandoen.
Dette gjelder spesielt i scenario 2. Hvorfor er ikke dette automatisert ?

# Tillegg G

# Statusrapporter

## G.1 Statusrapport 1

# Statusrapport #1 ved kræsjhåndtering(bsrmgr).

Dato: 31.01.2014

**Status for:**

- **Forprosjekt:**
  - Forprosjektet er gjort ferdig til den avtalte tiden.
  - Ble gjort noen endringer etter tilbakemelding fra veileder.
  - Kommer også til å endre noen mindre deler i nærmeste fremtid.

- **Research:**
  - Det er gjort research på mye av teorien som vi trenger i løpet av prosjektet.
  - Blant dette kommer LVM, xfs, btrfs, zfs, SAN OS, iSCSI, san og bcache.
  - Med tanke på SAN OS, er ubuntu server 13.10 valgt.
  - I skrivende stund skal konfigurering av iSCSI begynnes på, samt logisk oppsett av serverne(gjelder statisk ip internt i vårt oppsett).
  - Vi i gruppa regner med at det kommer til å bli mer lesing om hvordan de forskjellige delene(xfs, btrfs, bchache, osv) skal settes opp(config) når disse elementene kommer på tapetet.

- **Sluttrapport:**
  - Det er påbegynt skriving av sluttrapport, hovedsaklig del 1(innledning).
  - Det vil også i nærmeste fremtid bli skrevet i del 2(teori) og del 4(implementering/utførelse).

- **Webside:**
  - Websiden er foreløpig under utvikling.
  - Den skal ferdigstilles i løpet av helga, ansvarlig er webansvarlig.
  - Planen for websiden er at den skal oppdateres jevnlig. Torsdager er fast avsatt til dette. Møtelogg vil også bli ført inn i websiden fortløpende.

- **Problemer:**
  - Det var ved prosjektets begynnelse planlagt å bruke 4TB disker i backend, men pga. RAID-controller er ikke dette mulig. Vil bli brukt 2TB disker i stedet.
  - Ingen problemer ellers å rapportere om.

- **Tidsbruk:**
  - Enkelte av deltakerne har til nå følt at tidsbruket har vært forholdsvis lavt i forhold til andre grupper, både denne våren og tidligere. Grunnlaget for denne følelsen ligger i at gruppa har hatt "lite" å gjøre de første ukene i oppgaveperioden.
  - Vi ligger foreløpig godt an i forhold til Gantt-skjemaet som ble laget ved prosjektets begynnelse.

- **Veiledermøter:**
  - Gruppa foreslår at fra neste møte og utover februar, kan intervallet mellom statusmøter økes til 14 dager. Neste møte vil fortsatt bli avholdt den 05.02.2014.
  - Statusrapporter vil bli skrevet og levert til oppdragsgiver og veileder mot slutten av hver måned.

Thomas R. Kristiansen, dokumentansvarlig - Gjøvik 31.01.2014

---

## G.2 Statusrapport 2

# Statusrapport #2 ved kræsjhåndtering(bsrmgr).

Dato: 28.02.2014

**Status for:**

- **Forprosjekt:**
  - Forprosjektet er helt ferdig, og kommer til å bli lagt til som vedlegg i sluttrapporten.

- **Research:**
  - Det er gjort noe mer research siden sist.
  - Hovedtyngden av dette ligger på det praktiske i forhold til oppgaven, med tanke på configurering av de ulike komponentene.
  - Temaene som er lest om er følgende: zfs, xfs, bcache, glusterFS, fio, iSCSI, DHCP.

- **Sluttrapport:**
  - De første delene av sluttrapporten begynner å ta form.
  - I skrivende stund er del 1(Innledning) og del 2(Teori) er lagt inn i LaTeX-filer, men mangler noen "kosmetiske" endringer.
  - Siden mye av det teoretiske er gjennomført, så er det planlagt at gruppa får til BibTeX i den nærmeste fremtid, slik at dette ikke kommer tett inntil siste innlevering.
  - Gruppa har også planlagt å gå på Lynkurs nr.2, for at gruppa skal kunne optimalisere rapporten.
  - Det er planer om å få kontaktet noen personer som kan lese gjennom rapporten, til slutt og hver enkelt del ettersom de blir ferdige.

- **Webside:**
  - Websida skal være helt ferdig, med tanke på det som kreves av den.

  - Planen for websida er å holde den oppdatert, og hver torsdag skrive en kort oppsummering av hva som skjedde den respektive uka.
  - Det er mulig at utseende på websida blir endret noe, men høyst usannsynlig.

- **Problemer:**
  - Gruppa har til nå hatt lite problemer med oppsettet som er implementert.
  - Utenom dette ble prosjektet "satt på pause" ca en uke grunnet hardware som ble levert litt sent. Det bør sies at det fortsatt var fremgang i prosjektet selv om hardware ikke var til stede.
  - Ellers har gruppa hatt noen problemer når det gjelder opprettelse av virtuelle maskiner, men det ser ut som gruppa har kommet forbi disse problemene.
  - SAN hadde noen problemer med å få gjennopprettet konktakten mellom frontend og backend serverene, hovedsaklig skyldtes dette at fstab ble kjørt før rc.local scriptet og dermed ville ikke bricksene kobles opp, dette er det gjort en hotfix på.
  - Dette blir det laget en ordentlig ordning for så fort alle kommer igang over helga. Tanken er å få oppkobling mot backend lagring å skje i boot-prosessen til frontend. Det er da viktig at backend er skrudd på først.

- **Tidsbruk:**
  - Den ukentlige timebruken har gått noe opp, slik at hvert gruppemedlem ligger over minimumstallet satt i gruppereglene.
  - I forhold til planen som er satt, ligger gruppa foreløpig godt an.
  - Mye av Mars er planlagt å gå til testing, som igjen fører med seg en del skriving. Timeantallet er derfor forventet å øke noe i de nærmeste ukene.

- **Veiledermøter:**
  - Det har gått greit med veiledermøter annenhver uke, så gruppa ønsker å fortsette med dette fremover også.

- **Gruppemøter:**
  - Gruppa bestemte seg for at gruppemøter skal fast avholdes tirsdager og torsdager, med mindre sykdom kommer i vegen for det.
  - I tillegg kommer statusmøtene og eventuelle andre møter som er nødvendige for å holde oversikt over progresjonen og arbeidsplanene.

- **Statusrapporter:**
  - Videre har gruppa tenkt å skrive en statusrapport i slutten av hver måned.
  - Det tilsier at det skrives en sluttrapport for både Mars og April, selv om kun tre er påkrevet.

Thomas R. Kristiansen, dokumentansvarlig - Gjøvik 28.02.2014

---

## G.3 Statusrapport 3

# Statusrapport #3 ved kræsjhåndtering(bsrmgr).

Dato: 28.03.2014

**Status for:**

- **Research:**
  - Siden sist er det gjort en del research rundt hvordan man skal sette opp zfs on linux.
  - I tillegg er det blitt lest om følgende temaer; SSD vs HDD, KVM/QEMU, libvirt, lvm, qcow. Disse temaene er det hovedsaklig lest teori om.
  - Det skal gjøres mere research rundt testing med fio, ettersom gruppa har for lite kunnskap om dette, og at testene klarte å fylle cachen.

- **Fremdrift:**
  - Gruppa har gjort seg ferdig med løsning 1 for nå, og har begynt på løsning 2.
  - Gjort en del tester rundt å opprette zfs-pool.

- **Sluttrapport:**
  - Det er skrevet en del mer på sluttrapporten.
  - I kapittel 2 er det skrevet om noen flere aspekter ved teorien som vi har møtt på i dette prosjektet, slik at alle aspektene av oppgavebeskrivelsen er dekket.
  - I kapittel 3 er det skrevet om fio-testing samt om de ulike metodikkene for administrasjon av virtuelle maskiner vi har vært innom.
  - I kapittel 3 er det satt opp en hardware-oversikt over løsning 1, samt en liste over den programvaren som er brukt på hver av serverne.
  - I kapittel 4 er det skrevet om hvordan man skal opprette virtuelle maskiner ved de 2 forskjellige metodene, samt hvilke metoder som har virket og ikke på vårt system. Dette er vinklet slik at det skal reflektere hvordan det vi har gjort møter kravspek.
  - I kapittel 4 er det skrevet om oppsettet i løsning 1, men gruppa er usikre på om dette skal stå som kapittel 4, eller om det er ønskelig å ha med dette som en appendix.

  - Videre skal det skrives en mer detaljert versjon av kravspesifikasjon i kapittel 2.
  - Gruppa har funnet to som er villige til å lese sluttrapporten ettersom den blir ferdig, men ingen av dem kan noe om fagområdet. Så hvis veileder eller oppdragsgiver har noen forslag er disse velkomne.
  - Det er ønskelig at oppdragsgiver og veileder også kan lese gjennom de enkelte delene av sluttrapporten etterhvert som de er antatt ferdige fra gruppas side. Det er da ønskelig med gode tilbakemeldinger på hva som ikke er som det skal være, så dette kan ordnes opp i.

- **Webside:**
  - Websiden fortsettes å oppdateres som før, med ukentlig oppdatering på torsdager, samt fortløpende oppdatering med møtereferater.

- **Problemer:**
  - Det viste seg at kjøring av tester skulle gi mere problemer enn først antatt, og håpet.
  - I forhold til de virtuelle maskinene, kræsja flere av disse uten at vi visste hvorfor. Problemet her viste seg å være at testene genererte så mye data at disken til den virtuelle maskinen ble fyllt opp, og forårsaket dermed en kræsj.
  - Som en følge av dette måtte det gjøres noen justeringer på testene som kjøres i de virtuelle maskinene.

  - Vi fant ut litt i seneste laget at det hadde blitt gjort en feil ved opprettelse og tilkobling til caching-device i bcache, noe som gjorde at mange timer med tester ikke ga det ønskede resultatet.
  - Dette gjorde at vi fikk noe dårlig tid mot slutten av løsning 1, og ønsker derfor å sette opp igjen denne løsningen etter testing på løsning 2 er ferdig, dersom det er tid for det.
  - Testene genererte så mye data at cachen ble fyllt, så hvordan testene skal revurderes fram mot testing av løsning 2.

- **Tidsbruk:**
  - Det ukentlige antall brukte timer har fortsatt å stige, og det er forventet å stige noe mer.
  - Gruppa har avtalt at fremover er det krav om at hver gruppedeltaker arbeider minst 30 timer hver uke, med mindre det foreligger en god grunn for å ikke kunne gjøre dette.
  - Gruppa har mistet noe arbeidskraft i den foregående uka, ettersom et av gruppemedlemmene måtte reise hjem pga. familiære forhold. Dette gir litt mere krav for arbeidskraft fra de andre to deltakerne. Noe av de "tapte" timene skal prøves å ta inn igjen.

- **Veiledermøter:**
  - Det er ønskelig fra gruppas side å ha et veiledermøte onsdagen i uke 15.
  - Videre er det ønskelig å ha et statusmøte 30 april, for å se om vi ligger godt an i forhold til å få levert rapporten i tide, med alle de delene som skal være med.

- **Gruppemøter:**
  - Det ble denne uken vedtatt at gruppemøtene kun skal avholdes én gang per uke. Grunnen for dette er at medlemmene nesten daglig er i kommuniksjon med hverandre allikevel, noe som medfører at nødvendigheten for møter blir mindre.
  - I tillegg er det opprettet en "to-do" liste med alle arbeidsoppgaver som for øyeblikket trenger å gjøres før den siste innleveringa i mai.

- **Statusrapporter:**
  - Som også nevnt ved sist statusrapport, så vil det bli skrevet én statusrapport til etter denne. Den vil bli skrevet mot slutten av april.

Thomas R. Kristiansen, dokumentansvarlig - Gjøvik 28.03.2014

---

## G.4 Statusrapport 4

# Statusrapport #4 ved kræsjhåndtering(bsrmgr).

Dato: 26.04.2014

**Status for:**

- **Fremdrift:**
  - Gruppen har hatt god fremdrift så langt.
  - Begge oppsettene er ferdig satt opp, samt at alle tester er gjort ferdig.

- **Sluttrapport:**
  - Sluttrapporten begynner å ta form.
  - Frem til nå er kapittel 1 og 2 rettet av en ekstern lærer for å sjekke rettskriving.
  - Kapittel 3 er også levert for retting og forventer tilbakemelding på denne over helga.
  - Endringer påpekt av både oppdragsgiver og veileder er ordnet.
  - Kapitlene 1-4 skal være ferdig, trenger kun å finpusse litt og gjøre litt rettskriving.
  - Kapittel 5 skal nesten være ferdig, og skal leveres til oppdragsgiver og veileder i starten av neste uke.

- **Webside:**
  - Websiden fortsettes å oppdateres som før, med ukentlig oppdatering på torsdager, samt fortløpende oppdatering med møtereferater.

- **Problemer:**
  - Gruppen hadde en del problemer med testing mot zfs on linux.
  - Grunnen for disse problemene lå i at det hadde blitt brukt feil type OS når vi installerte, hvor det var nødvendig å bruke 64-bit fremfor 32-bit system.
  - Det å bruke 32-bit system gjorde at ved skriving av data mot zfs pool, skjedde det en halt innenfor zfs on linux.
  - Det tok en del tid før vi kom på å sjekke dmesg og /var/log/syslog for feilmeldinger mtp. zfs halt.
  - Er ikke et direkte problem, men det ble brukt en del tid på de ulike parameterne som finnes i zfs on linux, som deduplication, compression og recordsize.

- **Tidsbruk:**
  - Er usikkert om hver av gruppemedlemmene kommer til å få 570 timer totalt før innleveringen av rapporten skjer.
  - Det vil komme noen ekstra timer ved forberedelse og planlegging av fremføringen i tillegg etter at rapporten er innlevert.

- **Veiledermøter:**
  - Det er ønskelig fra gruppens side at det blir holdt ett veiledermøte, mulgens første hele uke i mai. Dato ønskes det tilbakemelding om fra veileder.

- **Gruppemøter:**
  - Gruppemøtene har fortsatt å være avholdt fast på tirsdager, og det vil fortsette å være slik frem mot innleveringen den 19.05.

- **Statusrapporter:**
  - Dette vil være den siste statusrapporten som blir skrevet i løpet av denne bachelorperioden.

Thomas R. Kristiansen, dokumentansvarlig - Gjøvik 26.04.2014

---

# Tillegg H

# Prosjektavtale

HØGSKOLEN I GJØVIK

# PROSJEKTAVTALE

mellom Høgskolen i Gjøvik (HiG) (utdanningsinstitusjon),

Erik Hjelmås (oppdragsgiver), og
Thomas Rongved Kristiansen
Kent-Marius Werner
Kim Holmebakken (student(er))

Avtalen angir avtalepartenes plikter vedrørende gjennomføring av prosjektet og rettigheter til anvendelse av de resultater som prosjektet frembringer:

1. Studenten(e) skal gjennomføre prosjektet i perioden fra 07.01.2014 til 15.05.2014.

   Studentene skal i denne perioden følge en oppsatt fremdriftsplan der HiG yter veiledning. Oppdragsgiver yter avtalt prosjektbistand til fastsatte tider. Oppdragsgiver stiller til rådighet kunnskap og materiale som er nødvendig for å få gjennomført prosjektet. Det forutsettes at de gitte problemstillinger det arbeides med er aktuelle og på et nivå tilpasset studentenes faglige kunnskaper. Oppdragsgiver plikter på forespørsel fra HiG å gi en vurdering av prosjektet vederlagsfritt.

2. Kostnadene ved gjennomføringen av prosjektet dekkes på følgende måte:
   - Oppdragsgiver dekker selv gjennomføring av prosjektet når det gjelder f.eks. materiell, telefon/fax, reiser og nødvendig overnatting på steder langt fra HiG. Studentene dekker utgifter for trykking og ferdigstillelse av den skriftlige besvarelsen vedrørende prosjektet.
   - Eiendomsretten til eventuell prototyp tilfaller den som har betalt komponenter og materiell mv. som er brukt til prototypen. Dersom det er nødvendig med større og/eller spesielle investeringer for å få gjennomført prosjektet, må det gjøres en egen avtale mellom partene om eventuell kostnadsfordeling og eiendomsrett.

3. HiG står ikke som garantist for at det oppdragsgiver har bestilt fungerer etter hensikten, ei heller at prosjektet blir fullført. Prosjektet må anses som en eksamensrelatert oppgave som blir bedømt av faglærer/veileder og sensor. Likevel er det en forpliktelse for utøverne av prosjektet å fullføre dette til avtalte spesifikasjoner, funksjonsnivå og tider.

4. Den totale besvarelsen med tegninger, modeller og apparatur så vel som programlisting, kildekode, disketter, taper mv. som inngår som del av eller vedlegg til besvarelsen, gis det en kopi av til HiG, som vederlagsfritt kan benyttes til undervisnings- og forskningsformål. Besvarelsen, eller vedlegg til den, må ikke nyttes av HiG til andre formål, og ikke overlates til utenforstående uten etter avtale med de øvrige parter i denne avtalen. Dette gjelder også firmaer hvor ansatte ved HiG og/eller studenter har interesser.

   Besvarelser med karakter C eller bedre registreres og plasseres i skolens bibliotek. Det legges også ut en elektronisk prosjektbesvarelse uten vedlegg på bibliotekets del av skolens internett-sider. Dette avhenger av at studentene skriver under på en egen avtale hvor de gir biblioteket tillatelse til at deres hovedprosjekt blir gjort tilgjengelig i papir og nettutgave (jfr. Lov om opphavsrett). Oppdragsgiver og veileder godtar slik

offentliggjøring når de signerer denne prosjektavtalen, og må evt. gi skriftlig melding til studenter og dekan om de i løpet av prosjektet endrer syn på slik offentliggjøring.

5. Besvarelsens spesifikasjoner og resultat kan anvendes i oppdragsgivers egen virksomhet. Gjør studenten(e) i sin besvarelse, eller under arbeidet med den, en patentbar oppfinnelse, gjelder i forholdet mellom oppdragsgiver og student(er) bestemmelsene i Lov om retten til oppfinnelser av 17. april 1970, §§ 4-10.

6. Ut over den offentliggjøring som er nevnt i punkt 4 har studenten(e) ikke rett til å publisere sin besvarelse, det være seg helt eller delvis eller som del i annet arbeide, uten samtykke fra oppdragsgiver. Tilsvarende samtykke må foreligge i forholdet mellom student(er) og faglærer/veileder for det materialet som faglærer/veileder stiller til disposisjon.

7. Studenten(e) leverer oppgavebesvarelsen med vedlegg (pdf) i Fronter. I tillegg leveres et eksemplar til oppdragsgiver.

8. Denne avtalen utferdiges med et eksemplar til hver av partene. På vegne av HiG er det dekan/prodekan som godkjenner avtalen.

9. I det enkelte tilfelle kan det inngås egen avtale mellom oppdragsgiver, student(er) og HiG som nærmere regulerer forhold vedrørende bl.a. eiendomsrett, videre bruk, konfidensialitet, kostnadsdekning og økonomisk utnyttelse av resultatene.

   Dersom oppdragsgiver og student(er) ønsker en videre eller ny avtale, skjer dette uten HiG som partner.

10. Når HiG også opptrer som oppdragsgiver trer HiG inn i kontrakten både som utdanningsinstitusjon og som oppdragsgiver.

11. Eventuell uenighet vedrørende forståelse av denne avtale løses ved forhandlinger avtalepartene i mellom. Dersom det ikke oppnås enighet, er partene enige om at tvisten løses av voldgift, etter bestemmelsene i tvistemålsloven av 13.8.1915 nr. 6, kapittel 32.

12. Deltakende personer ved prosjektgjennomføringen:

HiGs veileder (navn): Øivind Gulleen

Oppdragsgivers kontaktperson (navn): Erik Hjelmås

Student(er) (signatur): Thomas R. Kristiansen dato 15/1-2014

Jent-Marius Werner dato 15/1-2014

Kim Holmubakken dato 15/01-14

_____ dato _____

Oppdragsgiver (signatur): _____ dato 15/1-2014

IMT Dekan/prodekan (signatur): _____ dato _____

*Versjon Januar 2011hb*

# Tillegg I

# Timelister

Dette regnarket er mest for å få en oversikt over hvor mye hver arbeider med denne oppgaven, slik at vi kan opprettholde at alle jobber omtrent like mye.

| | Thomas | | | Kim | | | Kent-Marius | | |
|---|---|---|---|---|---|---|---|---|---|
| Dato | Tidspunkt(fra - til) | Antall timer | Beskrivelse | Tidspunkt(fra - til) | Antall timer | Beskrivelse | Tidspunkt(fra - til) | Antall timer | Beskrivelse |
| 07.01.2014 | 10:00 - 12:00 | 2,0 | grupperegler, timeliste, diverse | 10:00 - 12:00 | 2,0 | grupperegler, timeliste, diverse | 10:00 - 12:00 | 2,0 | grupperegler, timeliste, diverse |
| 08.01.2014 | 09:00 - 11:00 | 2,0 | Lynkurs nr.1 | 09:00 - 11:00 | 2,0 | Lynkurs nr.1 | 09:00 - 11:00 | 2,0 | Lynkurs nr.1 |
| | 11:00 - 12:00 | 1,0 | Møte med Erik | 11:00 - 12:00 | 1,0 | Møte med Erik | 11:00 - 12:00 | 1,0 | Møte med Erik |
| | 12:00 - 14:00 | 2,0 | Utkast kapittel 4 forprosjektrapport | 12:00 - 14:00 | 2,0 | Utkast kapittel 2 | 12:00 - 14:00 | 2,0 | Utkast kapittel 1 |
| | | | | | | | 23:00 - 01:00 | 2,0 | Utkast kapittel 1 |
| 09.01.2014 | 11:00 - 13:00 | 2,0 | Utkast kapittel 4 og 5 forprosjektrapport | 11:00 - 13:00 | 2,0 | Utkast kapittel 5 forprosjektrapport | 10:00 - 14:00 | 4,0 | Utkast kapittel 1 og Utkast kapittel 3 |
| 10.01.2014 | 10:00 - 14:00 | 4,0 | Utkast til forprosjekt kapittel 1 - 5 | 10:00 - 14:00 | 4,0 | Utkast til forprosjekt kapittel 1 - 5 | 10:00 - 14:00 | 4,0 | Utkast til forprosjekt kapittel 1 - 5 |
| 12.01.2014 | | | | | | | 23:00 - 00:00 | 1,0 | Utkast kapittel 1 |
| 13.01.2014 | 09:30 - 13:30 | 4,0 | Utkast til forprosjekt, ups, server oppsett | 09:30 - 13:30 | 4,0 | Utkast til forprosjekt, ups, server oppsett | 09:30 - 10:00 | 0,5 | Møte, utkast kapittel 1 |
| | | | | 13:30 - 14:00 | 0,5 | Satt HDD i raid, samt upgrade til FW | 17:30 - 19:30 | 2,0 | Utkast kapittel 1 |
| 14.01.2014 | 10:30 - 15:30 | 5,0 | Montering i rack, usecase | 10:30 - 16:30 | 6,0 | Gantt-skjema, Monetring i rack | 11:00 - 16:30 | 5,5 | Utkast kapittel 1 og use-case |
| 15.01.2014 | 10:30 - 16:30 | 6,0 | Montering i rack, forprosjektrapport, møte | 10:30 - 16:30 | 6,0 | Montering i rack, forprosjektrapport, møte | 10:30 - 16:30 | 6,0 | Montering i rack, forprosjektrapport, møte |
| 16.01.2014 | 11:00 - 16:30 | 5,5 | Sluttrapport, mal for loggføring av møter | 11:00 - 16:30 | 5,5 | Fullførte forprosjekt, startet på innleding i sluttrapport | 11:00 - 16:30 | 5,5 | Oppdatert firmware på serverne. Jobbet med nettside. |
| 17.01.2014 | 11:00 - 13:00 | 2,0 | Montering i rackskap, sluttrapport, loggført møte | 11:00 - 13:00 | 2,0 | Jobbet med innledning sluttrapport | 11:00 - 15:00 | 4,0 | Jobbet med nettside. |
| 21.01.2014 | 08:00 - 13:30 | 5,5 | Innledning sluttrapport, forprosjektrapport, loggført møte | 08:00 - 13:30 | 5,5 | Forprosjekt rapport, sluttrapport, LateX | 11:00 - 15:00 | 4,0 | Jobbet med nettside. |
| 22.01.2014 | 10:30 - 15:30 | 5,0 | Lesing, serverskap | 10:30 - 15:30 | 5,0 | lesing & notering | 10:30 - 15:30 | 5,0 | Jobbet med nettside. |
| 23.01.2014 | 11:00 - 15:00 | 4,0 | Lesing on iSCSI, LVM | 11:00 - 15:00 | 4,0 | Lesing om SAN OS | 11:00 - 15:00 | 4,0 | Jobbet med nettside. |
| 24.01.2014 | 10:30 - 14:30 | 4,0 | Forprosjektrapport, prosjektavtale | 10:30 - 14:30 | 4,0 | Lesing om Openfiler | 11:00 - 13:30 | 2,5 | Jobbet med nettside. |
| 28.01.2014 | 08:00 - 13:30 | 5,5 | Sjekka om raid controller kan byttes | 08:00 - 13:30 | 5,5 | Sjekket Openfiler, "omøblert racket", sjekker raid controller | 11:00 - 16:30 | 5,5 | Jobbet med nettside. |
| 29.01.2014 | 10:30 - 16:30 | 6,0 | Installer ubuntu server på backend, møte, endret noe i forprosjekt og sluttrapport | 10:30 - 16:30 | 6,0 | Installert ubuntu på backend, møte, lest mer på oppsett av SAN og iSCSI på linux, nytt oppsatt RAID | 11:00 - 16:30 | 5,5 | Jobbet med nettside. |
| 30.01.2014 | 13:30 - 18:00 | 4,5 | Statusrapport, litt config av backend | 13:30 - 18:00 | 4,5 | konfig på servere (keyboard og NIC) | 13:00 - 19:30 | 6,5 | Jobbet med nettside. Oppdatert møte loggene på websida. |
| 31.01.2014 | 10:00 - 10:30 | 0,5 | Gruppemøte, statusrapport | 10:00 - 10:30 | 0,5 | Gruppemøte | 10:00 - 11:30 | 1,5 | Jobbet med websida. |
| 01.02.2014 | | | | 08:00 - 09:00 | 1,0 | Skrev litt om zfs, leste mer om zfs | | | |
| 02.02.2014 | 11:30 - 12:30 | 1,0 | Skrevet om lvm og iscsi, lest litt mer | | | | | | |
| 04.02.2014 | 09:00 - 15:30 | 6,5 | DHCP, iSCSI | 09:00 - 16:00 | 7,0 | DHCP, iSCSI | 12:00 - 16:00 | 4,0 | Fått opp bilder på websida, oppdatert den |
| 05.02.2014 | 09:00 - 15:30 | 6,5 | Omdesignet arkitektur, et par clean installs, lagt inn lvm på khomas | 09:00 - 15:30 | 6,5 | Omdesignet arkitektur, clean install på tim, puddy. LVM, lest og installert pakke på khomas | 09:00 - 15:30 | 6,5 | Lest og Skrivd litt om LUN |
| 06.02.2014 | 11:00 - 18:00 | 7,0 | LUN, lvm, iscsi | 11:00 - 18:00 | 7,0 | LUN, lvm, iscsi, research | 11:00 - 18:00 | 7,0 | Configuering av LUN på Khomas og Tim |
| 07.02.2014 | 09:00 - 15:30 | 6,5 | LUN, LVM, Vandalay oppsett, nettverk.sh | 09:00 - 15:30 | 6,5 | LUN, LVM, Vandalay oppsett | 09:00 - 15:30 | 6,5 | Vandalay, Lippman , Puddy oppsett |
| 10.02.2014 | | | | | | | 11:00 - 12:00 | 1,0 | Lest opp på dokumentasjonen |
| 11.02.2014 | 07:30 - 13:30 | 6,0 | Bcache, sluttrapport, apt-get repo | 07:30 - 13:30 | 6,0 | bcache teori, sluttrapport, | 11:00 - 19:30 | 8,5 | Oppsett av OS, NAT, fjernet puppet, byttet hostname og routing Ferdig |
| 12.02.2014 | 08:00 - 13:30 | 5,5 | Lesing om zfs, | 08:00 - 13:30 | 5,5 | lesing san, btrfs | 08:00 - 13:30 | 5,5 | Oppsett av OS, NAT, fjernet puppet, byttet hostname, routing og oppkobling mot backend ferdig på lippman |
| 13.02.2014 | 10:00 - 10:30 | 0,5 | Møte | 10:00 - 10:30 | 0,5 | Møte | 10:00 - 10:30 | 0,5 | Møte |
| | 12:00 - 14:00 | 2,0 | Lesing | 12:00 - 14:00 | 2,0 | Lesning | 13:00 - 14:00 | 1,0 | Lesning |
| 14.02.2014 | 10:00 - 10:30 | 0,5 | Møte | 10:00 - 10:30 | 0,5 | Møte | 09:00 - 14:30 | 5,5 | Lese, møte og noe Tex innføring |
| | | | | 12:30 - 13:30 | 1,0 | skrevet om på dokumentasjon | | | |
| 15.02.2014 | 10:30 - 12:30 | 2,0 | Satt opp enkeltvis oppsett av serverne vi har | | | | | | |
| 16.02.2014 | 10:30 - 12:30 | 2,0 | Satt opp enkeltvis oppsett av serverne vi har, skrevet litt mer om zfs | 08:00 - 09:00 | 1,0 | skrevet på dokumentasjon | 15:00 - 16:00 | 1,0 | Føre inn i Tex Del 1 |
| | 16:00 - 17:30 | 1,5 | Skrevet om zfs | | | | | | |
| 17.02.2014 | | | | | | | 09:00 - 13:00 | 4,0 | Føre inn i Tex Del 1 |
| 18.02.2014 | 09:00 - 09:30 | 0,5 | Sett over teori | 09:00 - 09:30 | 0,5 | Sett over teori | 11:00 - 15:00 | 4,0 | Føre inn i Tex Del 1 og 2 |
| 19.02.2014 | 10:30 - 12:30 | 2,0 | Fio, møte | 10:30 - 12:30 | 2,0 | Fio, møte | 10:30 - 16:00 | 5,5 | Installert KVM og virtinst på Vandalay |
| | 15:30 - 16:00 | 0,5 | Fio | 15:30 - 16:00 | 0,5 | Fio | | | |
| | 17:00 - 18:00 | 1,0 | Fio | | | | | | |
| 20.02.2014 | 09:00 - 09:30 | 0,5 | Fio og dokumetering | 09:00 - 09:30 | 0,5 | Fio og dokumetering | | | |
| | 10:30 - 20:00 | 9,5 | Bcache, glusterfs, kvm, xfs, montert ssd | 10:30 - 20:00 | 9,5 | Bcache, glusterfs, kvm, xfs, montert ssd | | | |
| 21.02.2014 | 10:00 - 11:00 | 1,0 | Snakke med Erik | | | | 09:00 - 11:00 | 2,0 | Feilsøke virt-install |
| 23.02.2014 | 08:30 - 09:00 | 0,5 | Fio | 08:30 - 09:00 | 0,5 | Fio | | | |
| | 10:30 - 18:30 | 8,0 | bcache, glusterfs, fio, installert vm | 10:30 - 18:30 | 8,0 | bcache, glusterfs, fio, installert vm | 10:30 - 18:30 | 8,0 | bcache, glusterfs, fio, installert vm |
| | 17:00 - 18:00 | 1,0 | Fio | | | | | | |
| 25.02.2014 | 08:00 - 13:30 | 5,5 | Fio, møte, kvm | 08:00 - 13:30 | 5,5 | Fio, møte, kvm | 11:00 - 11:30 | 0,5 | Møte |
| | | | | | | | 14:00 - 16:00 | 2,0 | Satt meg inn i kommandolinjene og lest om GNUplot |
| 26.02.2014 | 09:00 - 17:00 | 8,0 | Fio, scripts | 09:00 - 17:00 | 8,0 | Fio, scripts | 11:00 - 13:00 | 2,0 | Fio |
| 27.02.2014 | 10:30 - 18:30 | 8,0 | Prøvd å sette opp vm | 10:30 - 18:30 | 8,0 | Prøvd å sette opp vm | 10:30 - 12:30 | 2,0 | Fio |
| | | | | | | | 13:30 - 17:00 | 3,5 | Fio |
| 28.02.2014 | 14:30 - 18:00 | 3,5 | LaTeX, statusrapport 2 | 10:00 - 15:00 | 5,0 | puddy bcache problemer, iscsi automount research | | | |
| 01.03.2014 | | | | 06:00 - 10:00 | 4,0 | fio, script lippman | | | |
| 02.03.2014 | 10:30 - 12:30 | 2,0 | Serverbasert oppsett | | | | 11:00 - 19:00 | 8,0 | Fio |
| | 15:00 - 16:00 | 1,0 | Serverbasert oppsett | | | | | | |
| 03.03.2014 | 18:00 - 20:00 | 2,0 | Sjekket og fikset på puddy og lippman | 18:00 - 20:00 | 2,0 | Sjekket og fikset på puddy og lippman | 10:30 - 12:00 | 1,5 | Koblet serverne til nettet, fio |
| 04.03.2014 | 08:00 - 13:30 | 5,5 | Fikset puddy og tim(bcache) | 07:30 - 13:30 | 6,0 | Fikset puddy og tim(bcache) | 11:00 - 15:00 | 4,0 | Fio kommando beskrivelse |
| 05.03.2014 | 10:30 - 15:00 | 4,5 | Lynkurs, bibtex, statusmøte, fio | 10:30 - 15:00 | 4,5 | Lynkurs, bibtex, statusmøte, fio | 10:30 - 16:30 | 6,0 | fio, bibtex, lynkurs, møte |
| | 15:00 - 19:00 | 4,0 | Fio, gluster, dokumentasjon | 15:00 - 19:00 | 4,0 | Fio, gluster, dokumentasjon | | | |
| 06.03.2014 | 08:00 - 14:30 | 6,5 | Fio, gluster, dokumentasjon, feilsøking | 08:00 - 14:30 | 6,5 | gluster, error code research, forum, IRC gluster, fio testing eksempler på gluster folder, feilsøking | 08:00 - 14:30 | 6,5 | Bibtex, fio |
| | 14:30 - 15:30 | 1,0 | qcow vm | 14:30 - 15:30 | 1,0 | qcow vm | | | |
| 07.03.2014 | 10:00 - 18:00 | 8,0 | Fio, feilsøkt puddy, innstallert vm, | 10:00 - 18:00 | 8,0 | watchdog, syslog, møte, møte med erik, mailing lists, research på feilkode | 10:00 - 18:00 | 8,0 | Fio måle resultater, Møte, Møte med Erik |
| 08.03.2014 | 12:00 - 13:00 | 1,0 | Fio, vm | 12:00 - 13:00 | 1,0 | Fio, vm | 15:00 - 16:00 | 1,0 | Skrivd inn fio måle resultater. Sett litt på skripting |
| | 14:00 - 18:00 | 4,0 | Fio, vm | 14:00 - 18:00 | 4,0 | Fio, vm | | | |
| 09.03.2014 | 18:00 - 19:00 | 1,0 | Fio, vm | 07:30 - 09:30 | 2,0 | fio, vm | 17:00 - 19:00 | 2,0 | Fått opp mer på exel dokumentet i forhold til fio |
| 10.03.2014 | 10:30 - 17:00 | 6,5 | Vm, reinstall av vandalay | 10:30 - 17:00 | 6,5 | Vm, reinstall av vandalay | 08:00 - 19:00 | 11,0 | Ført inn første testene fra Vandalay og Lippman |
| | 17:30 - 18:30 | 1,0 | Fio-tester på puddy og lippman | | | | | | |
| 11.03.2014 | 08:00 - 13:30 | 5,5 | Fått opp igjen lippman, research | 08:00 - 13:30 | 5,5 | Fått opp igjen lippman, research | 11:00 - 18:30 | 7,5 | Lest om Fio drop, ZFS faq |
| | 18:00 - 18:30 | 0,5 | Fio-tester på puddy | 16:00 - 18:00 | 2,0 | VM, scrip, fio jobfiler | | | |
| 12.03.2014 | 09:30 - 13:30 | 4,0 | Fio-tester på puddy | 09:30 - 13:30 | 4,0 | Lippmann oppsett | 09:30 - 18:00 | 8,5 | ZFS faq, server problems |
| 13.03.2014 | 10:00 - 17:00 | 7,0 | Lippmann, møte, tester puddy, libvirt, glusterfs, caching | 10:00 - 17:00 | 7,0 | Lippmann, hdd vs ssd, møte | 10:30 - 11:00 | 0,5 | Møte |
| | 17:00 - 17:30 | 0,5 | Gått over nye resultater på puddy | | | | | | |
| 14.03.2014 | 10:30 - 17:30 | 7,0 | Skrive på sluttrapport, ordnet rekkefølge på teori, gluster remount | 10:30 - 17:30 | 7,0 | snapshot, DWC, routing, feilsøking lippman | 11:00 - 17:30 | 6,5 | snapshot, DWC, routing, feilsøking lippman |
| 15.03.2014 | | | | | | | 09:00 - 18:00 | 9,0 | Tester lippman og Puddy, fått lagd ett standard excel dokument for fio |
| 16.03.2014 | 13:00 - 13:30 | 0,5 | Sjekket vandalay, puddy og lippman | | | | 13:30 - 16:30 | 3,0 | Startet test på Vandalay, hentet ut test resultatene på puddy og lippman |
| 17.03.2014 | | | | 09:00 - 09:30 | 0,5 | satte igang tester på VM | | | |
| 18.03.2014 | 07:30 - 13:30 | 6,0 | Skriving av teori | 07:30 - 13:30 | 6,0 | Ordnet med vm, tester, skriving teori. | 11:00 - 19:00 | 8,0 | Ført testene fra vandalay, puddy og lippman in i excel |
| | | | | 17:00 - 18:30 | 1,5 | Tester VM og 8k på vandalay | | | |
| 19.03.2014 | 08:00 - 20:00 | 12,0 | Feilsøking i bcache, satt opp bcache igjen, startet nye tester | 08:00 - 20:00 | 12,0 | Litt feilsøking i gluster, satt opp bcache på ny, skrevet teori | 08:00 - 20:00 | 12,0 | Fio testing, feilsøking bcache, |
| 20.03.2014 | 10:00 - 12:00 | 2,0 | Feilsøking lippman, tester vandalay | 10:00 - 12:00 | 2,0 | Feilsøking lippman, tester vandalay | 09:00 - 11:00 | 2,0 | Ført inn tester puddy og lippman |
| | 14:00 - 18:00 | 4,0 | Feilsøking vandalay, satt igang nye tester, lesing | 14:30 - 15:30 | 1,0 | Feilsøking lippman | | | |
| 21.03.2014 | 11:00 - 15:00 | 4,0 | Testing scenario 1 & 2 | 11:00 - 15:00 | 4,0 | Testing scenario 1 & 2 | 09:00 - 15:30 | 6,5 | testing scenario 1 og 2, fått opp igjen puddy |
| | | | | 16:00 - 18:00 | 2,0 | Fio script, seq. cutoff script, test | | | |
| 22.03.2014 | | | | 06:00 - 07:00 | 1,0 | Sett over resultat, prøvd å få opp puddy og lippman remote | | | |
| 23.03.2014 | 11:00 - 12:30 | 1,5 | Møtelogging, serverbasert oppsett | 16:00 - 18:00 | 2,0 | Fikk opp igjen puddy og lippman, startet fio testing på gluster 16:51 | | | |
| | 20:00 - 21:00 | 1,0 | Testing av "raw" vm | | | | | | |
| 24.03.2014 | 07:45 - 09:15 | 1,5 | Fiks av vm | 05:30 - 06:00 | 0,5 | Reinstall VM | 10:00 - 16:30 | 6,5 | fått opp server, iops ført inn, virt testing |
| | 09:30 - 10:00 | 0,5 | Fiks av vm | 06:30 - 07:30 | 1,0 | Fiotester på VM | 21:00 - 21:30 | 0,5 | iops test |
| | 10:30 - 11:30 | 1,0 | Satt opp vm, testing vm | | | | | | |
| | 13:45 - 15:45 | 2,0 | Satt opp vm, nye tester | | | | | | |
| | 17:00 - 20:00 | 3,0 | Henta ut test-data, satt igang nye tester, satt opp en oversikt over manglende temaer | | | | | | |

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| | 22:30 - 23:00 | 0,5 | Hentet ut test-data, satt igang test på vandalay | | | | | | |
| 25.03.2014 | 08:00 - 14:00 | 6,0 | Fresh install av servere, oppsett av vandalay, lippman og khomas, møte | 08:00 - 14:00 | 6,0 | Sluttrapport og møte | 11:00 - 17:30 | 6,5 | Møte, fresh install på puddy |
| | 17:30 - 18:30 | 1,0 | Organisert to-do lista | | | | | | |
| 26.03.2014 | 08:00 - 14:00 | 6,0 | Setup av lippman og khomas, skriving i sluttrapport | 09:30 - 11:30 | 2,0 | Latex --> sluttrapport | 08:00 - 14:00 | 6,0 | Setup av puddy og tim, lesing av korrektur |
| | 15:00 - 17:00 | 2,0 | Skriving av sluttrapport | | | | | | |
| 27.03.2014 | 08:00 - 18:00 | 10,0 | Skriving av sluttrapport, lagt inn i latex | | | | 10:30 - 20:00 | 9,5 | Sendt mail til ida, øyvind og erik. Ført inn de siste iops testene, lagd pool på puddy og testet |
| 28.03.2014 | 12:15 - 14:45 | 2,5 | Skriving av statusrapport | | | | 10:30 - 12:00 | 1,5 | ZFs oppsett |
| | | | | | | | 13:00 - 14:30 | 1,5 | ZFs oppsett |
| 29.03.2014 | | | | | | | 16:00 - 19:30 | 3,5 | ZFs oppsett på puddy og lippman |
| 30.03.2014 | | | | | | | 19:00 - 19:30 | 0,5 | ZFs testet om poolene fungerer |
| 31.03.2014 | | | | 10:45 - 15:45 | 5,0 | Latex --> sluttrapport | | | |
| 01.04.2014 | 08:00 - 13:30 | 5,5 | Skriving av sluttrapport, møte | 08:00 - 13:30 | 5,5 | Skriving av sluttrapport, møte | 14:30 - 19:30 | 5,0 | ZFs og gluster |
| 02.04.2014 | 08:30 - 17:30 | 9,0 | Ordning av referanser | 10:00 - 17:00 | 7,0 | Slutrapport | 09:00 - 19:00 | 10,0 | Fätt opp gluster på lippman, puddy og vandalay |
| 03.04.2014 | 09:00 - 14:30 | 5,5 | Ordnet med zfs, gluster og referanser | 09:00 - 14:30 | 5,5 | Ordnet noe rettskriving sluttrapport | 09:30 - 18:30 | 9,0 | Gluster, ZFs og VM oppsett |
| | 13:30 - 17:00 | 3,5 | Fikset "skrivefeil" i sluttrapport, begynt på figure-format og eps (vektorgrafikk) | 13:30 - 15:00 | 1,5 | sluttrapport, rettskrivning | | | |
| 04.04.2014 | 11:30 - 18:30 | 7,0 | Skriving av sluttrapport, fiksa litt i latex. | 11:30 - 18:30 | 7,0 | sluttrapport, rettskrivning, .eps i latex | 12:30 - 16:00 | 3,5 | ZFs test lesing |
| 05.04.2014 | 10:00 - 15:30 | 5,5 | Ordnet figurer, referert til figurer i latex, lagt inn kapittel 4 i latex | 10:00 - 15:30 | 5,5 | Sluttrapport, HA storage, SSD, kravspec | 10:00 - 18:00 | 8,0 | IOPS gjennomsnitt over alle lagd. Fio testing on zfs |
| 06.04.2014 | 09:00 - 14:30 | 5,5 | Ordnet formattering i latex, skrevet om de emnene som manglet | 09:00 - 15:30 | 6,5 | Logo, skring i sluttrapport | 14:00 - 18:00 | 4,0 | Fio innføring |
| 07.04.2014 | | | | | | | 10:30 - 16:00 | 5,5 | Fio innføring sluttrapport |
| 08.04.2014 | 08:00 - 12:00 | 4,0 | Rettet i sluttrapport, møte. | 08:00 - 12:00 | 4,0 | Logo, sluttrapport, møte | 09:30 - 12:00 | 2,5 | Pool, Møte |
| 09.04.2014 | 08:00 - 18:00 | 10,0 | Rettskriving sluttrapport. zfs | 08:00 - 18:00 | 10,0 | Oppsett av zfs klart for tester, fresh install | 08:30 - 18:00 | 9,5 | Oppsett av zfs klart for tester, fresh install |
| 10.04.2014 | 10:30 - 12:30 | 2,0 | Satt opp nytt zfs pool, laget block-devices | 10:30 - 12:30 | 2,0 | Satt opp nytt zfs pool, laget block-devices | 09:00 - 16:30 | 7,5 | ZFs pool, fio test resultater1 |
| | 13:00 - 17:00 | 4,0 | Begynt oppsett av filsystem på block device, skrevet på sluttrapport | 13:30 - 14:30 | 1,0 | xfs vs. ext4 | | | |
| | | | | 17:00 - 18:00 | 1,0 | logo | | | |
| 11.04.2014 | 10:30 - 13:00 | 2,5 | Jobbet med plakat | 10:00 - 13:00 | 3,0 | to-do lista, dvs zfs vs btrfs | 11:00 - 15:00 | 4,0 | Fio resultater |
| 12.04.2014 | | | | 08:00 - 09:00 | 1,0 | zfs pool / bricks | | | |
| | | | | 14:00 - 16:00 | 2,0 | zfs pool , "problemskyting" | | | |
| 13.04.2014 | 08:00 - 16:30 | 8,5 | zfs, rettskriving | 10:00 - 17:00 | 7,0 | zfs konfig, lesing | 10:00 - 16:00 | 6,0 | Fio pool, Fio resultater |
| 14.04.2014 | 08:00 - 14:30 | 6,5 | zfs pool | 08:00 - 14:30 | 6,5 | zfs pool | 10:00 - 16:00 | 6,0 | Fio resultater |
| 15.04.2014 | 09:00 - 11:30 | 2,5 | zfs | 09:00 - 11:30 | 2,5 | zfs | 09:00 - 17:30 | 8,5 | Fio resultater, møte, cleaning av puddy og lippman |
| 16.04.2014 | 10:30 - 17:00 | 6,5 | zfs, sluttrapport | 10:30 - 17:00 | 6,5 | zfs, sluttrapport | 09:00 - 16:00 | 7,0 | Server oppsett |
| 17.04.2014 | 08:00 - 15:00 | 7,0 | zfs, fiotesting frontend, serverbasert oppsett | 08:00 - 15:00 | 7,0 | zfs, fiotesting frontend, sluttrapport | | | |
| | | | | 18:30 - 19:30 | 1,0 | fiotester vandalay | | | |
| 18.04.2014 | 08:00 - 13:00 | 5,0 | skriving av sluttrapport | 08:00 - 14:00 | 6,0 | sluttrapport | | | |
| | 16:30 - 17:30 | 1,0 | Fiotester vm raw | 16:30 - 17:00 | 0,5 | sluttrapport | | | |
| 19.04.2014 | 08:00 - 09:00 | 1,0 | Fiotester vm snapshot | | | | 19:00 -19:30 | 0,5 | Lest bach om Skyhigh I/O |
| 20.04.2014 | 09:30 - 10:00 | 0,5 | Fiotester frontend | | | | 10:00 - 18:00 | 8,0 | sluttrapport |
| 21.04.2014 | | | | | | | 13:00 - 16:00 | 3,0 | sluttrapport |
| 22.04.2014 | 08:00 - 12:00 | 4,0 | gluster, scenario-testing | 08:00 - 12:00 | 4,0 | gluster, scenario-testing, vm | 08:30 - 13:00 | 4,5 | gluster, scenario testing |
| 23.04.2014 | 08:00 - 16:30 | 8,5 | Ny vm, rettskriving, scenario-testing | 08:00 - 16:00 | 8,0 | sluttrapport, LateX, modeller | 08:00 - 18:00 | 10,0 | Scenario-testing, sluttrapport, fio innføring |
| 24.04.2014 | 06:00 - 08:00 | 2,0 | Lagt ut forklaring i vedlegg, skrevet sluttrapport | 11:00 - 16:00 | 5,0 | sluttrapport, LateX | 08:00 - 17:00 | 9,0 | fio innføring |
| | 11:00 - 14:30 | 3,5 | Serverbasert oppsett | | | | | | |
| 25.04.2014 | 08:00 - 15:00 | 7,0 | Sluttrapport, figurer for fio-testinga | 08:00 - 15:00 | 7,0 | Sluttrapport, LateX | 09:00 - 14:30 | 5,5 | fio innføring |
| 26.04.2014 | 08:00 - 13:00 | 5,0 | Statusrapport, figurer | 08:00 - 12:00 | 4,0 | Sluttrapport, LateX | | | |
| 27.04.2014 | | | | 08:00 - 10:00 | 2,0 | Diskusjon av resultater, dvs skriving | | | |
| 28.04.2014 | 09:00 - 15:30 | 6,5 | Sluttrapport | 09:00 - 15:30 | 6,5 | fiotest, satt opp zfs annerledes, diskusjon | 09:00 - 15:00 | 6,0 | Test skriving |
| | | | | | | | 16:30 - 18:30 | 2,0 | Sluttrapport |
| 29.04.2014 | 08:00 - 13:00 | 5,0 | Sluttrapport, figurer | 08:00 - 13:00 | 5,0 | Diskusjon, fio test | 11:00 - 11:30 | 0,5 | Møte |
| | 15:00 - 16:30 | 1,5 | Helping Kent with ZFs stuff | 15:00 - 17:00 | 2,0 | Helping Kent with ZFs stuff | 12:30 - 13:30 | 1,0 | Bilde pdf creation |
| | | | | | | | 14:00 - 17:30 | 3,5 | Prøve ZFs stuff og ringe Kim |
| 30.04.2014 | | | | | | | 11:30 - 17:00 | 5,5 | Bilde tester, sluttrapport |
| 01.05.2014 | 08:00 - 12:00 | 4,0 | Sluttrapport | 08:00 - 12:00 | 4,0 | Sluttrapport | | | |
| 02.05.2014 | 08:00 - 12:00 | 4,0 | Testing på zfs | 08:00 - 12:00 | 4,0 | Testing zfs | 08:00 - 12:30 | 4,5 | Sluttrapport |
| 03.05.2014 | | | | 08:00 - 11:00 | 3,0 | Diskusjon, sluttrapport endringer | | | |
| 04.05.2014 | 10:00 - 12:00 | 2,0 | Formatering av sluttrapport | 15:30 - 16:00 | 0,5 | Fiotester | | | |
| 05.05.2014 | 08:30 - 15:00 | 6,5 | Sluttrapport | 08:30 - 15:00 | 6,5 | Sluttrapport | 09:00 - 15:00 | 6,0 | Sluttrapport |
| 06.05.2014 | 08:00 - 14:30 | 6,5 | Sluttrapport, figurer | 08:00 - 14:30 | 6,5 | Sluttrapport, figurer | 11:00 - 17:30 | 6,5 | Sluttrapport |
| 07.05.2014 | 08:30 - 15:00 | 6,5 | Sluttrapport, møte | 08:30 - 15:00 | 6,5 | Sluttrapport, møte | 08:30 - 16:30 | 8,0 | Sluttrapport |
| 08.05.2014 | | | | | | | 09:00 - 15:00 | 6,0 | Sluttrapport |
| 09.05.2014 | 08:30 - 11:00 | 2,5 | Sluttrapport, invitasjon til fremføring | 09:00 - 11:30 | 2,5 | sluttrapport | 15:30 - 18:30 | 3,0 | Sluttrapport |
| 10.05.2014 | | | | | | | 13:00 - 16:30 | 3,5 | Sluttrapport |
| 11.05.2014 | 08:00 - 12:00 | 4,0 | Sluttrapport | 08:00 - 12:00 | 4,0 | Sluttrapport | 13:30 - 14:30 | 1,0 | Sluttrapport |
| 12.05.2014 | 08:00 - 12:30 | 4,5 | Sluttrapport | 08:00 - 12:00 | 4,0 | Sluttrapport | 08:00 - 12:30 | 4,5 | Sluttrapport |
| | | | | | | | 13:00 - 14:00 | 1,0 | Sluttrapport |
| 13.05.2014 | 11:00 - 12:00 | 1,0 | Møte | 11:00 - 12:00 | 1,0 | Møte | 11:00 - 12:00 | 1,0 | Møte |
| | | | | | | | 17:00 - 17:30 | 0,5 | Sluttrapport |
| 14.05.2014 | 08:00 - 17:00 | 9,0 | Sluttrapport, GlusterFS feilsøking | 08:00 - 17:00 | 9,0 | Sluttrapport | 08:00 - 18:30 | 10,5 | Sluttrapport, reinstall frontend |
| 15.05.2014 | | | | 08:00 - 11:00 | 3,0 | Fio, oppsett 1 | | | |
| | | | | 15:00 - 15:30 | 0,5 | fio tester resultater VM | | | |
| 16.05.2014 | | | | 08:00 - 08:30 | 0,5 | fio tester block devices frontend | 13:00 - 15:00 | 2,0 | Hentet ut fio tester |
| 17.05.2014 | | | | | | | 13:00 - 15:00 | 2,0 | Fio |
| 18.05.2014 | 12:00 - 16:00 | 4,0 | Sluttrapport | 12:00 - 17:00 | 5,0 | Sluttrapport | 13:00 - 14:00 | 1,0 | Fio, forord |
| 19.05.2014 | 09:00 - 11:00 | 2,0 | Siste finpuss | 09:00 - 11:00 | 2,0 | Siste finpuss | | | |

Forberedelse presentasjon + individuelt refleksjonsnotat

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| SUM: | | 509,0 | | | 491,5 | | | 520,0 | |
| Total: | 1 520,5 | | | | | | | | |

| | | | | |
|---|---|---|---|---|
| | | **Timeantall:** | | |
| | **Ukenr.** | **Thomas** | **Kim** | **Kent-Marius** |
| | **2** | 13,0 | 13,0 | 18,0 |
| | **3** | 22,5 | 24,0 | 23,5 |
| | **4** | 18,5 | 18,5 | 15,5 |
| | **5** | 17,5 | 17,5 | 19,0 |
| | **6** | 26,5 | 27,0 | 24,0 |
| | **7** | 20,0 | 16,5 | 23,0 |
| | **8** | 24,5 | 21,5 | 23,5 |
| | **9** | 28,0 | 30,5 | 18,0 |
| | **10** | 37,5 | 39,0 | 29,0 |
| | **11** | 32,5 | 32,0 | 46,0 |
| | **12** | 30,5 | 32,0 | 28,5 |
| | **13** | 36,0 | 9,5 | 36,0 |
| | **14** | 41,5 | 43,5 | 39,5 |
| | **15** | 31,0 | 31,0 | 35,0 |
| | **16** | 30,0 | 30,0 | 30,0 |
| | **17** | 30,0 | 30,0 | 32,0 |
| | **18** | 23,0 | 25,0 | 23,0 |
| | **19** | 26,0 | 26,0 | 34,0 |
| | **20** | 20,5 | 25,0 | 22,5 |
| | **21** | Forberedelse presentasjon + individuelt refleksjonsnotat | | |
| | **22** | | | |
| | **23** | | | |
| **Gj. time pr uke** | | 23,14 | 22,34 | 23,64 |

# Tillegg J

# Forprosjektrapport

# Kapittel 1

# Mål og rammer

## 1.1 Bakgrunn

Høgskolen i Gjøvik(HiG) jobber med å utvikle en skyløsning kalt SkyHiGh. Skyløsningen skal kjøre virtuelle maskiner. De virtuelle maskinene kan brukes i diverse fag, som for eksempel Database og Applikasjonsdrift og System Administration, men det må presiseres at det ikke kjøres der per dags dato.

Studentene kan også bruke denne løsningen for å besvare andre oppgaver som skal gjennomføres. Skyløsningen skal dermed gi studenter ved HiG muligheten til å lage egne virtuelel maskiner, uten å måtte bekymre seg om at tilstanden til disse blir borte ved et eventuelt strømbrudd.

Foreløpig finnes det lite til ingen recovery-løsning på systemet som er implementert. Det er da her vår løsning kommer til å trå i kraft, slik at vi får opprettet gode rutiner for recovery prosedyren. Herunder går også lagring av produksjonsdata og backup av den gitte informasjonen.

Oppdragsgiver har et par problemer som han ønsker at vi skal finne løsning på:

- Finne ut konsekvensene av at serverne mister strømtilførselen.
- Hvordan få systemet til å komme opp igjen raskt og effektivt.
- Finne ut hvilke lagringsløsning som fungerer med tanke på stabilitet og ytelse.

## 1.2 Prosjektmål

I nåtidens løsning av SkyHiGh , så er det en rekke ting som kan forbedres. Blant annet problemet med at når flere skal kjøre kommandoer på de virtuelle maskinene, så tar det veldig langt tid før disse kommandoene blir ferdig. Det finnes heller ingen forsikring for at dataen blir ivaretatt når serverne kræsjer.

Effektmål:
Vi vil derfor ha effektmål som både er realistiske og innenfor det vi har fått i oppgave. Disse skal også være målbare for å se om vi har nådd de målene vi ønsker.

- Systemet skal komme raskere opp igjen etter et eventuellt strømbrudd, og minimalisere mengden datatap.
- Finne ut hvilket software-miljø eller hardware-miljø som fungerer best i forhold til ytelse og stabilitet.

Resultatmål:
Vi har også en rekke resultatmål vi vil prøve å oppnå. Disse målene har blitt satt opp ut ifra hva vi har fått som oppgavebeskrivelse av arbeidsgiver.

- Resultatet vil bli tatt i bruk i SkyHiGh.
- Systemet skal være skalerbart.
- Vite hva som skjer når systemet kræsjer.
- Forbedre hverdagen til de som benytter seg av SkyHiGh.

## 1.3 Rammer

Vi har ikke fått tildelt noe øvrig grense med tanke på pris ved et eventuelt kjøp av UPS(Uninterruptable Power Supply/Source). Ved hjelp av en online guide fikk vi tre alternativer til UPS som passer for vårt system, hvor den billigste koster kr 11 756,17 (1 400 EURO, litt varierende etter hvor man undersøker). Navnet på UPS er følgende, APC Smart-UPS 3000VA LCD 230V. [1]

Det er i tillegg to alternativer til den nevnte UPS ovenfor:

- APC Smart-UPS XL 3000VA 230V Tower/Rack Convertible + (1)SUA48XLBP Battery Unit (2 169 EURO)
- APC Smart-UPS XL 3000VA 230V Tower/Rack Convertible + (2)SUA48XLBP Battery Unit(2 738 EURO)

For mer informasjon om hvordan en UPS vil stå i forhold til vår oppgave, se delkapittel 2.3.

Vi må benytte oss av gamle servere fordi det er kun disse som har plass til 3.5" disker. Hovedgrunnen til å bruke 3.5" HDD er fordi lagringskapasiteten er rimelig mye større enn i en 2.5" HDD. Dette vil medføre at vi får en noe nedsatt hastighet i backend i oppsettet vårt.

Vi har fått to forskjellige løsninger som vi skal jobbe utifra. Den ene vil være en hardware-løsning som vil bestå av enten xfs eller btrfs. Bcache vil bli brukt som cache opp imot filserverne.
Den andre vil være en software løsning som bare består av ZFS, som har alle delene nevt ovenfor under samme shell.
Disse to løsningene er beskrevet med større detalj i kapittel 2.3.

[1] http://www.apc.com/tools/ups_selector/NO/no/server/device/results

# Kapittel 2

# Omfang

## 2.1 Fagområde

Helheten av oppgaven er innenfor driftsmiljøet som man finner innen IT-fagfeltet. Dette er fordi at oppgaven i seg selv går ut på å håndtere hva som må til for å kunne forebygge datatap i et OpenStack miljø.
Herunder legges det da vekt på hva som skjer når strømmen går, og hvordan man kan implementere gode rutiner for å kunne holde tilstanden stabil. Vi tar for oss ulike hendelser som kan resultere i et datatap.

## 2.2 Avgrensning

Backup, lagring og recovery området i seg selv er et stort område, og for å kunne utarbeide det oppdragsgiver vil ha implementert må vi avgrense oss til noen få scenarioer der man kan teste systemet.

Det som hovedsaklig vil bli testet er hva som skjer dersom serverne skulle miste strømmen, eller om man mister forbindelsen mellom dem. Det man ikke skal ta høyde for er om det er noen eventuelle hardware feil som kan oppstå, eller om noen deler er defekte. Dette vil bli tatt hånd om av systemet når det er satt ut i produksjon, hvor man da kan bytte ut f.eks. disker under kjøring. Med andre ord vil man fokusere mye på det software relaterte aspektet med recovery, backup og lagring.

Man vil antakeligvis ikke ha muligheten til å teste mer enn to løsninger, samt hvordan disse systemene håndterer de ulike scenarioene som oppstår. Dette fordi det kan skape problemer med ukjente parametre strukket over flere forskjellige systemer. Vi vil fokusere på å ha en bcache løsning med xfs/btrfs og HWRAID 10, samt en annen løsning som benytter seg av zfs. Begge løsningene skal være i et cluster som er definert av glusterFS systemet.

Disse to løsningene vil bli testet opp mot et par forhåndsdefinerte scenarioer som skal demonstrere hva som skjer dersom man mister strømmen eller om man mister forbindelsen mellom serverne.

Hvordan dette skal gjøres vil man gå i detalj på i kapittel 2.3 Oppgavebeskrivelse.

## 2.3 Oppgavebeskrivelse

Oppgaven går ut på finne ut hvilke Uninterruptible Power Supply (UPS) som kan være den rimeligste og mest effektive foruten en software løsning. For å deretter teste softwareløsninger som kan være et alternativ til en gitt UPS. Det må her presiseres at vi som gruppe skal ikke gå til innkjøp av en UPS, og den kommer mest sannsynlig ikke til å bli bestlit og implementert før oppgaven vår er ferdig gjennomført.

Vi vil sette opp to ulike software-miljøer for å teste ulike scenarioer på disse miljøene, for deretter å se hva som eventuelt kan være mest effektivt, eller hva som kan være den optimale løsningen mellom en software implementasjon og en hardware implementasjon.

Som nevnt vil man jobbe med to ulike løsninger, disse bygger på dette gitte oppsettet:

Løsning 1:

- xfs/btrfs
- bcache
- Logical Volume Manager (LVM)
- HWRAID 10
- 32GB RAM

Løsning 2:

- zfs on linux
- 32GB RAM

Overordnet vil begge løsningene ligge i et cluster, som er definert av glusterFS for å skape redundans mellom systemene som skal kjøres.

På disse løsningene skal man teste ulike scenarioer. Det er hovedsaklig to scenarioer som skal utføres:

Scenario 1:
Vi fjerner strømmen fra serverne og ser hvordan recovery prosessen kan utføres. Hensikten er da å se hvilken tilstand vi får miljøet tilbake i, samt å kunne determinere hvor mye data som eventuelt er tapt.

Scenario 2:
Vi fjerner tilkoblinger mellom servere, for å se hvordan man kan få serverne synkronisert igjen.

Alt av lagring som skal benyttes i systemet skal være i RAID10, og det skal settes opp en SSD cache, der SSD cachen skal vare i et RAID1.
Det vil ikke bli benyttet noen spesielle lisenser for disse produktene vi setter sammen til en løsning, slik at man ikke har noen direkte konflikter med produkter.

# Kapittel 3

# Prosjektorganisering

## 3.1 Ansvarsforhold og roller

Ved prosjektet vårt så er førsteamanuensis Erik Hjelmås, ved Høgskolen i Gjøvik, oppdragsgiver. Vår veileder er da høgskolelektor Øivind Kolloen ved Høgskolen i Gjøvik. Vi vil benytte oss av hans faglige kunnskap for å oppnå den progresjonen som er ønsket av gruppa.

Gruppa vår er bestående av 3 medlemmer:
Kim Holmebakken: Gruppeleder, hovedkontaktpersonen.
Thomas R Kristiansen: Gruppemedlem.
Kent-Marius Werner: Gruppemedlem, Webansvarlig.

## 3.2 Regler

I denne delen av forprosjektrapporten har vi valgt å ha med våre grupperegler for prosjektet.

§**1 Kommunikasjon:**

a Vi må klare å ha dialoger på sms, samt "i virkeligheten". Blir ikke sms besvart, blir vedkommende kontaktet via telefon.

§**2 Arbeid som utføres:**

a Det forventes at arbeid som en person er satt opp til å gjøre, faktisk blir utført innen tidsfristen som er satt opp.

  i Dersom et medlem trenger hjelp, evt. ikke rekker å gjøre ferdig sin oppgave, må det gis beskjed til resten av gruppa.

  ii Det kan da evt. bli enighet om at oppgaven skal gjennomføres i løpet av dagen, avhengig av estimert tidsbruk, men på bekostning av vedkommendes fritid.

b Det skal føres timer for arbeidet du utfører, dette skal også ha en beskrivelse. Jfr. §3.

c Språket som brukes når det skrives i rapporten skal være formell og vel strukturert.

d Det er forventet at hver deltaker i gruppa jobber minimum 20 timer per uke, men timeantallet bør nok havne på rundt 30 timer per uke, helst over, ettersom dette er gjennomsnittlig arbeidstid for 20 studiepoeng. Det presiseres at timeantallet regnes som et gjennomsnitt.

e Ved en eventuell faglig uenighet, skal gruppa sette seg sammen og diskutere seg frem til en ordning. Hvis dette ikke er mulig vil det ble kontaktet veileder for innsyn.

§**3 Føring av timelister:**

a Timer skal føres opp i eget dokument, “Timeføring”. Føres med følgende informasjon:

  i Start og slutt tid, samt dato.

  ii Antall timer.

  iii Kort beskrivelse av oppgaven.

b En time tilsvarer 60 minutter. Dette inkluderer muligheten til 15 minutters pause om ønskelig.

c Hvis pauser ikke benyttes, kan denne tiden føres opp som ekstra timer i timeføringa.

§**4 Møter:**

a Møt opp til møter som har blitt avtalt, og vær forberedt.

b Det vil bli avholdt daglige møter innad i gruppa hvor det blir diskutert hva som er gjort, samt hva som er planen videre. Med daglig menes hverdager.

c Det vil også bli avholdt ukentlige møter med oppdragsgiver og veileder for å sikre at fremgangen i prosjektet holder mål.

d Gi beskjed i god tid hvis du ikke kan møte på det gitte tidspunktet, slik at møtet kan flyttes.

e Møter vil bli avholdt klokka 11:00 tirsdag til fredag, og 14:00 på mandager. Den dagen det er møte med veileder og oppdragsgiver blir det ingen intern møte.

§**5 Sykdom:**

a Dersom en av medlemmene skulle bli syke under arbeidsforløpet må vedkommende melde fra til gruppeleder eller et annet gruppemedlem, samt angi en tidsperiode der man estimerer når man er friskmeldt igjen.

b Dersom en sykdom strekker seg over 3 uker må medlemmet kontakte gruppen for å finne en løsning på hva som eventuelt kan gjøres.

c Dersom et av medlemmene er syke i lengre perioder (over 3 uker), må det tas en beslutning på om vedkommende kan bli værende i gruppen.

§**6 Opphavsrett:**

a Hver av gruppemedlemmene har likeverdig opphavsrett til alle dokumenter som blir skrevet gjennom prosjektets tidsperiode. Dette gjelder dokumenter som skrives i Google Docs samt dokumenter som ligger i versjonskontrollsystemet.

b Ved brudd på opphavsretten, enten ved at et medlem fjerner tilgang til de andre medlemmene eller at et medlem sletter all informasjon, blir det utkastelse fra gruppa samt at informasjonen må skaffes til veie igjen på vedkommendes bekostning.

§**7 Brudd på regelverk:**

a Dersom det blir for mange brudd på disse reglene, vil det bli meldt ifra til Studie-Program-Ansvarlig og veileder.

# Kapittel 4

# Planlegging, oppfølging og rapportering

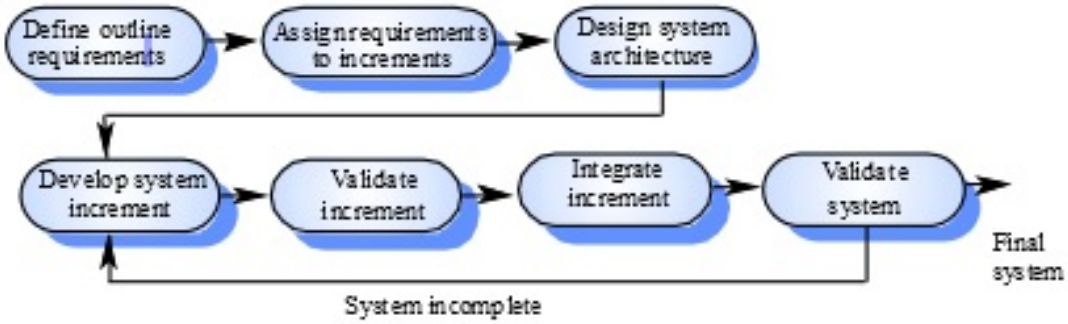
## 4.1 Hovedinndeling av prosjektet

Karakteristikker for vårt prosjekt: Prosjektet vil i første omgang hovedsaklig dreie seg om å studere ulike varianter av datalagring og caching i OpenStack systemet på HiG, SkyHigh. Til nå har oppdragsgiver satt opp en mulig løsning på lagringsløsningen, så vi må i denne første perioden se på alternativer til dette. Det vil da bli vurdert om oppdragsgivers løsning er optimal, eller om det finnes ett bedre alternativ.

Siden det er kun eksisterende løsninger som er aktuelle å implementere i vårt tilfelle, taler jo dette mot å bruke deler av gjenbruksmetoden i vårt prosjekt. Dette vil føre til mindre dokumentasjon enn ved andre modeller, som f.eks. Scrum og RUP, grunnet at løsningene allerede finnes.

Den tildelte oppgavebeskrivelsen var rimelig åpen i definisjonen, så det kan komme endringer senere i prosjektet. Eventuelle endringer senere i prosjektet taler mot å bruke en iterativ modell, som inkrementell modell eller Scrum. Inkrementell modell deles inn i to deler, evolusjonær og sekvensiell. Evolusjonær inkrementell føler vi at blir for uoversiktlig i et så stort prosjekt som en bacheloroppgave, så den er derfor ute av bildet i valg av modell.

Som nevnt over da står vi igjen med to modeller som vi føler står ganske likt, sekvensiell inkrementell modell og Scrum. Grunnet timeplaner og våre preferanser for de ulike modellene, velger vi å bruke sekvensiell inkrementell modell i vårt prosjekt.




## 4.2 Plan for statusmøter og beslutningspunkter

Det er lagt opp ukentlige statusmøter som omgår gruppa som helhet, veileder og oppdragsgiver. Under disse møtene skal det gjennomgås det arbeidet som er gjort siden sist møte. Det skal også planlegges hva som skal jobbes med i den neste uken. Her er det medregnet at veileder kommer med tips til endringer som kan gjøres underveis i prosjektet. Veileder har her meddelt at ukentlige møter vil forekomme frem til gruppen er godt igang med oppgaven, og deretter vil statusmøtene med veileder ikke være like ofte.

I tillegg til dette skal det avholdes daglige møter innad i gruppa for å passe på at gruppemedlemmene arbeider i henold med grupperegler, samt at de oppgavene som er satt blir gjort ferdig til rett tid. I disse møtene vil det bli gjennomgått hva som er gjort siden forrige møte, og hva som blir de enkeltes oppgave til neste møte. Det vil bli tatt hensyn til det enkelte medlems timeplan når oppgaver tildeles.

Vi vil i tillegg til statusmøter skrive statusrapporter for å holde oppdragsgiver og veileder med på hva som skjer i forhold til oppgaven vi løser. Statusrapporten vil være en overordnet beskrivelse av arbeidet som er gjort fra sist rapport ble levert, med vekt på praksisen som er gjennomført, som vil beskrives i litt større detalj. Det vil bli levert statusrapporter en gang hver måned. Grunnen for dette er at vi da ofte nok kan holde oppdragsgiver og veileder oppdatert på fremgangen, og det blir heller ikke rapportert for ofte.

# Kapittel 5

# Organisering og kvalitetssikring

## 5.1 Dokumentasjon, standardbruk og kildekode

For å kunne ha en oversikt over hva som er blitt implementert i løsningen som settes opp, skal det dokumenteres alt av konfigurering, pakke-installasjoner og kommandoer som er blitt kjørt. Dette skal gjøres fortløpende og presist iht. de kommandoene som blir kjørt.

Gjennom prosjektet vil det bli brukt Google Docs for å holde orden på de ulike delene av prosjektet. Her vil det bli skrevet utkast til rapporter, ført timelister, skrevet referat fra møter o.l. Ved sammensetting av rapport vil tekst fra utkastene finskrives, deretter videre kopiert og lagt inn i LaTeX-filer. Vi bruker dette formatet for å få det ønskede utseende på rapporten.

Når det kommer til standarder skal det utføres så nær "best practice" innen fagfeltet som man kommer. Dette er for å sikre et universelt oppsett som gjør at dersom man skal utarbeide andre implementeringer til systemet vil dette kunne bli gjort av fagpersoner og/eller studenter innen samme fagområde. Siden fagfeltet er relativt nytt har det dermed ikke blitt utrettet mange standarder som må følges, og vi føler at omfanget ved å knytte dette tett til ITIL vil bli for stort iht. prosjektets størrelse.

I tillegg til dette vil det være lite til ingen utvikling av kildekode fra gruppens side, men det vil bli brukt kildekode som allerede brukes i de moduler vi implementerer i vårt prosjekt. Denne kildekoden vil bli henvist i referanser, hovedsaklig gjelder dette da referansemanualer til den gitte modulen.

## 5.2 Versjonsstyring

Som versjonskontroll har vi valgt å bruke Subversion som verktøy. Dette er fordi Subversion er et verktøy vi er godt kjent med fra før, samt at skolen tilbyr dette som en tjeneste. Subversion har også de fasiliteter som oppfyller våre krav.

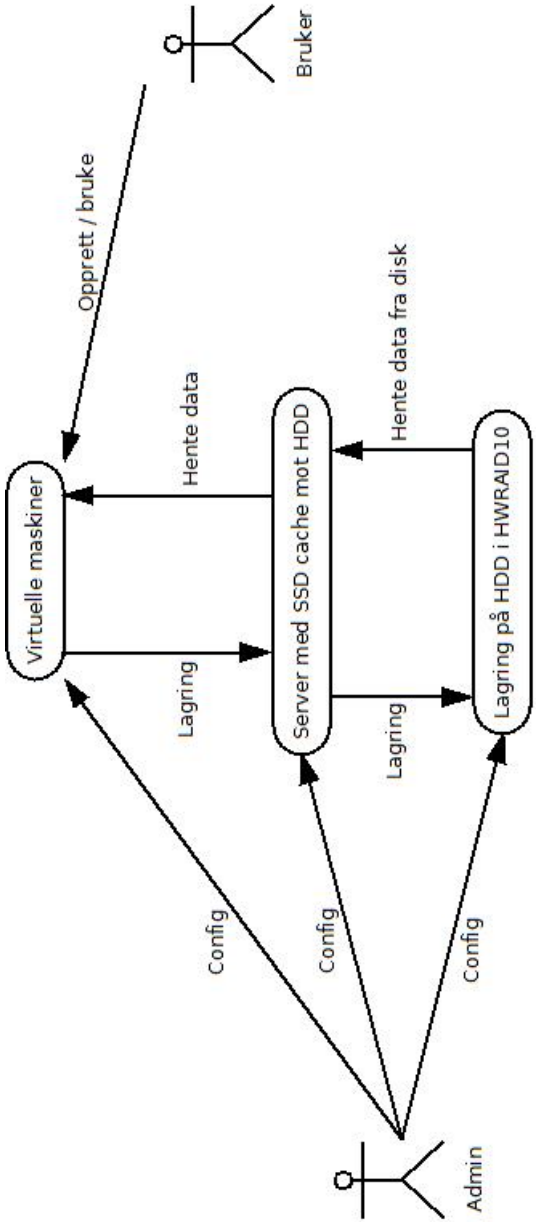
I tillegg til Subversion bruker vi som nevnt tidligere også Google Docs ved skriving av utkast til rapport. Google Docs har egen endringslogg implementert, og gruppa kan dermed gå tilbake til tidligere versjoner hvis dette er ønskelig. Dette gir også muligheten til større redundans i prosjektet, ved at filene i Google Docs kan synkroniseres lokalt hos de respektive medlemmene i gruppa.

All konfigurasjon blir også dokumentert i Google Docs, og vil da gi oss
1 muligheten til å holde kontroll over hva som er blitt implementert, jfr. kapittel 5.1 for mer om dokumentasjon.

# Kapittel 6

# Plan for gjennomføring

## 6.1 Use case diagram



## 6.2 Høynivå use case

| Use case: | Virtuelle maskiner |
|---|---|
| Aktør: | Bruker, Admin |
| Mening: | Virtuelle maskinene kjøres i SkyHiGh. |
| Beskrivelse: | Administratoren skal konfiguere og tildele tilgang til de virtuelle maskinene. Brukerne skal benytte seg av disse maskinene til forskjellige fag og oppgaver. Virtuelle maskinene skal lagre data opp imot serverne som har SSD i seg. De skal også hente data derifra. |

| Use case: | Server med SSD cache mot HDD |
|---|---|
| Aktør: | Admin |
| Mening: | Skal være frontend mot 3,5” HDD storage. |
| Beskrivelse: | Skal ta imot data som kommer fra de virtuelle maskinene, deretter skal det lagres på cache som er satt i et bcache miljø med SSD. Cachen skal så tømmes til 3,5” HDD som er en egen server for lagring på HDD i HWRAID1+0. |

| Use case: | Lagring på HDD i HWRAID1+0 |
|---|---|
| Aktør: | Admin |
| Mening: | Ligger som en treg backend i systemet, lagrer data fra cache. |
| Beskrivelse: | HDD i det fysiske oppsettet vil i all hovedsak ligge som en treg backend. HDD er satt opp i HWRAID10(RAID1 + 0). Dette vil si at vi bruker både speiling og striping, som igjen medfører at systemet kan utnytte $^1/_4$ av den totale lagringskapasiteten. Striping vil føre til at systemet får økt hastighet. RAID’et skal ta imot informasjon fra SSD cache når lasten er lav nok, og levere data til SSD cache ved etterspørsel. |