BACHELOROPPGAVE:

SKYHIGH I/O

YTELSESVURDERINGER OG BESTE PRAKSIS FOR I/O I SKYHIGH

FORFATTERE:

EIRIK BAKKEN
ERIK RAASSUM

Dato:

23.05.2012

SAMMENDRAG AV BACHELOROPPGAVEN

Tittel:	SkyHiGh I/O	Dato: 23.05.2012
Deltakere:	Eirik Bakken Erik Raassum	
Veileder:	Hanno Langweg	
Oppdragsgiver:	Erik Hjelmås	
Stikkord: (3-5)	Nettsky, OpenStack, virtuelle maskiner, disk- og nettverksytelse, iSCSI	
Antall sider: 196	Antall vedlegg: 10	Publiseringsavtale inngått: Ja

Kort beskrivelse av master/bacheloroppgaven:

<abstract>
Nettskyteknologien og virtuelle maskiner er blitt «allemannseie» i dataindustrien og øker stadig i popularitet. Nå kan man leie tjenester (virtuelle maskiner for eksempel) fra offentlige nettskyer eller sette opp en nettsky til privat bruk. Disse virtuelle maskinene har en viss mengde minne, prosessorkapasitet og I/O-karakteristika (disk og nettverk).

Tradisjonelt sett har I/O-ytelse i virtuelle maskiner ikke klart å tangere fysiske maskiner. Disse tilkortkommenhetene kan i visse tilfeller motvirkes noe ved å benytte seg av beste praksis-konfigurering. I denne oppgaven har gruppen testet ytelsen på disk og nettverk i virtuelle Linux-maskiner i en nettsky kalt «SkyHiGh» som er implementert som del av en annen bacheloroppgave. Nettverksytelse ble målt med tre ulike drivere og diskytelse ble undersøkt i ulike scenarier, med filene som de virtuelle maskinene ser på som sin fysiske disk oppbevart på ulike lokasjoner: på harddisken til den fysiske maskinen de virtuelle maskinene kjører på eller på et iSCSI-target delt av en lokal lagringsnode.

Målet til denne bacheloroppgaven har vært å undersøke og demonstrere beste praksis som gjelder for optimalisering av diskytelse og nettverksytelse for virtuelle Linux-maskiner når disse kjører i nettskyen «SkyHiGh». Fremkomne resultater viser at ytelse tydelig er sensitivt overfor konfigurasjon, og lesere vil få kunnskap om nettskyer, utfordringer knyttet til I/O for virtuelle Linux-maskiner og beste praksis-anbefalinger.
</abstract>

THESIS ABSTRACT

Title:	SkyHiGh I/O	Date: 23.05.2012
Participants:	Eirik Bakken Erik Raassum	
Supervisor:	Hanno Langweg	
Employer:	Erik Hjelmås	
Keywords (3-5)	Cloud, OpenStack, virtual machines, disk and network performance, iSCSI	
Number of pages: 196	Number of appendices: 10	Availability (open/confidential): Open

Short description of the bachelor thesis:

Cloud computing and virtual machines have become ubiquitous terms in the industry and are ever-increasing in popularity. Today you can provision virtual machines from public clouds throughout the world, or you may setup your own private cloud. These virtual machines have a certain amount of memory, processor-capacity and I/O (disk and network) characteristics.

Traditionally, I/O in virtual machines has been unable to perform at the same level as physical machines. These shortcomings can in many cases be alleviated to some extent by consulting best practices when configuring.

In this project, we have tested disk and network performance in virtual Linux machines in an Infrastructure-as-a-Service cloud named “SkyHiGh” that has been implemented by another bachelor thesis group. Network performance was measured with three different drivers, and disk performance was assessed in different scenarios, with the virtual machines' images being in two different locations: the physical harddisk of the machine running the virtual machines and on targets served by an iSCSI-server.

The goal of this bachelor thesis has been to identify and confirm best practices with regard to optimizing disk and network performance in virtual Linux machines that are running in the “SkyHiGh” cloud. Derived results clearly show that configuration plays a huge role in attained performance, and readers will become acquainted with cloud computing nomenclature, challenges concerning I/O in this context and relevant best practices.

Forord

Av bacheloroppgavene som ble foreslått av oppdragsgivere for våren 2012 var det "SkyHiGh"-prosjektet gruppen i størst grad ønsket å fordype seg i. Som følge av en annen bachelorgruppes interesse ble SkyHigh delt i to oppgaver – én for administrasjon og implementasjon av en nettsky, og én for undersøkelse av I/O i implementasjonen av denne nettskyen.

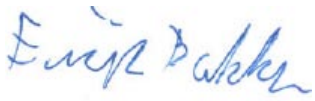
Lærekurven for oppgaven har vært bratt og krevende, men mye nyttig, relevant og spennende kunnskap er fremkommet både som resultat av sluttrapportskriving, teorifordypning og ukentlig diskusjon med oppdragsgiver og veileder under oppgaveløpets gang.

Gruppen føler at bacheloroppgaven har medført stort faglig utbytte, både i forhold til å planlegge og utføre større oppgaver, og rundt fordypning i teknologi og fagstoff som i høyeste grad er yrkes- og studierelevant.

En stor takk rettes oppdragsgiver Erik Hjelmås for variert og utstrakt hjelp og pågående dialog i løpet av hele oppgaveløpet, og til veileder Hanno Langweg, som har gitt oss råd, konstruktiv kritikk og faglig støtte knyttet til den praktiske gjennomføringen av bacheloroppgaven. Gruppen ønsker også å takke den andre gruppen knyttet til SkyHiGh-prosjektet for godt samarbeid og hjelp underveis i dette semesteret.

Gjøvik 23.05.2012

Eirik Bakken



Erik Grina Raassum

Ordforklaringer

AoE	ATA over Ethernet
AWS	Amazon Web Services
Bonding	Link Aggregation
Compute-node	Fysisk maskin som kjører virtuelle maskiner
Controller-node	Fysisk maskin som kontrollerer OpenStack
EBS	Elastic Block Storage
EC2	Elastic Compute Cloud
Ekstern lagring	iSCSI-target, NFS-område e.l
FC	Fibre Channel
FIO	Flexible IO Tester
Gantt-diagram	Planleggingsskjema for oppgaveløp
Gjest	Virtuell maskin
HiG	Høgskolen i Gjøvik
Hypervisor	Programvarelag som kontrollerer virtuelle maskiner
IaaS	Infrastructure as a Service
IMRAD	Introduction, Methodology, Results And Discussion
IMT	Avdeling for Informatikk- og Medieteknikk
I/O	Input/Output
IOPS	Input Output Operations per Second
Instans	Virtuell maskin
Instanstype	Maskinvaren den virtuelle maskinen har tilgjengelig
iSCSI	Internet Small Computer System Interface
Jumbo Frames	Ethernet-frames som innholder mer enn 1500 bytes
KVM	Kernel-Based Virtual Machine, en populær fri-kildekode hypervisor
Lagringsnode	Fysisk maskin som kjører iSCSI-server
Link Aggregation	Sammenkobling av nettverkskort til én logisk enhet
Lokal lagring	Lagring som fysisk tilhører en maskin
LVM	Logical Volume Manager / Logical Volume Management

MB/s	Megabyte per Sekund
Mbit/s	Megabit per Sekund
Megabit	1Mbit tilsvarer 10^6 bit
Megabyte	1MB tilsvarer 2^{20} byte
MTU	Maximum Transmission Unit
NAS	Network Attached Storage
OpenStack	IaaS-nettsky basert på åpen kildekode
Overhead	Ekstradata eller omvei som påvirker hastighet/respons
PaaS	Platform as a Service
QoS	Quality of Service
RAID	Redundant Array of Independent Disks
S3	Simple Storage Service
SaaS	Software as a Service
SAN	Storage Area Network
SkyHiGh (I/O)	Kodenavn på nettsky/bacheloroppgaven
Scheduler	Program eller algoritme for å bestemme rekkefølger
Throughput	Hastighet på overføring av data
Virtio	Linuxstandard for øking av I/O-ytelse i VM
Virtual Machine Monitor	Hypervisor
Virtuell maskin	En logisk/emulert komplett datamaskin
VM	Virtuell maskin
Xen	Populær fri-kildekode hypervisor

Innhold

Figurer

Tabeller

Fotnoter

Kapittel 1

Innledning

Bakgrunnen for denne oppgaven er at oppdragsgiver ønsker å vite hvilken ytelse som foreligger i forhold til I/O for virtuelle maskiner i en privat nettsky som implementeres våren 2012. Når det gjelder prosesseringskraft og minne kan ytelse ofte forbedres ved å allokere mer maskinvare til problemet, men for det samme spørsmålet vedrørende I/O er ikke svaret like sort-hvitt. Med I/O menes i denne sammenheng primært nettverk og datalagring, men også klientaksess går tradisjonelt inn under begrepet når det er snakk om virtualisering.

Nettskyer (cloud computing) er en spennende teknologi som har fått mye oppmerksomhet de siste årene. Den gjør det mulig for sluttbrukere å kjøre sine virtuelle maskiner på andres maskinvare og infrastruktur. En slik nettsky, med kodenavn SkyHiGh, vil bli implementert for IMT på Høgskolen i Gjøvik i løpet av våren 2012 som en parallell bacheloroppgave, og det bringer på bane mulige utfordringer i forhold til oppnåelse av tilfredsstillende grad av I/O-ytelse.

En rekke emner og aktiviteter ved utdanningsinstitusjoner som Høgskolen i Gjøvik kan ha nytte av å flytte gjøremål over på en nettsky. Dette kan dreie seg om datalabaktiviteter for studenter, slik som øvinger og andre oppgaver, i emner som i dag krever at studenter enten benytter seg av datalaboratorium eller virtuelle maskiner på privateide bærbare maskiner. En slik nettsky har potensiale til å kunne gjøre slike emner enklere å gjennomføre og forberede undervisningen til, forutsatt at en har ytelse som tilsvarer fysiske maskiner.

Det eksisterer også et mulig behov for en slik nettsky i forskningsøyemed, hvor krevende forskningsvirksomhet som er avhengig av store mengder regnekraft får tildelt den maskinvaren som trengs for å få utført oppgaven, tidsnok og i den grad forskningsoppgavens innhold fordrer. Andre eksempler på bruk inkluderer utprøving av teknologi på skoleadministrerte virtuelle maskiner

(for eksempel studenter som har lyst til å prøve Linux).

Oppdragsgiver ønsker derfor en utredning hvor man ser på I/O-ytelsen for virtuelle maskiner i nettskyen som implementeres våren 2012, og vurderer og drøfter ulike årsaker til eventuelle ytelsesproblemer, og forsøker å utlede noen generelle anbefalinger i forhold til hvordan en kan forbedre dette området i form av konfigurasjonsalternativer eller et alternativt lagringsoppsett fra det som er standard.

Oppgavebeskrivelse

Oppdragsgiver har flere ønsker i forhold til hva han ønsker at gruppen skal undersøke og presentere i en avsluttende rapport. Disse ønskene følger nedenfor:

- Vurdering og drøfting av de ulike alternativene og løsningene for hvordan de virtuelle maskinene kan lagres i SkyHiGh
- Testing av ytelse for nettverk og lokal og ekstern lagring (iSCSI) i SkyHiGh, for virtuelle og fysiske maskiner tilknyttet infrastrukturen
- Identifisering av hvilke potensielle flaskehalser eller optimaliseringer som eksisterer eller vil kunne være gjeldende og om mulig gi noen generelle anbefalinger for hvordan ytelse potensielt kan forbedres på mulige tilkortkommende områder

Rent konkret skal deltakerene i gruppen være i stand til å gi noen anbefalinger i forhold til de forskjellige løsningene, alternativene og konfigurasjonsmulighetene som eksisterer for OpenStack og den valgte hypervisoren i sluttfasen av denne bacheloroppgaven, som følge av data og resultater som fremkommer fra ytelsestesting samt en teoretisk fordypningsdel.

Avgrensninger

Som følge av denne oppgavens åpne natur og potensielt store omfang, er det fornuftig å avgrense innholdet tidsnok og i stor nok grad. Blant annet vil valg av infrastrukturoppsett og implementering av OpenStack håndteres av en annen bacheloroppgave som foregår parallelt. SkyHiGh I/O vil kun se på ytelsesrelaterte forhold i implementasjonen og infrastrukturen oppført av den parallelle bachelorgruppen. Dette kan være begrensende i forhold til

hvilke alternativer gruppen er i stand til å kunne teste ettersom det ikke nødvendigvis vil være praktisk mulig å endre på en del konfigurasjonsmuligheter når nettskyen er implementert. På den annen side gjør dette at gruppen får mer tid til å se på oppdragsgivers ønsker, om enn ikke i like omfattende utstrekning.

En annen avgrensning som gjøres gjeldende er at oppgaven i utgangspunktet ikke vil omhandle klientaksess til de virtuelle maskinene som kjører på SkyHiGh, med mindre det blir tilstrekkelig med tid til overs til å undersøke dette. Med klientaksess menes fjerntilgang over VNC eller RDP [1]. Gruppen vil kun teste ytelse på Linux-plattformen, med mindre tidshensyn tilsier at testing på virtuelle maskiner med annet operativsystem (Windows) ikke vil utgjøre et betydelig ekstraarbeid i form av konfigurasjon og oppsett.

Målgruppe

Rapportens målsetning er å gi leseren kunnskap om hvilke potensielle ytelsesproblemer som eksisterer for I/O og virtualisering, i tillegg til å sette dette i en privat nettskykontekst. Gruppen vil se på ytelsen til virtuelle Linux-maskiner i en OpenStack-arkitektur som settes opp parallelt med denne oppgaven, og sammenligne ytelsesresultatene med de fysiske maskinene i infrastrukturen for å identifisere forskjeller. Oppgaven vil inneholde mye teknisk fordypningsinformasjon i den hensikt at deltakerene får undersøkt noe av den underliggende teknologien involvert, og for å gi lesere en begripelig kontekst.

Målgruppen for sluttrapporten er oppdragsgiver, oppgavens veileder, sensor, fagressurser på HiG, IMT-studenter og ellers de som er interesserte i nettskyteknologi og dertilknyttet problematikk, ytelsestesting og virtualisering.

Prosjektmål

De ulike relevante prosjektmålene er effektmål, som viser til hvilke positive virkninger oppgaven kan bringe på bane; læringsmål, som sier noe om hva gruppedeltakere ender opp med å ha lært etter oppgavens ferdigstillelse, mens resultatmål er de faktiske, målbare og håndgripelige vurderingene, resultatene og eventuelle anbefalingene gruppen kommer frem til i kjølvannet av denne semesterlange bacheloroppgaven.

[1] http://en.wikipedia.org/wiki/Remote_desktop_software/

Effektmål

Effektmålene for oppgaven er disse:

- Eventuelle sluttbrukere som benytter skyen etter vellykket implementasjon opplever, om mulig, bedre ytelse og en mer tilfredsstillende tjeneste ved innføring av fremkomne anbefalinger
- Nettskyen vurderes som enten umoden eller moden for større og mer utbredt implementering til avdelingens formål med den tilgjengelige maskinvaren som infrastrukturen er implementert med

Læringsmål

Læringsmålene som gjelder er at gruppemedlemmene tilegner seg verdifull kunnskap innenfor følgende områder:

- Yrkesrelevante fagområder (herunder nettskyteknologi, virtualisering, ytelse- og ytelsesvurderinger, videreutvikling av ferdigheter innen scripting, grunnleggende statistikk og overvåkning)
- Planlegging, og ikke minst gjennomføring av prosjekter av større omfang

Resultatmål

Resultatmålene for oppgaven er følgende:

- Gruppen kommer frem til relevante anbefalinger som gjelder for ytelse i forhold til lagrings- og nettverksrelaterte hensyn i en privat OpenStack-arkitektur, slik at eventuelle sluttbrukere vil oppleve en så tilfredsstillende tjeneste som mulig innenfor de forutsetninger som legges til grunn
- Vurdering av hva slags ytelse de virtuelle maskinene har kontra fysiske maskiner i forhold til I/O. Dersom ytelsen ikke er tilstrekkelig god, vil studenter og ansatte se til andre alternativer. I så tilfelle vil ikke en virtuell lab nødvendigvis kunne erstatte den fysiske varianten på en måte som gagner sluttbrukerene

Rammer

Gruppen må forholde seg til de til enhver tid gjeldende rammene og begrensningene som er fremsatt av IMT og HiG i forhold til gjennomføring av større studentoppgaver. Med dette menes tidsfrister i løpet av våren 2012, blant annet ved innlevering av forprosjektrapport, kontrakt mellom oppdragsgiver og gruppen og sluttrapportinnlevering i mai.

Andre krav til oppgavegjennomføringen er statusmøter og loggføring i løpet av oppgavens gang. Gruppen er også nødt til å forholde seg til en mal for rapportskriving på bachelornivå for sluttoppgaver utført av studenter på Høgskolen i Gjøvik.

Gruppens faglige bakgrunn

Deltakerene i gruppen er begge tredjeårs bachelorstudenter ved studiet "Drift av Nettverk og Datasystemer". I løpet av disse tre årene har deltakerene vært gjennom mange emner som er relevante i forhold til denne bacheloroppgavens omfang og innhold. Eksempler på slike emner er blant annet "Operativsystemer", "Database- og applikasjonsdrift", "Datakommunikasjon og nettverkssikkerhet", "Nettverksadministrasjon" og "Systemadministrasjon". Denne faglige ballasten vil være særs verdifull i månedene bacheloroppgaven strekker seg over, men likevel innser gruppen at det vil bli behov for store mengder kunnskapsinnhenting samt prøving og feiling for å få forståelse for, og innsikt i, sentrale begreper og teknologi.

Områder som oppgaven omfatter, men som gruppemedlemmene ikke har særlig god innsikt i ved oppgavens startfase, er blant annet nettskyer, måling av disk- og nettverksytelse, grunnleggende statistiske prinsipper for ytelsesmålinger og visualisering av fremkomne ytelsesresultater i tabeller og med grafer. Dette er kunnskap gruppemedlemmene må tilegne seg i løpet av oppgavens gang og veien vil i stor grad bli til mens arbeidet pågår.

Gruppens valg av oppgave falt på SkyHiGh, som ble delt i to som følge av at to grupper var interesserte, fordi gruppen tror nettskyteknologien i økende grad vil bli relevant i mange sammenhenger i hverdagen og smitte over i flere og flere områder i samfunnet innen kort tid, i tillegg til at utfordringene gruppen møter trolig vil kunne være svært nyttig tankeammunisjon å ta med seg når deltakerene senere skal tre ut i arbeidslivet.

Roller

Bachelorgruppen

- Eirik Bakken - Web- og dokumentansvarlig
- Erik Raassum - Prosjektleder

Rollene ovenfor er i stor grad kun tilstede for å ha noen overordnede ansvarsområder for hver enkelt. For de aller fleste andre områder vil gruppen jobbe i fellesskap der det er praktisk og tildele oppgaver etter som oppgaven skrider fremover ut i fra de gjøremål som fremdeles gjenstår å gjennomføre.

Oppdragsgiver

Oppgavens oppdragsgiver er Erik Hjelmås, førsteamanuensis ved Høgskolen i Gjøvik, studieprogramansvarlig for bachelor i "Drift av Nettverk og Datasystemer" og fagressurs på områder som systemadministrasjon og operativsystemer.

Veileder

Oppgavens veileder er førsteamanuensis Hanno Langweg ved Høgskolen i Gjøvik. Han er ressursperson ved avdeling for informatikk- og medieteknikk innen blant annet nettskyteknologi og programvaresikkerhet.

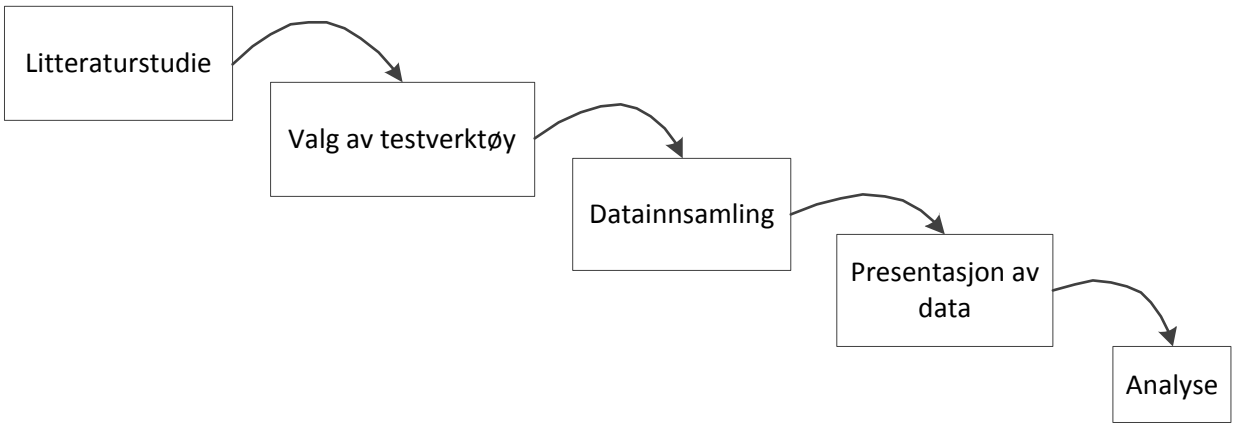
Organisering

Gantt-diagram

Gantt-diagrammet som ble presentert i forprosjektrapporten vil justeres etter som arbeidet skrider fremover. Når oppgaven leveres inn skal den reflektere hvordan tiden faktisk ble brukt og ikke nødvendigvis slik det i januar ble forespeilet at arbeidsflyten kom til å være. Gruppen ser for seg at avvikene mellom tenkt og faktisk gantt-diagram kan komme til å være betydelige, da det er vanskelig å ha innsikt i hvordan arbeidet på en oppgave mer omfattende enn noe annet gruppen har jobbet med til nå i realiteten blir gjennomført. Gantt-diagrammene finnes i vedlegg I.

Arbeidsmetode og utviklingsmodell

SkyHiGh I/O er ikke en typisk systemutviklingsoppgave, men et mer teoretisk og forskningsrettet arbeid, noe som ble diskutert i samråd med veileder og oppdragsgiver. Således er det ikke nødvendigvis noen naturlige utviklingsmodeller å jobbe etter, men gruppen ymtet imidlertid i forprosjektet (se vedlegg J) til at fossefallsmetoden med innfall fra den inkrementelle metoden kunne være fornuftig dersom en slik modell skulle velges. Metoden det overordnet vil arbeides etter og flyt for denne kan observeres i figur 1.1. Den er lik fossefallsmetoden ved at det er klart definerte arbeidsoppgaver som gjennomføres sekvensielt. Muligheten til å gå tilbake til tidligere oppgaver ved behov vedkjennes slik at det også er hint av den inkrementelle metoden å spore i vår arbeidsmetode.



Figur 1.1: Arbeidsmetode

Rapporten

Rapportens struktur vil basere seg på "IMRAD", et oppsett for akademisk skriving som står for "Introduction, Method, Results And Discussion". Dette er en anerkjent metode å legge opp artikler og avhandlinger på. Høgskolen i Gjøvik har en egen IMRAD-inspirert mal for hvordan man skal skrive oppgaverapporter som gruppen vil følge. Dette er spesielt aktuelt siden denne oppgaven i stor grad er av forskningsmessig og teoretisk rettet art.

Vancouverstilen vil benyttes for kilder til forskningsarbeid, artikler, nettsider og bøker, et krav fremsatt av avdelingen gruppen er underlagt. Fotnoter be-

nyttes der det er fornuftig. Vancouverstilen går ut på at man har en numerisk organisert liste over kilder som man refererer til i teksten med klammeparenteser rundt tallet til kilden slik den er plassert i lista, for eksempel "[1]". Kilden referert til i dette eksempelet er den aller første nevnt i litteraturlista.

Oppgaven vil nødvendigvis bestå av å skrive en rekke script for å hente ut ytelsesinformasjon ved hjelp av eksisterende verktøy, i tillegg til analysering og formatering av disse dataene. Kommandoer og kode vil bli vist på følgende måte i rapporten:

```
root@compute:# mount /dev/sda1 /var/lib/nova/instances
```

Oversikt over kapitler

Nedenfor avspeiles hvordan rapportens overordnede kapitteloversikt ser ut og hva de enkelte kapitlene har av innhold.

Kapittel 1 - Innledning

Oppgaven introduseres, sammen med oppgavebeskrivelse, avgrensninger, informasjon om prosjektmål og gruppens bakgrunn.

Kapittel 2 - Teoretisk fordypning

Relevant fordypningsteori presenteres. Her presenteres kunnskap om nettskyer, virtualisering, lagring og nettverk.

Kapittel 3 - OpenStack

I kapittel 3 presenteres nettskyløsningen som skal benyttes i implementasjonen av SkyHiGh og hvilke komponenter denne nettskyen består av.

Kapittel 4 - Ytelsestesting og testverktøy

Konseptet ytelsestesting presenteres. Gruppen går også inn på ulike ytelsestestingmetoder og mulige fallgruver, og vurderer hvilke ytelsestestingsverktøy som skal benyttes.

Kapittel 5 - Testmiljø

I kapittel 5 presenteres infrastrukturen og utstyret slik det er satt opp av den andre bacheloroppgavegruppen.

Kapittel 6 - Praktisk gjennomføring av ytelsestesting

Gruppen dokumenterer her overfladisk testene som skal gjennomføres og hvordan utførelsen av disse vil foregå i praksis.

Kapittel 7 - Resultater

Resultater som er fremkommet under ytelsestestingen presenteres.

Kapittle 8 - Diskusjon og anbefalinger

I dette kapittelet diskuteres resultatene gruppen kom frem til under ytelsestestingen og det forsøkes å resonnere frem årsakssammenhenger og noen generelle anbefalinger.

Kapittel 9 - Konklusjon og evaluering

Konklusjon av oppgaven, forslag til videre arbeid og evaluering av gjennomføringen.

Bibliografi

Kildene (relevant forskningsarbeid, litteratur og nettsider) benyttet i oppgaven refereres her.

Vedlegg

- **A** Utfyllende og detaljert oversikt over alle tester utført
- **B** Linux-oppsett for LVM, iSCSI, link aggregation og jumbo frames
- **C** Utviklede script benyttet i ytelsestestingen
- **D** Arbeidsfil benyttet under testing med verktøyet FIO
- **E** Dagslogg for gruppemedlemmene for hele oppgaveperioden
- **F** Uke- og timeoversikt for gruppemedlemmene for hele oppgaveperioden
- **G** Logg fra møter med veileder og/eller oppdragsgiver
- **H** Kontrakt mellom oppdragsgiver og gruppen
- **I** Gantt-diagram fra forprosjekt i januar samt det faktiske gantt-diagrammet kontinuerlig utbedret for å reflektere realitetens oppgavegjennomføring

- **J** Forprosjekt fra januar

Kapittel 2

Teoretisk fordypning

Det er en hel del terminologi, begreper og konsepter som det er nødvendig å fordype seg i før og under gjennomføringen av en slik oppgave. I dette kapittelet gjennomgås viktige begreper og punkter i relevant teori.

Nettskyen

Nettskyen blir opphøyet til å representere et genuint paradigmeskifte [1] innen den moderne datateknologien og regnes for å være blant de heteste nyvinningene på markedet. Det opereres med mange (flere titalls) definisjoner på hva nettskyen faktisk er og omfatter, og blant disse er US National Institute of Standards and Technology (NIST) og Gartner, Inc. sine fremtredende fordi de er ledende institusjoner innenfor vitenskap og teknologi.

NIST [2] sin definisjon er denne:

"... a model for enabling convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction"

Gartners [3] beskrivelse lyder:

"A style of computing where massively scalable IT-enabled capabilities are delivered as a service to external customers using Internet technologies"

Gartner fremhever også at følgende karakteristika ved nettskyen er spesielt vesentlige:

- Tjenestebasert
- Høyt skalerbar og elastisk
- Delt blant flere brukere
- Faktureres etter bruk (offentlige)
- Vidtstrakt bruk av internetteknologi

Fordeler som kjennetegner nettskyen er blant annet følgende:

- Fleksibilitet, med hurtig og enkel utrulling av ønskede tjenester
- Ofte reduserer utrulling av nettskyer IT-budsjetter
- Fokus på ny teknologi
- Standardisering av teknologi oppmuntres og tilrettelegges

Fra dette kan man trekke ut hovedessensen i hva nettskyer er: tjenestebaserte teknologier som kan skalere bredt, deles med et ukjent antall andre mennesker, blir fakturert etter bruk og er kraftig understøttet av internetteknologier som TCP/IP. Brukere skal kunne få levert ønskede tjenester (være seg virtuelle maskiner, applikasjonsrammeverk eller applikasjoner) idet de får behov for disse. For brukerene fremstår den som en "black box" [1], hvilket vil si at det ikke kommer frem hvilke ressurser man faktisk benytter seg av og hvor de fysisk blir hentet fra.

Med tanke på hvor mange ulike definisjoner som eksisterer, dekker imidlertid ikke de overnevnte beskrivelsene alle kvalitetene som kan eksistere innenfor teknologien. Dette har mye med at den fremdeles er under rivende utvikling og modning.

Det finnes et stort antall eksempler på nettskyteknologi som benyttes i stor skala den dag i dag. Fra Amazon Web Services [2] kan brukere på enkelt og hurtig vis få levert virtuelle maskiner som kjører på deres infrastruktur; Google App Engine [3] lar utviklere av applikasjoner fokusere helhjertet på å

[1] http://en.wikipedia.org/wiki/Black_box/
[2] http://aws.amazon.com/
[3] http://developers.google.com/appengine/

produsere programmer uten å tenke på hva slags maskinvare eller infrastruktur disse skal kjøre på, og Google Mail [4] lar millioner av brukere aksessere sin Google e-postkonto fra Google sin nettsky.

Eksemplene over beskriver i tillegg de tre eksisterende anerkjente nettskynivåene, henholdsvis "IaaS", "PaaS" og "SaaS". Hvor stort ansvar for administrasjonen av systemet kunden har avhenger av hvilken av de tre skymodellene, definert av NIST, som benyttes.

Infrastructure as a Service (IaaS)

"Infrastructure as a Service" innebærer at kunden blir tildelt en grunnleggende infrastruktur av (som oftest virtuelle) servere, lagring og nettverk. Dette er det laveste nivået i nettskyen, og brukeren gis mulighet til å velge sitt eget oppsett av plattform med operativsystemer og tjenester. Det er på dette nivået brukeren får tilgang til å opprette virtuelle maskiner med maskinvare og operativsystem etter behov. Offentlige eksempler er blant annet tidligere nevnte Amazon Web Services og Rackspace [5].

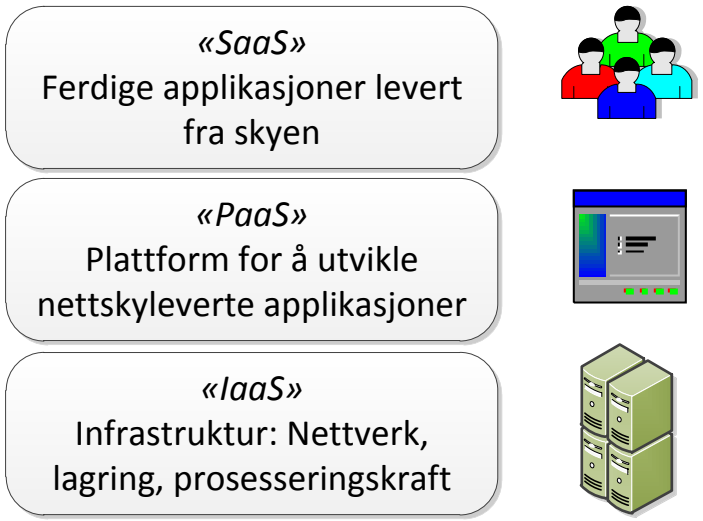
Platform as a Service (PaaS)

"Platform as a Service" er posisjonert på nivået direkte over "IaaS", hvor skytilbyderen tilbyr brukeren en plattform med for eksempel operativsystemer og verktøy for å kunne utvikle systemer og programmer. Et godt eksempel er tidligere nevnte Google App Engine.

Software as a Service (SaaS)

Det øverste nivået er "Software as a Service", der kunden har tilgang til tjenester og applikasjoner som står klare til bruk, ofte gjennom en nettleser. Et eksempel på dette er Google Docs [6], der Google står for all infrastruktur og lagring, utviklingsplattformen og selve kontorprogrampakken. Dermed må ingenting relatert til "Google Docs" installeres eller vedlikeholdes av sluttbrukeren, som kun behøver tilgang til en datamaskin med internett og en tilstrekkelig oppdatert nettleser. Ved bruk av "SaaS" vil det i tillegg være mindre behov for at hver enkelt bruker sitter på kraftige lokale maskiner ettersom mye av prosesseringen vil skje på serversiden inne i skyen.

[4] http://www.gmail.com/
[5] http://www.rackspace.com/
[6] http://docs.google.com/



Figur 2.1: Ulike nettskynivåer

Offentlige og private skyer

Videre skilles det mellom offentlige og private nettskyer, i tillegg til overnevnte nettskynivåer. Offentlige nettskyer, som Amazon sin Elastic Compute Cloud (EC2), leveres av selskaper som baserer seg på å selge skytjenester til eksterne bedrifter og/eller enkeltpersoner over Internett. En privat sky derimot, settes opp av ett enkelt selskap, en organisasjon eller en institusjon for å kunne kontrollere alle aspekter rundt implementasjon, drift og vedlikehold; av sikkerhetshensyn eller fordi de ønsker spesiell funksjonalitet som ikke tilbys av offentlige tilbydere. Noen karakteristika for de to alternativene er avspeilet i tabell 2.1.

	Privat	Offentlig
Ytelse	Kontrollen over ressursene er større, og man har full oversikt over last på servere.	Man har ikke nødvendigvis kontroll over ressursene som servere plasseres på, noe som kan føre til større ytelsesvariabilitet og mangel på oversikt.
Drift	Dersom interne ressurser må benyttes til å drifte skyen, kan dette være en kompliserende faktor og risiko. Hva gjør man om vedkommende slutter?	Håndteres av den eksterne skyens egne ansatte.
Sikkerhet	Håndteres av egne interne ressurser.	Håndteres av den eksterne skyens egne ansatte.

Tabell 2.1: Forskjeller mellom private og offentlige nettskyer

SkyHiGh

Den tenkte implementasjonen av SkyHiGh befinner seg på "IaaS"-nivået, og karakteriseres av å være en privat sky ment for intern bruk på Høgskolen i Gjøvik av ansatte eller studenter som behøver virtuelle maskiner. Brukere skal kunne starte virtuelle maskiner etter behov, med de ressurser og karakteristika som må til for å understøtte oppgavens krav, være seg forskningsrelaterte eksperimenter som krever ren og skjær prosesseringskraft, studenter som raskt trenger en maskin med et spesielt operativsystem for å kunne jobbe med en emnespesifikk øvingsoppgave eller annet.

Virtualisering av datamaskiner

Nettskyen er nærmest uløselig knyttet til konseptet virtualisering. Virtualisering er et vidt begrep og kan favne flere områder, blant annet hele systemer, nettverk og lagring. I konteksten SkyHiGh dreier det seg om at man har én eller flere fysiske maskiner som hver kjører flere virtuelle maskiner [4], hvor en virtuell maskin er en komplett datamaskin som blir emulert ved hjelp av programvare. I nettskyen vet man imidlertid ikke nødvendigvis noe om hvilken fysisk maskin de allokerte ressursene (for eksempel en virtuell maskin) havner på, og hvilke andre virtuelle maskiner eller tjenester som også kjører på denne fysiske maskinen og hvordan de påvirker hverandres ytelse.

Historien til virtualisering går helt tilbake til stormaskindriftstiden (1960- og 1970-tallet) [5], hvor det eksisterte et behov for å isolere brukere fra hverandre. Dette førte til utviklingen av "time-sharing"-systemer som gav

brukere mulighet til å dele disse kostbare stormaskinene. Etter hvert ønsket imidlertid brukerene å kjøre forskjellige operativsystemer på den samme maskinvaren, et krav som kun kunne tilfredsstilles ved hjelp av virtualisering, noe som etablerte virtuelle maskiner for System/360-plattformen i 1972. Selv om IBM oppfant teknologien for mange tiår siden, ble den ikke "gjenoppfunnet" før sent på 1990-tallet. Siden den gang har den vært åsted for mye forskningsarbeid og videreutvikling.

Virtuelle maskiner kjører et operativsystem separat fra den fysiske maskinvaren de i realiteten kjører på, og fremstår som om de var reelle fysisk maskiner, hver med en gitt form og mengde for prosesseringskraft, minne, lagringskapasitet og nettverkstilkobling som brukeren helt eller delvis definerer. Den virtuelle maskinen kan presentere et annet syn på maskinvaren enn det som faktisk er tilgjengelig rent fysisk.

Gitt oppgavens kontekst er det også nødvendig å forstå hva som driver de virtuelle maskinene. Dette er hypervisorens [6] oppgave, også kalt en "virtual machine monitor". Hypervisoren fungerer som et abstraksjonslag som legger seg mellom den fysiske maskinvaren og de virtuelle maskinene og håndterer delingen (multiplexingen) av de fysiske ressursene. I praksis utfører hypervisoren oppgavene (systemkallene) som gjesteoperativsystemene (de virtuelle maskinene) ber den fysiske maskinvaren om, hvilket de tror de er alene om å kontrollere.

Populære fri kildekode-hypervisorer er Xen [7] og KVM/QEMU [8], mens to av de mest kjente proprietære teknologiene tilhører VMWare [9] og Microsoft med sin sin Hyper-V [10]. De to aktuelle hypervisorene for SkyHiGh og OpenStack er imidlertid KVM og Xen. VMWare er en mulighet, men fordrer lisensbetaling. Hyper-V er ikke mulig å bruke i en OpenStack-implementasjon.

Full virtualisering og paravirtualisering

Full virtualisering vil si at hypervisoren emulerer i programvare den maskinvaren (for eksempel et RTL8139 nettverkskort) som blir tildelt gjesten. Fordelen med dette er at den virtuelle maskinen, også kalt gjesten, kan kjøre hvilket operativsystem som helst, siden emuleringen på hypervisoren tar seg av kompatibiliteten mellom maskinvare og operativsystem [7]. Ulempen med denne typen virtualisering er at hypervisoren må fange alle systemkall som fremmes av gjesten, noe som skaper "overhead" og således forlenget utførelsestid.

[7] http://xen.org/
[8] http://www.linux-kvm.org/
[9] http://www.vmware.com/
[10] http://www.microsoft.com/en-us/server-cloud/hyper-v-server/

For å komme rundt dette kan man bruke paravirtualisering, som innebærer at gjesteoperativsystemet blir gjort oppmerksom på at det er virtualisert. Dette kan gjøres ved å modifisere eller installere drivere i operativsystemet som snakker med et paravirtualiserings-API på hypervisoren. Resultatet blir at gjesten kan samarbeide med hypervisoren for å mer effektivt kunne kommunisere med den underliggende fysiske maskinvaren.

Hypervisor

KVM

KVM er en Linux-kjernemodul og virtualiseringsløsning som gir mulighet for full virtualisering av gjestemaskiner, og støtter virtualisering av et stort antall gjesteoperativsystemer. I tillegg til full virtualisering har KVM også støtte for et API kalt VirtIO [8] som gir støtte for blant annet paravirtualisering av nettverkskort og harddisker og er støttet i Linux fra og med 2.6.25-kjernen. For Windows-gjester må man laste inn en egen Virtio-driver sammen med installasjonen av operativsystemet. KVM benyttes som virtualiseringsplattform i den endelige implementasjonen av SkyHiGh.

Xen

Xen [9] er en hypervisor som i likhet med KVM støtter både full virtualisering og paravirtualisering av gjester. Administrasjon foregår ved hjelp av en priviligert virtuell maskin kalt "Dom0", som tar seg av resten av gjestene, kalt "DomU". Xen var opprinnelig ment å skulle brukes i SkyHiGh, men ble etter hvert byttet ut med KVM grunnet flere utfordringer ved forsøkt implementasjon av OpenStack med Xen som bærende hypervisor. En av utfordringene er at OpenStack må installeres på innsiden av en virtuell maskin i Xen, altså som "DomU". I tillegg var mangel på dokumentasjon fra OpenStack sin side en avgjørende faktor for overgang til KVM.

KVM vs Xen

Deshane et al. [10] påviste i 2008 forskjeller mellom KVM og Xen. KVM ble demonstrert å skalere dårligere enn Xen, mens Xen hadde noe mindre ytelsesisolasjon, altså hvor mye de individuelle virtuelle maskiner påvirker hverandre. KVM utmanøvrerte Xen på I/O-ytelse, mens Xen hadde bedre ytelse når det kom til prosessering (kompilering av Linux-kjernen). Hvilken hypervisor som velges kan tydelig ha stor påvirkning på ytelsen til de individuelle virtuelle maskinene og stabiliteten og integriteten til alle gjestene som befinner seg på den enkelte fysiske maskin.

Fordeler og ulemper ved virtualisering

Det er en rekke fordeler [11] som kan fremkomme når man benytter seg av overnevnte form for virtualisering, blant annet:

- Økt utnyttelse av eksisterende fysisk maskinvare
- Færre fysiske maskiner (konsolidering)
- Kostnadseffektivitet (utnyttelsen fører til at nye innkjøp av maskinvare kan utsettes og man trenger muligens færre fysiske maskiner når man først gjør innkjøp)
- Raskere utrulling av tjenester (fra bruker x ønsker tjeneste y, til tjenesten er operasjonell)
- Virtuelle maskiner kan benyttes til testing uten konsekvenser, før man faktisk begynner utrulling på servere i produksjon
- Enkel migrering og kloning av eksisterende virtuelle maskiner

Det eksisterer også noen ulemper, hvorav sistnevnte er selve problemområdet for denne bacheloroppgaven:

- Isolering av feil (dersom den fysiske maskinen svikter vil alle de virtuelle maskinene som kjører på maskinvaren bli rammet). Disse kan imidlertid fort bli startet på en annen fysisk maskin grunnet fleksibiliteten til virtuelle maskiner, gitt at man har en sikkerhetskopi
- Ytelsen på en virtuell maskin med en viss maskinvare tilsvarer ikke nødvendigvis det ikke-virtualisert fysisk maskinvare klarer å levere

Lagringsenheter er spesielt utsatt for forringing av ytelse ved virtualisering. Harddisk-I/O har alltid vært et av de største ankepunktene ved virtualisering, noe som blant annet kommer av måten en VM får presentert disken på. En vanlig måte å gjøre dette på er at vertsnoden, eller hypervisoren, gir den virtuelle maskinen en fil på en fysisk lagringsenhet som presenteres til den virtuelle maskinen som en blokkenhet, altså tror den virtuelle maskinen at den skriver til og fra sin egen fysiske disk, mens det i realiteten kun er en flat fil som ligger i vertens filsystem.

Én faktor som kan spille inn når den virtuelle maskinen befinner seg på en flat fil, er at det vil finnes to filsystemer oppå hverandre fordi både hypervisoren og den virtuelle maskinen begger skriver på sine respektive disker. I studien "Understanding Performance Implications of Nested File Systems

in a Virtualized Environment" [12] ble det påvist store ytelsespåvirkninger avhengig av hvilke filsystemer man velger å bruke, hvorvidt det er det samme filsystemet både på den virtuelle maskinen og hypervisoren, og om det velges å bruke nøstede filsystemer i det hele tatt. Tradisjonelt sett velges filsystem etter forventet arbeidsbelastning, men studien viser at man også burde vurdere om man i det hele tatt skal ha filsystem i begge lag, fordi det gir ytelsesmessige konsekvenser avhengig av om arbeidsbelastningen stort sett er skrivesentert, lesesentrert eller mest avhengig av lave ventetider. I tillegg nevnes det at det finnes stort rom for optimalisering av ytelse ved konfigurering og justering av filsystemparametere på hypervisoren, fordi det er der I/O-forespørslene ender opp.

Virtualisering av lagring

Virtualisering kommer som sagt i flere former. En annen av disse er virtualisert lagring. Dette gjøres på tilsvarende måte som med systemvirtualisering, ved at man tar en enhet (harddisk, RAID-volum) og deler (multiplexer) den opp i flere ressurser (Logical Volume Management), eller ved at man samler mange fysiske ressurser til én virtuell enhet (RAID-nivå 1/6 e.l) som selvsagt deretter kan deles i mindre enheter. De bakenforliggende fysiske enhetene er abstrahert bort slik at sluttbrukeren ikke vet noe om strukturen annet enn de logiske enhetene som blir tildelt.

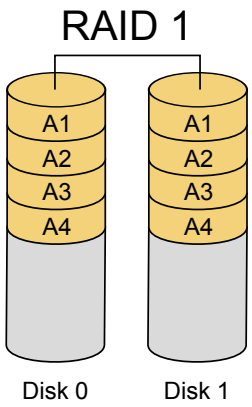
RAID

"Redundant Array of Independent Disks" [13] er en samling teknologi som benyttes for å samle flere fysiske disker til én logisk enhet. Det finnes ulike metoder å gjøre dette på, kalt RAID-"nivåer", og det er forskjellige fordeler og ulemper involvert med hvert av disse nivåene. Videre gjennomgås nivåene som er implementert i OpenStack-infrastrukturen i SkyHiGh.

RAID 1

På RAID-nivå 1 har man to harddisker som er identiske kopier av hverandre. Dette nivået kalles derfor speiling og fordelene er blant annet redundans (ryker én disk vil man fremdeles ha én tilbake med det samme innholdet) og økt lesehastighet [13]. Skrivehastigheten er typisk litt lavere siden alt må skrives til to disker. I SkyHiGh benyttes RAID 1 som systemdisk i controllernoden, compute-nodene og lagringsnoden. Se figur 2.2 [11] for illustrasjon av RAID 1.

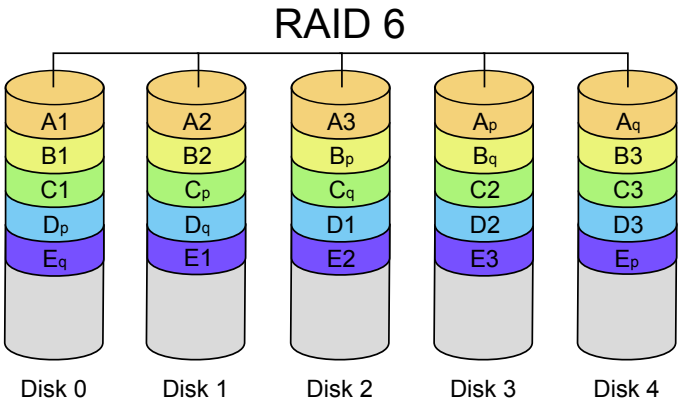
[11] http://en.wikipedia.org/wiki/File:RAID_1.svg lisensiert under Creative Commons



Figur 2.2: To harddisker i RAID 1

RAID 6

Dette nivået vil finnes på lagringsnoden som skal tilby "nova-volume"-funksjonalitet, som gir muligheten til å tilknytte ytterligere logiske lagringsenheter for virtuelle maskiner i OpenStack, samt lagring av virtuelle maskiner (som flate filer eller med en partisjon eller et LVM-volum som sitt rotfilsystem). Med RAID 6 skrives paritetsinformasjon til to disker blant de tilgjengelige enhetene, slik at raidet kan tåle at inntil to disker faller fra uten at noen informasjon går tapt. RAID 6 behøver et minimum av 4 disker. Dette vil si at av "n" disker (hvor n >= 4), kan 2 ryke og all data er fremdeles intakt. Siden dataene spres utover diskene, er fordelene med dette nivået økt lesehastighet og feiltoleranse, mens skrivehastighet kan bli noe straffet fordi paritetsinformasjon skrives til to disker, som skaper såkalt "overhead". Se figur 2.3 [12] for illustrasjon av RAID 6.



Figur 2.3: Fem harddisker i RAID 6

[12] http://en.wikipedia.org/wiki/File:RAID_6.svg lisensiert under Creative Commons

LVM

"Logical Volume Management / Manager" [14] er et begrep som finnes i de fleste moderne operativsystemer. LVM er et tynt abstraksjonslag som kan lage logiske enheter av grupper av fysiske disker. For OpenStack sin del benyttes LVM i forbindelse med EBS-ekvivalenten "nova-volume". Det anbefales å lage en volumgruppe, altså en logisk enhet fra for eksempel et RAID, til dette formålet.

Problemområdet

Selv om det er flere komponenter ved en infrastruktur enn I/O (disk og nettverk), er det disse elementene oppdragsgiver ønsker å få utredet i en OpenStack/nettsky-kontekst, og ikke ytelse relatert til prosessering og minne. Derfor er denne oppgaven helhjertet fokusert mot disk- og nettverksytelse. Trenger man mer prosessorkraft eller minne er det ofte aktuelt å oppgradere fra tokjernede til firekjernede prosessorer, sette inn ekstra minnemoduler, eller kjøpe flere fysiske servere, for å kunne tilfredsstille maskinvarehungeren til virtuelle maskiner. Dette gir heller ikke nødvendigvis tilsvarende ytelse for en virtuell masin som i en fysisk ekvivalent på grunn av omveier og overhead. I forhold til disk og nettverk har man også konfigurasjonsmuligheter og alternativer.

Input/Output

Med "Input/Output" menes i denne bacheloroppgavekonteksten primært disklagring og nettverk, og ytelsen til disse: hastighet målt i antall MB skrevet eller lest per sekund, hastighet i antall Mbit overført eller sendt per sekund, og responstid oppgitt i millisekunder. Hovedsakelig har man to forskjellige typer applikasjoner: I/O-bundede og CPU-bundede [15]. At en applikasjon er CPU-bundet vil si at den bruker brorparten av utførelsestiden i prosessoren, altså at det er prosessoren som eventuelt vil bli en flaskehals, mens I/O-bundet da nødvendigvis betyr at utførelsestiden er avhengig av hvor mye tid applikasjonen må vente på I/O, hovedsakelig harddisk-I/O, da dette typisk er den tregeste I/O-enheten i de fleste systemer. Typisk vil en CPU-bundet applikasjon gå raskere jo kraftigere prosessor som benyttes og en I/O-bundet applikasjon vil bli raskere med en hurtigere harddisk i systemet.

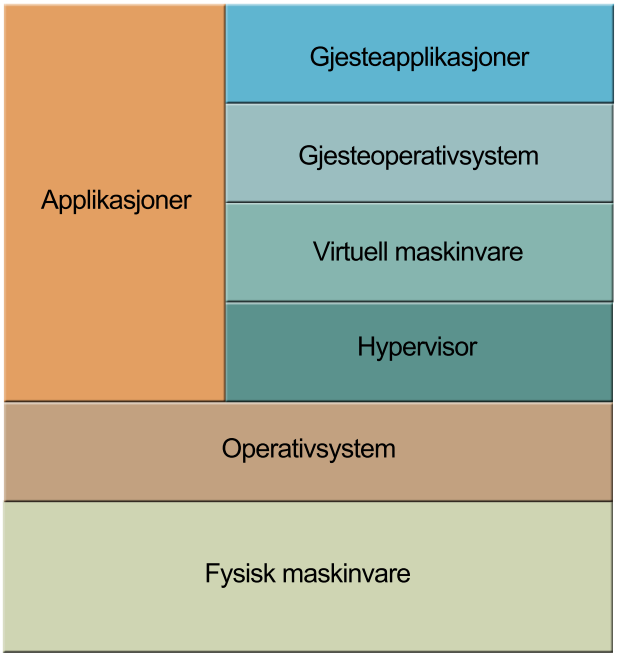
I/O på virtuelle maskiner - lag på lag

Input/output-ytelse har historisk sett vært en utfordring for virtuelle maskiner. Når en fysisk maskin kjører flere virtuelle maskiner deler hypervisoren den fysiske maskinvaren (prosessor, minne, I/O-enheter) inn i et tilsvarende antall logiske kopier, som operativsystemet i de enkelte virtuelle maskinene oppfatter som om skulle være fysisk tilknyttet seg. Dette kan skape køer og omveier som øker responstiden og reduserer ytelsen betraktelig.

Selv om virtualisering har mange fordeler er det også mange potensielle utfordringer. I forhold til I/O kan virtualisering introdusere ekstrabelastninger (overhead) som følge av de mange abstraksjonslagene som skapes. Rosenblum et al. [16] nevner at både responstid, hastighet på overføring og påvirkning av CPU forringes når I/O virtualiseres på grunn av omveien som informasjonen må gjennom, og at tidlige virtualiseringsløsninger hadde I/O-ytelse som bare nådde opp til omtrent halvparten av det man kunne forvente av den fysiske maskinvaren de virtuelle maskinene kjørte på. Dette vil si at en disk som kunne levere 100MB/s på den fysiske maskinvaren, kun nyttiggjorde 50MB/s i en virtuell maskin.

I artikkelen kommer det overfladisk frem de mange trinnene som hver enkelt I/O-forespørsel skal gjennomgå. Fra en virtuell maskin fremmer en slik forespørsel (for eksempel å skrive til eller lese fra en fil, via et systemkall), blir den prosessert av I/O-"stacken" til den virtuelle maskinen, hvorpå driveren (nettverks- eller diskdriveren) sender forespørselen til den logiske I/O-enheten. Denne forespørselen blir følgelig fanget opp av hypervisoren. Hypervisoren klargjør denne IO-forespørselen ved å legge den i en kø av mange for den faktiske, fysiske IO-enheten hos den fysiske maskinen. Når forespørselen er fullført, må denne sekvensen gjennomgås i omvendt rekkefølge.

I figur 2.4 vises de forskjellige overordnede lagene ved virtualisering med KVM som hypervisor (type 2). KVM-virtualisering vises til høyre i figuren, sammenlignet med en fysisk maskin til venstre.



Figur 2.4: Sammenligning mellom fysisk og virtuell maskin

I praksis finnes mange teknikker for å optimalisere rekkefølgen og antallet forespørsler totalt sett. Siden maskinene deler på den fysiske maskinvaren, er det også viktig at det er en viss rettferdighet med tanke på ressursallokering, slik at ikke én virtuell maskin monopoliserer alle tilgjengelige ressurser av en viss type (for eksempel nettverk): Quality of Service (QoS) er derfor en viktig del av hypervisorens arbeid.

Karakteristika for mekaniske harddisker

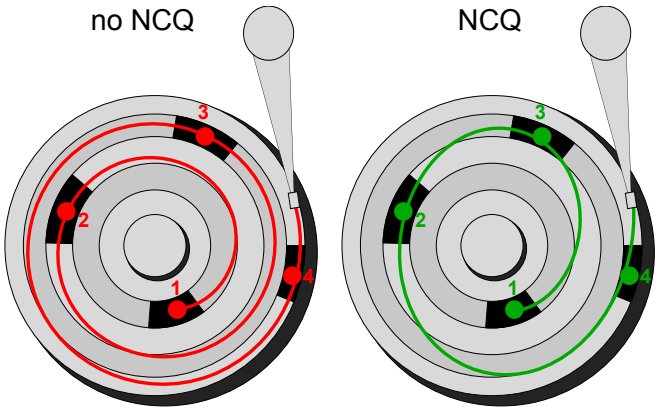
Konvensjonelle harddisker har en magnetplate som spinner rundt med en viss hastighet, ofte i hastigheter på 7200 eller 10000 RPM, mens et lesehode flyttes frem og tilbake over denne platen for å skrive og lese. Aksesstiden til en harddisk består av søketid (seek time), ventetid (latency) og eventuell ekstra tid brukt på kommando-overhead [17]. Søketid vil si den tiden det tar lesehodet å flytte seg til riktig posisjon på disken, mens ventetiden er den tiden det tar selve spindelen å rotere slik at riktig fysisk del av platen blir liggende under lesehodet.

Lesing og skriving på en harddisk deles opp i to kategorier hvor den ene er sekvensiell aksess mens den andre er tilfeldig aksess. Ved sekvensiell aksess får lesehodet mindre jobb fordi rekkefølgen på de fysiske dataene er etter

hverandre på disken, og dermed faller søketidparameteren bort. Det eneste som trengs er at spindelen roterer. Det er sjelden ren sekvensiell aksess forekommer, men ved for eksempel skriving av filer på flere GB kan man få noe i nærheten av sekvensiell skriving [18].

Ved tilfeldig aksess derimot kreves det at lesehodet kontinuerlig flytter seg til riktig posisjon på disken, noe som kan føre til mye større forsinkelser i aksesstiden enn ved sekvensiell aksess med påfølgende lavere overføringshastigheter. Selv om lesing og skriving av store filer kan se ut som en typisk sekvensiell jobb, så kan swapping fortsatt gjøre dette til mer tilfeldig aksess, da deler av filen vil bli flyttet rundt på disken. Det er denne typen aksess som skjer oftest ved "normal" desktop-bruk, der man ofte trenger aksess til mange forskjellige filer og applikasjoner som ikke befinner seg på samme fysiske lokasjon på harddisken.

Det er utviklet algoritmer for måten lesehodet flytter seg på for å forsøke å kutte ned på aksesstiden ved tilfeldig aksess på harddisker, hvorav Native Command Queuing (NCQ) [19] er en av dem. NCQ er en avansert algoritme for SATA-harddisker som utnytter det faktum at en harddisk som regel har flere oppgaver liggende i køen som venter på å bli utført, og dermed kan NCQ planlegge hvilke kommandoer den skal utføre i en gitt rekkefølge, noe som minsker avstanden lesehodet må flytte seg mellom hvert punkt. NCQ ble implementert i SATA II og skal derfor være i bruk på de aktuelle diskene i infrastrukturen tilgjengelig. Se figur 2.5 [13] for illustrasjon av NCQ.



Figur 2.5: Illustrasjon av NCQ

[13]http://en.wikipedia.org/wiki/File:NCQ.svg lisensiert under Creative Commons

iSCSI

iSCSI-protokollen [20] spesifiserer hvordan man snakker med SCSI-enheter over et TCP/IP-nettverk, for eksempel over Internett. Det er en IP-basert SAN-løsning som presenterer blokkenheter hos klienter som fremtrer som om de faktisk var lokale fysiske disker montert i maskinen.

Tidligere satte begrensninger i oppnåelig båndbredde for datanettverk en stopper for praktisk implementering av slike løsninger. Det er imidlertid blitt stadig mer attraktivt ettersom nettverksstandarder som 1GBit, 10Gbit og 100Gbit-Ethernet har blitt utgitt. 1Gbit er i dag en utbredt og omfavnet teknologi, og 10Gigabit er forventet å bli tilgjengelig i større grad innen kort tid.

Konkurrentene til iSCSI er blant annet FC (Fiber Channel) og AoE (ATA over Ethernet). Blant fortrinnene som iSCSI har over disse er at det kan rulles ut på komponenter og teknologi som er vidt tilgjengelige, slik som Ethernet, og man trenger derfor ikke å ha flerfoldige typer nettverk slik som er tilfellet med FC. Dette gjør iSCSI til en fleksibel, kostnadsbesparende og allsidig skikkelse i SAN-markedet.

iSCSI sender pakker med SCSI-kommandoer for å få utført operasjoner. Eksempler på dette er å lese og skrive fra lagringsenheten. iSCSI bruker i tillegg TCP-protokollen til kommunikasjon. Dette fører til forutsigbarhet og pålitelighet i form av følgende fordeler:

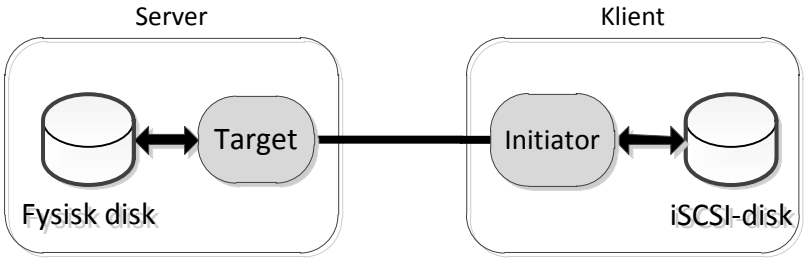
- Pakker kommer frem i riktig rekkefølge
- De pakkene som skulle gå tapt blir sendt på nytt
- Fungerer på mange forskjellige infrastrukturer, i tillegg til å være godt forstått og etablert som transportlagprotokoll
- Protokollen har også mekanismer som trår til hvis det skulle bli opphopninger i nettverket

iSCSI fungerer i en klient-server-kontekst. Klienter (kalt initiators) kobler seg til en lagringsenhet på en server (kalt target). Etter at klienten er autentisert, dersom dette er påkrevet, får klienten frem lagringsenheten. Dersom blokkenheten aldri er benyttet tidligere, må den formateres med filsystem etter eget ønske og monteres før den er klar til bruk.

Ytelse i iSCSI avhenger av flere faktorer:

- Tilgjengelig båndbredde på nettverket

- Noe overhead må påregnes i form av kontrollpakker for SCSI-kommandoer (for eksempel skrive- og lesekommandoer)
- Trolig må man ha flere TCP-tilkoblinger for hver klient mot lagringsserveren for å kunne oppnå maksimal hastighet



Figur 2.6: Illustrasjon av iSCSI

Konfigurasjonsvalg

For blokkenhetene som iSCSI deler har man en rekke konfigurasjonsalternativer, blant annet for hvor mange "tråder" som skal eksistere per tilkobling mellom "initiator" og "target". Standard antall tråder er 8, men gruppen vil undersøke med 16 for å se om det gir avkastning i form av forbedret ytelse.

Et annet vesentlig valg står mellom "blockio" og "fileio". Fileio bruker VFS-laget [14] i Linux for aksess til sitt target. Dette betyr at ved "fileio" vil dataene bli cachet i lagringsnodens hurtigminne [21], i motsetning til "blockio" som leser target-disken som en ren blokkenhet og dermed ikke utnytter denne muligheten. På grunn av cachingen bør "fileio" kunne gi bedre ytelse enn "blockio" ved gjentatt lesing av filer som er mindre enn lagringsnodens mengde RAM. Filer som har mye til felles skal også kunne få en slik hurtiglagringseffekt.

Bruksområde for SkyHiGh

"Nova-volume"-komponenten av OpenStack har anledning til å benytte seg av iSCSI for sin kommunikasjon mellom lagringsenhet og compute-noder,

[14] http://en.wikipedia.org/wiki/Virtual_file_system

noe som også anbefales av utviklerene. Denne modulen tar seg av iSCSI-kommandoer bak teppet for å tilknytte opprettede volum til kjørende virtuelle maskiner.

I SkyHiGh vil iSCSI også brukes til å oppbevare virtuelle maskiner på flate filer (kalt "images"), for å vurdere hvorvidt dette er en god løsning for mange virtuelle maskiner av gangen.

Oppsett av iSCSI for Linux-maskiner er beskrevet i vedlegg B.

Link Aggregation

Gigabit Ethernet (1Gbit) er etter hvert blitt standarden [22] for lokal dataoverføring på datamaskiner. Tidligere har 10- og 100MBit vært populære, og det er også ankommet etterfølgere i 10Gbit (10 000Mbit) og 100Gbit (100 000Mbit), selv om disse ennå ikke har tatt av i privatsjiktet av markedet, og er på vei opp og frem i organisasjoner.

Standard	Maksimal teoretisk hastighet
10Mbit	1.25MB/s (10 / 8)
100Mbit	12.5MB/s (100 / 8)
1000Mbit (Gbit)	125MB/s (1000 / 8)
10 000Mbit (10GbE)	1.25GB/s (10 000 / 8)
100 000Mbit (100GbE)	12.5GB/s (100 000 / 8)

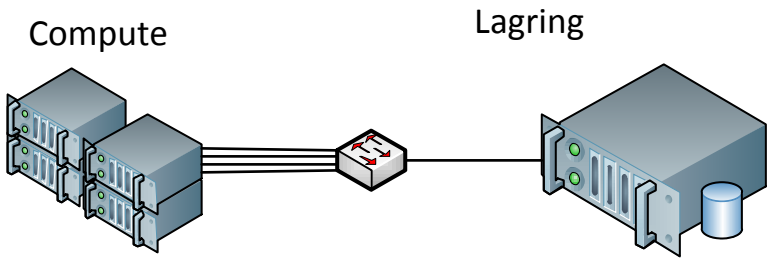
Tabell 2.2: Teoretisk oppnåelige dataoverføringshastigheter over Ethernet

Tabell 2.2 viser de maksimalt teoretisk oppnåelige dataoverføringshastighetene for Ethernet. I praksis vil hastighetene være lavere på grunn av pakker som må sendes på nytt samt polstring (overhead i form av data om sender, mottaker og lignende) i Ethernet-pakkene.

I mange sammenhenger kan flaskehalsen i en infrastruktur være hastigheten på dataoverføringene gjennom nettverket. I disse tilfellene kan man enten oppgradere maskinvare, ved å for eksempel gå fra Gigabit til 10Gbit på alle maskiner, switcher og rutere som trenger det. En annen mulighet er link aggregation, også kalt "trunking" eller "bonding" [23]. Aggregation er et paraplybegrep som omfatter at man sammenkobler to eller flere enheter til én enhetlig og kraftigere logisk enhet. I tilfellet link aggregation betyr det at man eksempelvis kombinerer to nettverkskort à 1Gbit for å oppnå en teoretisk hastighet på 2Gbit. Link aggregation har flere potensielle fordeler:

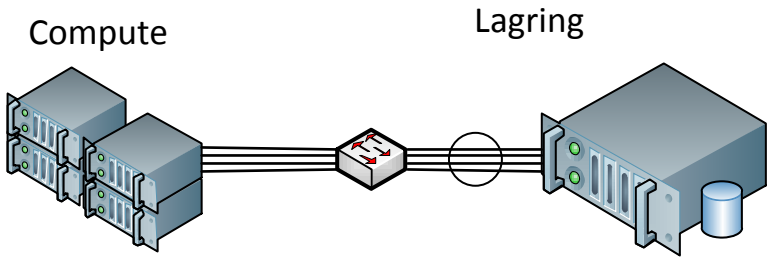
- Failover/redundans, hvilket vil si at nettverket forblir oppe dersom ett nettverkskort svikter. Dette fører til høyere grad av potensiell tilgjengelighet
- Lastbalansering for alle deltakende nettverkskort, slik at ikke ett av to Gbit-nettverkskort står for all lasten, dersom bare 50% av den aggregerte 2Gbit-forbindelsen blir utnyttet (altså ingen misforhold i deltakelse)

Eksemplene nedenfor viser hvorfor link aggregation kan være nyttig dersom man ikke har mulighet til å oppgradere til 10GbE over hele infrastrukturen:



Figur 2.7: Infrastruktur uten link aggregation

I figuren 2.7 ser man fire compute-noder koblet til én lagringsnode gjennom en Gbit-switch. Dersom hver node har Gbit-nettverkskort, vil hver compute-node maksimalt kunne oppnå 250Mbit/s med throughput (1000Mbit / 4).



Figur 2.8: Infrastruktur med link aggregation

Figur 2.8 viser et eksempel på link aggregation. Hver compute-node har i dette tilfellet mulighet til maks throughput til storage-noden, hvilket vil si 1Gbit/s på hver rent teoretisk. Lagringsnoden har koblet fire Gbit-nettverkskort til switchen, hvilket gir det resulterende logiske nettverkskortet en teoretisk dataoverføringshastighet på 4Gbit.

Oppsett av link aggregation for Linux-maskiner er beskrevet i vedlegg B.

Jumbo Frames

Jumbo frames defineres som Ethernet-frames hvor størrelsen på Maximum Transmission Unit (MTU) overstiger standardstørrelsen på 1518 bytes [24] helt opp til maksimalstørrelsen 9018 bytes. En annen vanlig størrelse er 4048 bytes. Med standardstørrelsen så består enheten av en "payload" med faktiske nyttedata som overføres og en viss andel "overhead". For standardstørrelsen er payload-delen 1500 bytes, mens resten er overhead og ikke nyttedata. Når framestørrelsen øker, forblir overhead-delen statisk. Bruk av jumbo frames kan derfor ha følgende fordeler:

- Økning av andel nyttedata, som reduserer prosesseringsbehov
- Høyere utnyttelsesgrad av nettverket

Konfigurering av jumbo frames kan også føre med seg noen negative konsekvenser:

- Høyere risiko for korrupsjon av pakker, noe som krever sending av disse på nytt
- Lengre responstider (færre pakker totalt sett blir sendt)

Hva som passer best for hvert enkelt tilfelle er derfor ikke gitt: standardstørrelse, maksstørrelse eller en mellomverdi. Tester med flere alternativer må utføres for å avgjøre hvilken konfigurasjon som gir ønsket resultat.

Oppsett av jumbo frames for Linux-maskiner er beskrevet i vedlegg B.

Kapittel 3

OpenStack

Bakgrunn

Et forprosjekt utført i emnet "Systemadministrasjon" i løpet av høsten 2011 avgjorde at OpenStack var den mest fleksible plattformen å basere implementasjon av en nettsky for Høgskolen i Gjøvik, altså SkyHiGh, på, gitt kravene til oppdragsgiver om å få til flytting av virtuelle maskiner til og fra en nettsky etter behov.

OpenStack [25] er en samling open-source programvare som er utarbeidet for å muliggjøre utrulling av høyt skalerbare og funksjonsrike skyarkitekturer som er enkle å implementere for små såvel som store organisasjoner. Både private og offentlige skyer er målet for prosjektet, og det er i tillegg fullt kompatibelt med AWS/EC2 sin funksjonalitet. OpenStack kan bare implementeres på Linux, og utviklerene selv anbefaler at Ubuntu benyttes i utrullingen fordi utstrakt testing ikke har foregått på andre distribusjoner.

Prosjektet utvikles av en rekke solide og renommerte aktører, blant annet NASA og RackSpace, og er gjort fritt tilgjengelig under Apache 2.0-lisensen. Fremdeles er prosjektet nokså ferskt, men modnes og utvikles i veldig høyt tempo, uten tvil på grunn av de store bakenforliggende ressursene som understøtter prosjektets gang.

Det er fem sammenkoblede hovedprosjekter som utgjør OpenStack-arkitekturen:

- Compute (Nova)
- Storage (Swift)
- Imaging (Glance)
- Dashboard (Horizon)

- Identity (Keystone)

Disse står for henholdsvis prosessering, lagring, oppbevaring av "images" som inneholder en type operativsystem man kan bruke som mal for virtuelle maskiner, brukergrensesnitt for administrering og til sist autentisering. Videre vil noen av komponentene beskrives i mer detalj, blant annet "Compute" og "nova-volume", hvor sistnevnte tilbyr virtuelle maskiner permanent lagringsplass dersom det er behov for dette.

Compute

Compute (kodenavn Nova) er hovedkomponenten i Openstack, og kan sammenlignes med Amazonz Elastic Compute Cloud (EC2) [1]. Den organiserer opprettelsen, driften, vedlikeholdet og i siste instans termineringen av virtuelle maskiner ved å utføre en rekke oppgaver bak teppet som avhenger av hvilken hypervisor som benyttes. Hypervisoren er typisk Xen eller KVM, og valget kan være svært avgjørende for funksjonalitet og ytelse. Noen storstilt uttesting av andre hypervisorer er ikke utført på Openstack.

SkyHiGh ble forsøkt implementert med Xen og Linux-distribusjonen Debian etter ønske fra oppdragsgiver, noe som mislyktes på grunn av dårlig dokumentasjon. Deretter ble det avgjort at Ubuntu og KVM ville benyttes, noe som også er den anbefalte fremgangsmåten.

Storage

Storage, også kjent under kodenavnet Swift, er ekvivalenten til Amazons S3-løsning [2]. Den er designet for å tilby et veldig enkelt grensesnitt mot såkalt "blob storage", hvilket vil si at det legges opp til å være lagringsplass for "hva som helst". Gruppen kommer ikke til å se på denne delen av OpenStack i denne oppgaven ettersom det ikke vil implementeres av den parallelle bacheloroppgavegruppen som en del av SkyHiGh.

Nova-volume

Nova-volume står for opprettelse, tilknytting, frakobling og sletting av permanent lagringsplass for virtuelle maskiner (kalt instanser i OpenStack). Denne løsningen tilbyr sammenlignbar funksjonalitet som Amazons Elastic Block Storage [3] (EBS). Nova-volume kan benytte seg av blant annet iSCSI,

[1] http://aws.amazon.com/ec2/
[2] http://aws.amazon.com/s3/
[3] http://aws.amazon.com/ebs/

FiberChannel (FC) og ATA over Ethernet (AoE). Førstnevnte er imidlertid det som er mest vanlig. Med Nova-volume får instansene tildelt en blokkenhet som de deretter kan formatere og bruke som om det var en lokalt montert harddisk. Nova-volume kan ikke sammenlignes med et NAS, all den tid lagringsenhetene ikke kan deles og kun har anledning til å være tilknyttet én instans av gangen.

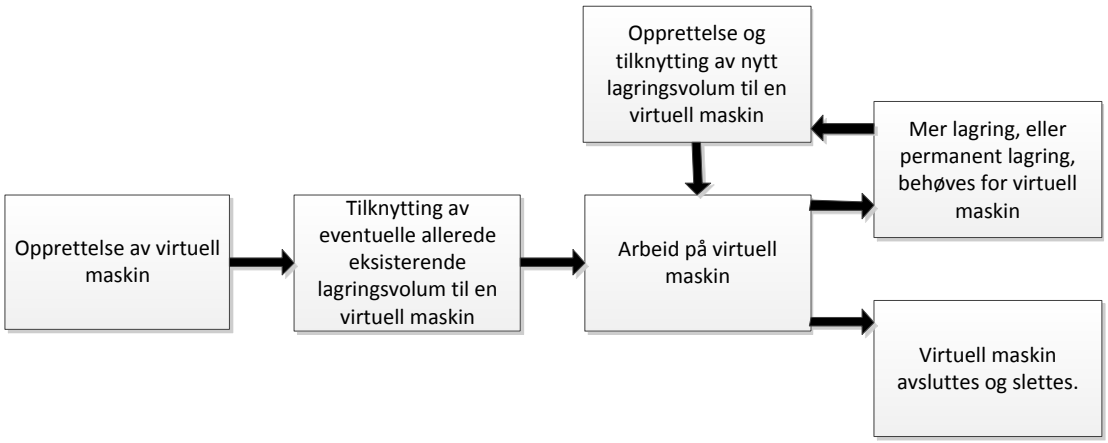
Lagringsplassen på de virtuelle maskinene som kjører på Openstack har i utgangspunktet begrenset levetid. Når kjørende instanser omsider slettes vil også alt som er lagret på det som representerer den lokale disken for instansen forsvinne (uavhengig av om dette er EBS eller en fil), med mindre det er tatt en form for sikkerhetskopi. Nova-volume kommer derfor til sin rett i de tilfeller hvor man har behov for å ha en mer vedvarende form for lagring. Etter at en instans slettes vil lagringsplass tildelt med nova-volume fremdeles være tilgjengelig, og kan tilknyttes andre kjørende instanser, eller en ny instans.

Gruppen vil ikke se på denne delen av OpenStack ettersom det ikke er en prioritert del av hva som implementeres av den andre bachelorgruppen som en del av SkyHiGh.

Figur 3.1: Samspill mellom LVM, RAID og Nova-volume

Arbeidsflyt

Forespeilet arbeidsflyt for en virtuell maskin i OpenStack er forsøkt illustrert i figur 3.2. Her observeres hvordan "nova-volume" deltar dersom mer, eller permanent lagringsplass behøves i en virtuell maskin.



Figur 3.2: Forespeilet arbeidsflyt i OpenStack

Lokal disk for virtuelle maskiner

Oppdragsgiver er interessert i informasjon om hvordan ytelsen til de virtuelle maskinene påvirkes av hvor de fysisk plasseres på disk i OpenStack-arkitekturen. Alt resterende innhold i dette kapittelet er basert på valget av KVM som hypervisor.

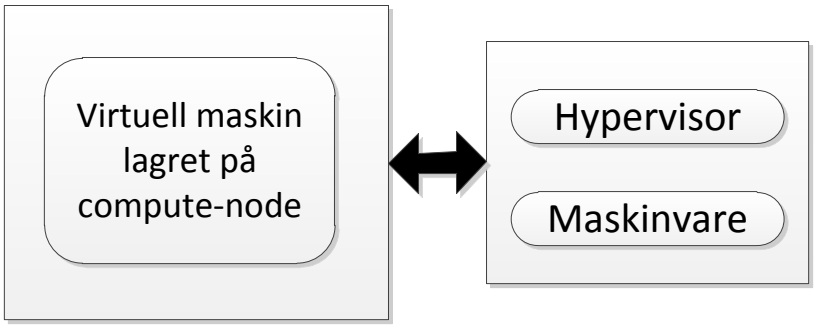
Virtuelle maskiner i OpenStack starter som en klone av en allerede ferdiglaget virtuell maskin av en spesiell type og starter denne med en viss maskinvare. Dette gjøres for å spare plass og for å gjøre utrullingen raskest mulig.

Det finnes mange ulike måter å lagre disse virtuelle maskinene på, som fil og lagret på enten compute-noden den kjører på, et SAN eller NAS, eller at den virtuelle maskinen får et rotfilsystem på et LVM-volum eller en ledig partisjon. Gruppen vil kun kjøre tester med virtuelle maskiner lagret som en fil på enten compute-nodens fysiske harddisk eller et iSCSI-target montert på compute-noden. Når disse lagres som fil, finnes det flere ulike formater.

I OpenStack anbefales det å bruke "QCOW"-formatet [4], men det eksisterer også flere som kan gi bedre ytelse grunnet mindre overhead. Dette formatet gjør at filene kun vokser etter behov, og er små til å begynne med hvilket gir rask utrulling.

Som fil på compute-noden

Compute-noder i OpenStack blir beordret av controller-noden til å starte virtuelle maskiner når brukere ber om dette. Standardoppsettet for OpenStack er at instansene lagres som filer lokalt på disse compute-nodene, i området `/var/lib/nova/instances`. Diskytelse for den virtuelle maskinen er dermed fullstendig avhengig av hvilken lagringsløsning som befinner seg på compute-noden den virtuelle maskinen blir oppført hos.



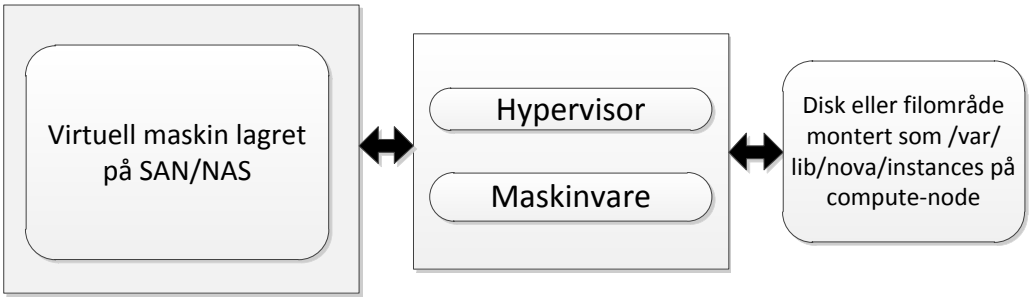
Figur 3.3: Compute-nodens disk som oppbevaringssted for instansfiler

Som fil på et SAN eller NAS

Filen som inneholder den virtuelle maskinen kan også oppbevares på et SAN eller NAS, hvor kommunikasjonen går gjennom lokale nettverk eller Internett. Et SAN [26] innebærer deling av lagringsenheter, hvor kommunikasjonen foregår ved hjelp av SCSI-protokollen over et nettverk. Dette er teknologier som iSCSI (Internet SCSI), FC (FiberChannel) eller AoE (ATA over Ethernet). Rå blokkenheter deles og det ser ut som om de skulle være tilknyttet lokalt hos datamaskiner som er tilkoblet SANet. NAS på den annen side, er

[4] http://people.gnome.org/ markmc/qcow-image-format.html

ren deling av filer ved hjelp av protokoller som NFS og CIFS. Disse filområdene eller lagringsenhetene må deretter monteres i `/var/lib/nova/instances` for at virtuelle maskiner skal oppbevares der.



Figur 3.4: SAN/NAS som oppbevaringssted for instansfiler

På EBS (nova-volume)

Amazons EC2-instanser har lenge kunnet bruke Elastic Block Storage som sitt rotfilsystem. Dette har kun vært mulig i OpenStack i form av et ustabilt og varierende "proof-of-concept" til nå, men vil etter alt å dømme bli implementert i fungerende stand i en fremtidig versjon. Med denne løsningen er ikke den virtuelle maskinen lenger en flat fil, men kjører fra en logisk lagringsenhet som "nova-volume" tildeler den. Denne funksjonaliteten har fremdeles ikke blitt rullet ut i versjonen av OpenStack som implementeres i SkyHiGh, og blir derfor ikke mulig å teste.

Beste praksis

IBM har undersøkt hvilke beste praksis-anbefalinger som foreligger for virtuelle Linux-maskiner når hypervisoren i infrastrukturen er KVM i en studie fra 2010 [7]. Her går de blant annet inn på hvilke fordeler og ulemper som eksisterer for henholdsvis virtuelle maskiner med rotfilsystem på en blokkenhet og virtuelle maskiner lagret som en fil.

Studien nevner at fordelene med virtuell maskin på fil er at de er enkle å migrere og gir god nok ytelse i de fleste tilfeller, men at den generelt sett er dårligere enn alternativet.

Å oppbevare virtuelle maskiner på en blokkenhet (LVM-volum eller partisjon) er ikke like fleksibelt i forhold til migrering, men gir best ytelse og lavest responstider. Den forbedrede ytelsen fremkommer som følge av at det blir færre kostbare programvarelag å måtte gjennom for hver gang I/O-forespørsler fremmes.

Gruppen har ikke klart å finne artikler eller forskningsarbeid som har sett på diskytelsen for virtuelle maskiner når disse lagres på iSCSI. I artikkelen "Revisiting the Storage Stack in Virtualized NAS Environments" (Hildebrand et al.) [27] kommer det derimot frem at det er en forholdsvis ny trend å plassere virtuelle maskiner på ekstern lagring, og de ser på ytelsen for virtuelle maskiner lagret på et NAS (NFS). Resultatene deres viser at det er veldig kostbart å plassere virtuelle maskiner på ekstern lagring fordi det introduserer ny overhead, men at resultatene avhenger av en rekke parametre, blant annet blokkstørrelse: dess mindre blokkstørrelse, dess dårligere ytelse, hvilket har med at det da blir flere forespørsler totalt sett.

Drivere for virtuelle maskiner

For både disk og nettverk har man et knippe aktuelle drivere å velge mellom [7]:

Diskdriver	Beskrivelse
IDE	Emulerer en IDE-disk
SCSI	Emulerer en SCSI-disk
Virtio	Paravirtualisert driver

Tabell 3.1: Mulige diskdrivere for virtuelle maskiner

Nettverksdriver	Beskrivelse
RTL8139	Standarddriver som emulerer et kort i RTL8139-klassen
E1000	Driver som emulerer et Intel Pro/1000Mbit nettverkskort
Virtio	Paravirtualisert driver

Tabell 3.2: Mulige nettverksdrivere for virtuelle maskiner

Hver compute-node har en mal for hvilke drivere den skal benytte seg av for nystartede virtuelle maskiner. Denne filen befinner seg i `/usr/share/pyshared/nova/virt/libvirt.xml.template` og man kan enkelt endre drivere til det med best ytelse. I oppgavens tester vil det for nettverk avgrenses til de overnevnte tre, selv om det finnes flere valgmuligheter. For disk vil det kun bli test med virtio [8] siden gruppen. Tidligere

studier [7] har vist at dette er det beste alternativet, og er i tillegg standard i KVM-malen. IDE og SCSI har en I/O-kø som er betydelig mindre og paravirtualiserte virtio gir både responstids- og båndbreddemessige fordeler.

Instanstyper og operativsystem

OpenStack kommer med fem forskjellige ferdigkonfigurerte instanstyper [28] fra installasjonen av. En instanstype – kalt "flavor" i OpenStack – er den virtuelle maskinvaren som en instans blir rullet ut med, det vil si maskinvarekarakteristika for en maskin, herunder hva den skal ha tilgjengelig av minne, prosessorkraft, lagringskapasitet og "swap".

Av infrastruktur- og tidsbegrensende årsaker har gruppen valgt å kun teste ytelsen på en modifisert versjon av instanstypen "m1.tiny". Begrensningene ligger i at compute-nodene har for svak maskinvare til å understøtte et stort antall virtuelle maskiner, blant annet grunnet kun 2GB minne. Den modifiserte instanstypen, kalt "skyhigh.tiny", er vist til slutt i tabell 3.3. Forskjellen fra "m1.tiny" ligger i økt mengde diskplass for å tilrettelegge for testing av diskytelse på filer som er store nok til at man ikke bare tester mellomlagring, men hva man faktisk kan forvente av diskytelse.

Instansnavn	Minne	Antall virtuelle prosessorer	Lagring
m1.xlarge	16GB	8	160GB
m1.large	8GB	4	80GB
m1.medium	4GB	2	40GB
m1.small	2GB	1	20GB
m1.tiny	512MB	1	0GB
skyhigh.tiny	512MB	1	10GB

Tabell 3.3: Utvalget instanstyper i OpenStack

Kapittel 4

Ytelsestesting og testverktøy

Det finnes et stort antall ytelsesmålingsverktøy som allerede er utviklet av andre. Disse befinner seg i hovedsaklig to hovedkategorier, herunder av typen syntetiske og realistiske ytelsestester. Litteratur rundt emnet er undersøkt for å finne relevante testverktøy. Innholdet i disse kildene avgjorde i stor grad hvilke testverktøy det ble valgt å gå for, i tillegg til mulighetene for scriptet automatisering.

Ytelsesmåling

Ytelsesmåling [1], bedre kjent på engelsk som "benchmarking", går i denne konteksten ut på å forsøke å vurdere hvordan en del av et datasystem yter. Dette foregår ved at man benytter spesialiserte programmer eller realistiske arbeidsbelastninger for å skape en spesifikk last på et system, og måler resultatene.

Syntetisk og realistisk ytelsesmåling

De syntetiske [29] testene utfører, som navnet foreslår, tester og operasjoner som ikke nødvendigvis er representative for det som ville tilsvart faktisk bruk av applikasjoner eller datamaskiner hos en sluttbruker. Syntetisk ytelsesmåling består av kunstige tester som forsøker å si noe om den potensielle og maksimale ytelsen som kan oppnås for en oppgave gitt en spesifikk konfigurasjon, gjennom innhenting av data som følge av kontrollerte eksperimenter. Alle testene i denne bacheloroppgaven faller inn under den syntetiske kategorien, hvilket er en svakhet, men som likevel kan brukes til å trekke slutninger for hva som muligens lønner seg.

[1] http://en.wikipedia.org/wiki/Benchmark_(computing)

Motsetningen til syntetiske tester er realistiske tester, som ofte har typiske brukerscenarier som gjennomgås og tester ytelsen for oppgaver som er noenlunde tilsvarende det en gjennomsnittlig bruker ville utført. Et eksempel på dette er et program eller script som automatisk starter et tekstbehandlingsprogram, skriver inn tekst med en hastighet som tilsvarer en sluttbrukers og lukker det igjen, mens data om diskytelse, minnebruk, CPU-last og andre relevante parametre overvåkes parallelt. Slike realistiske tester ble tatt i bruk i en bacheloroppgave [2] ved HiG fra 2010 skrevet av Kristoffer M. Elde og Øyvind Haugedal. I bacheloroppgaven ble det utviklet script for å skape belastning som skulle etterligne virkelig bruk på blant annet Apache og MySQL-servere kjørende på ulike typer hypervisorer.

Ønsker fra oppdragsgiver

Oppdragsgiver ytret tidlig et ønske om at verktøyet for diskytelsestesting skulle kunne vise responstider og tilfeldig lesing og skriving, i tillegg til de sedvanlige operasjonene sekvensiell lesing og skriving, noe som ville legge føringer for hvilke verktøy som var mulige å benytte. Fra tidligere hadde gruppen kjennskap til Bonnie++ fra det driftsspesifikke emnet "Database- og applikasjonsdrift", men var nødt til å undersøke litteratur og Internett for å se hvilke andre verktøy som anbefales for disse formålene.

Videre vil tre av de mulige testverktøyene for disk og to for nettverk gjennomgås for å undersøke hva de tilbyr.

Beskrivelse av ønskede målemuligheter

Sekvensiell og tilfeldig skriving og lesing

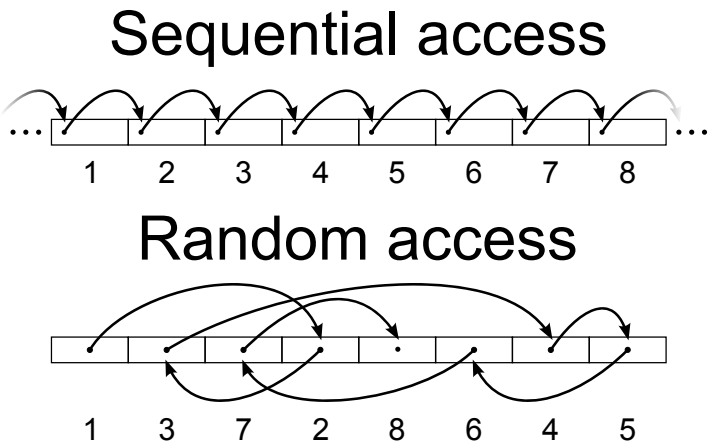
Sekvensiell skriving og lesing [30] betyr som tidligere nevnt at blokkene som utgjør filene på disk blir liggende sammenhengende på det fysiske lagringsmediet. Dette fører vanligvis til høyere ytelse enn tilfeldig lesing og skriving, da mindre tid blir brukt på å søke seg frem til områdene det skal arbeides på. Et eksempel på sekvensiell aksess er kompilering, hvor man går sekvensielt gjennom kildekode for å skape kjørbare programmer.

Tilfeldig lesing og skriving betyr at blokkene som utgjør filene på disk er plassert på en usammenhengende måte, noe som for mekaniske harddisker skaper lavere ytelse, ettersom det fører til søking til og fra fysiske områder i større grad enn når det arbeides sekvensielt. Det kan også bety at man leser

[2] http://brage.bibsys.no/hig/handle/URN:NBN:no-bibsys_brage_12612

eller skriver fra mange ulike filer som er spredt rundt på disken. Et eksempel på tilfeldig aksess er databaseoperasjoner, hvor det ikke er gitt at radene befinner seg sekvensielt etter hverandre.

Diskytelse oppgis i antall MB man er i stand til å lese eller skrive per sekund. Se figur 4.1 [3] for illustrasjon av sekvensiell og tilfeldig aksess av disk.



Figur 4.1: Forskjell mellom sekvensiell og tilfeldig aksess av disk

Responstid for diskytelse

Med responstid menes tiden det tar fra en I/O-oppgave forespørres til den er ferdig utført. For eksempel, skriver man en fil til disk sekvensielt med en blokkstørrelse på 4KB, er responstiden tiden det tar fra en slik blokk bes om å bli skrevet til det er utført.

Nettverkstester

Gruppen skal også teste nettverkshastigheten for virtuelle maskiner i infrastrukturen med ulike konfigurasjoner. Dette vil måles i antall Megabit man er i stand til å overføre per sekund, et mål på throughput - hvor fort data overføres. Responstid vil også testes for nettverket, hvilket betyr tiden det tar fra en pakke sendes til et svar mottas.

[3] http://en.wikipedia.org/wiki/File:Random_vs_sequential_access.svg lisensiert under Creative Commons

Undersøkelse av mulige verktøy

Bonnie++

Bonnie++ [31] er et mikroytelsesmålingsverktøy for harddisker og forskjellige filsystemer. Verktøyet er basert på programmet Bonnie, og ble til som et resultat av at utviklerene ikke klarte å bestemme seg for C++ eller C som programmeringsspråk for et samarbeid på et nytt verktøy. De store forskjellene mellom Bonnie og Bonnie++ er muligheten til å teste systemer med mer enn 2GB lagringsplass, i tillegg til implementasjon av flere tester, blant annet metadataoperasjoner som systemkallene *create()*, *stat()* og *unlink()*.

Verktøyet har mulighet til å teste ytelsen på tre generelle hovedområder for et filsystem eller en lagringsenhet:

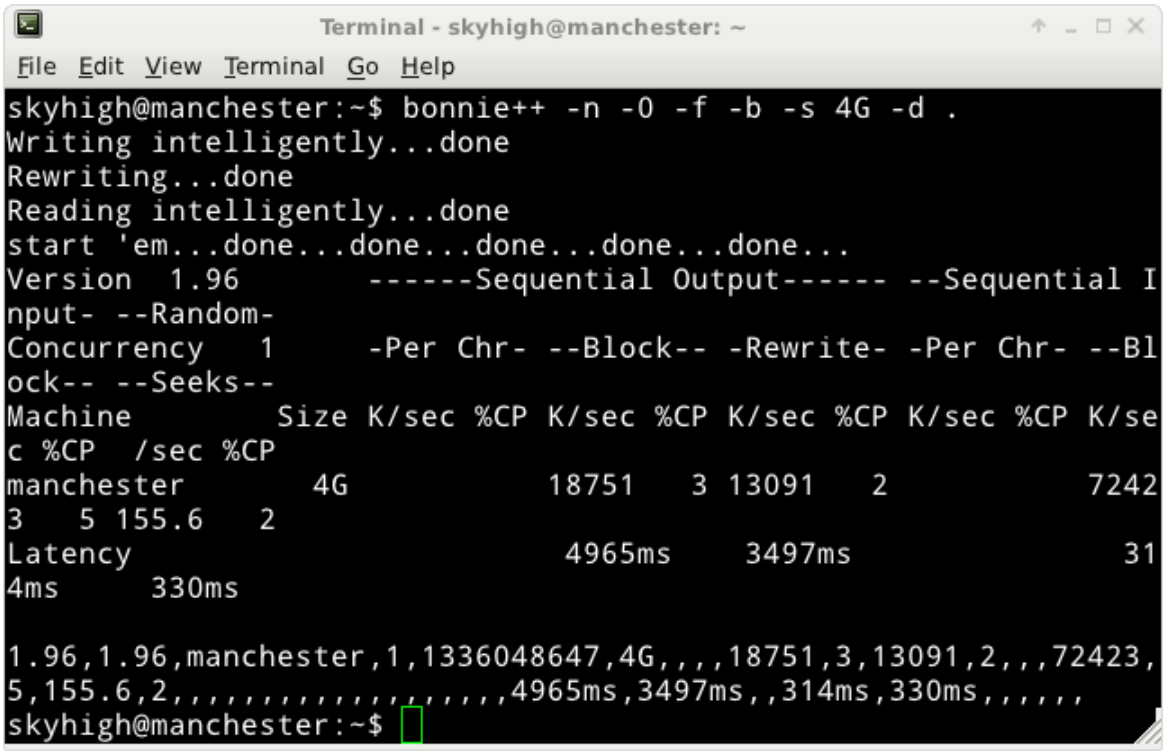
- Hastighet, CPU-bruk og responstid for sekvensiell skriving og lesing, henholdsvis per blokk eller per byte. Det testes også noe som kalles "rewrite", en operasjon som leser av en blokk på 8KB, søker tilbake til starten av blokka og deretter overskriver den. I følge utvikleren simulerer dette databaseoperasjoner
- Antall tilfeldige søk på disk per sekund sammen med responstid og CPU-bruk
- Antall metadataoperasjoner som kan utføres per sekund, samt responstid og CPU-bruk

Hastighetene som oppgis etter at testene er utført rapporteres i Kilobyte per sekund, og responstid i mikro eller millisekunder. Resultatet kan lagres som CSV (filer hvor data er kommaseparert). Disse dataene kan deretter gjøres om til egnet format eller parses og brukes til grafgenerering.

De viktige alternativene man har når man bruker Bonnie++ er disse:

- **-s** Velger størrelsen på fila som det skal gjøres tester mot. Denne må være minst to ganger så stor som den mengden minne systemet har tilgjengelig, slik at man tester lagringsenhetens ytelse og ikke hurtiglagringskapasitet. Bonnie++ tillater deg ikke å kjøre tester med filer under denne størrelsen
- **-x** Antall tester som skal kjøres. Denne funksjonen har en del problemer hvis det skal kjøres mange tester, slik at en løkke i et script vil være mer forutsigbar

- **-d** Området som skal testes. Dette kan være et sted på lokal disk eller en nettverksdisk



Figur 4.2: Skjermdump av Bonnie++

Flexible I/O Tester

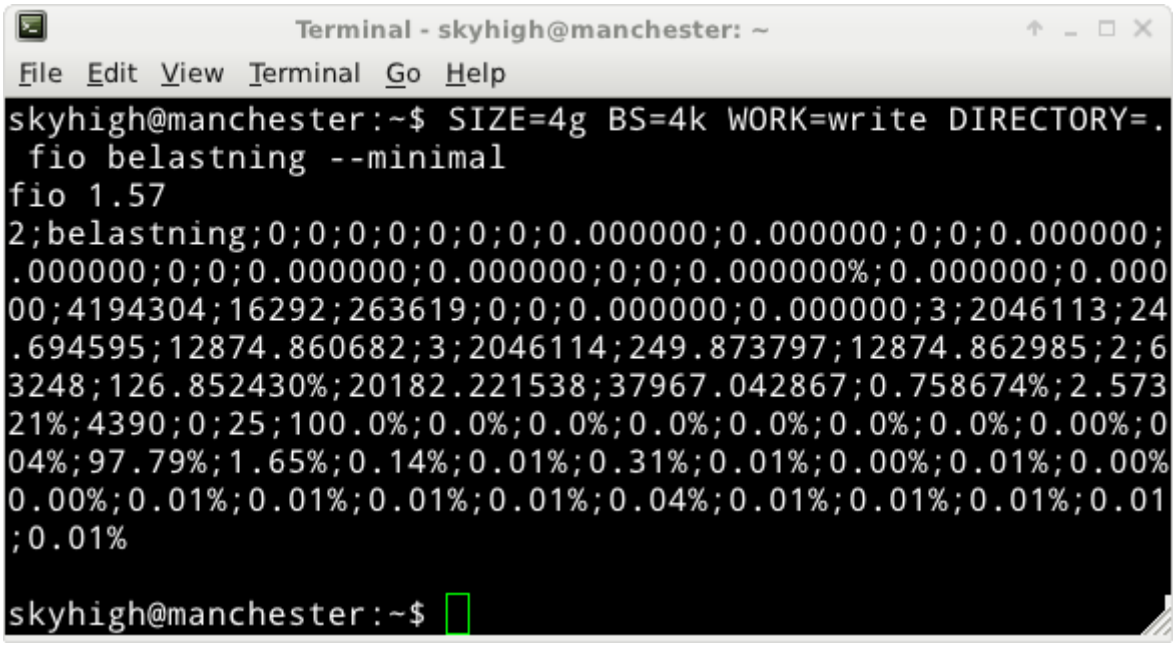
FIO [32] er en forkortelse for Flexible I/O Tester, et ytelsestestingsverktøy for lagringsmedier utarbeidet av Jens Axboe. Verktøyet ble utviklet for å lette hans arbeid med å utarbeide spesifikke arbeidsbelastninger og testprogrammer. Han ønsket muligheten til å definere komplekse arbeidsbelastninger uten å måtte skrive et eget testcase-program hver gang dette behovet dukket opp.

Verktøyet er i stand til å definere ulike I/O-arbeid, blant annet sekvensiell og tilfeldig skriving og lesing. Responstider er også inkludert, sammen med et vidt spenn av annen generert utdata, og man har i tillegg svært mange alternativer når det kommer til generering av arbeidsbelastning, blant annet:

- Blokkstørrelse

- I/O-dybde
- Filstørrelser og antall filer
- Parallelle jobber
- Prosentvis lesing og skriving (blanding)

FIO kjøres ved hjelp av kommandolinja og arbeidsbelastningsfilen benyttet under testene i oppgaven finnes i vedlegg D.



Figur 4.3: Skjermdump av FIO

Iometer

Iometer [33] er ment som et feilsøking- og ytelsesmålingsverktøy av lagringsmedia og blir hovedsakelig brukt til å måle "Input/Output Operations Per Second" (IOPS) og lese- og skrivehastigheter. Iometer ble opprinnelig utviklet av Intel, men er nå ivaretatt av The Linux Foundation [4] (tidligere The Open Source Development Lab).

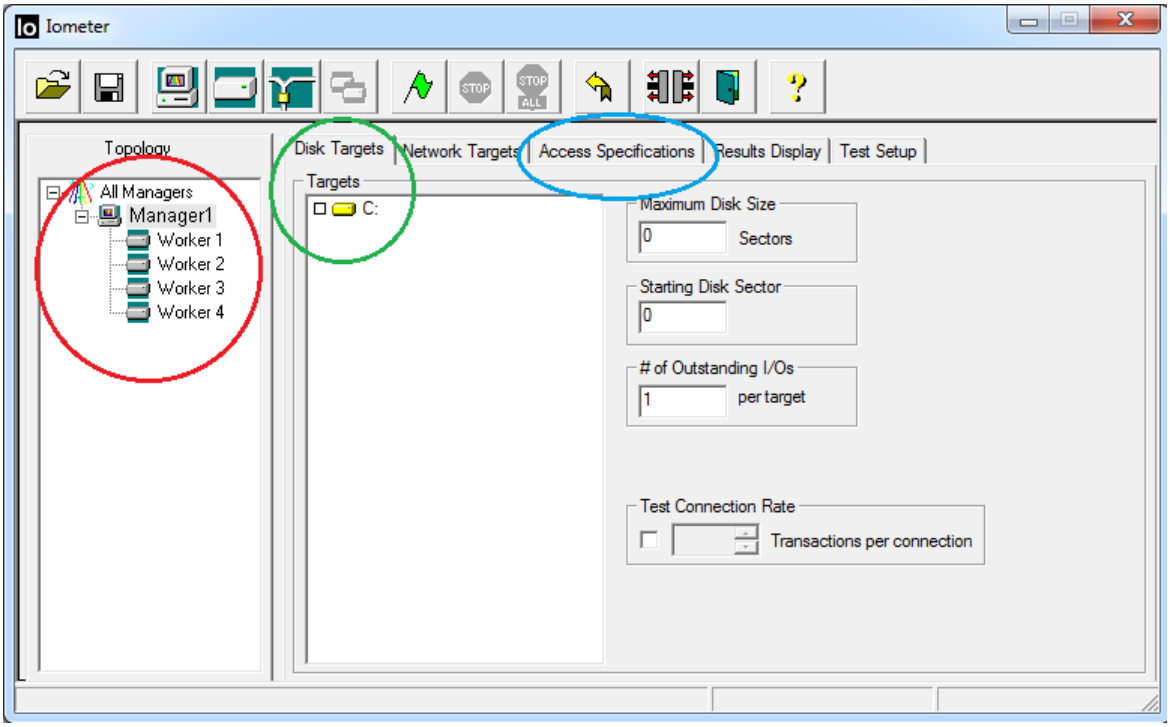
Verktøyet består av to moduler, der det ene (Iometer) gir et grafisk brukergrensesnitt som håndterer den andre modulens (Dynamo) prosesser, kalt

[4] http://www.linuxfoundation.org/

arbeidere. Man trenger derfor en server som kjører Iometer som sender ut arbeidsoppgaver til Dynamo-klientene på hver maskin som skal testes. Iometer-modulen må kjøre på en Windows-maskin, mens Dynamo kan kjøre på både Windows og Linux-plattformer og resultatene fra Iometer blir skrevet til en .CSV-fil.

Iometer gir brukeren veldig stor frihet hva angår konfigurering av testene. Parametere som prosentvis fordeling av lesing og skriving, størrelse på hver overføring og total størrelse på filen som skapes og hvor lang tid testen skal bruke, med mer, er alle mulige å endre på via brukergrensesnittet.

En skjermdump av Iometer kan observeres i figur 4.4. Rød sirkel viser arbeidere, grønn sirkel viser målområder (disker eller partisjoner) og blå sirkel viser meny for testoppsett.

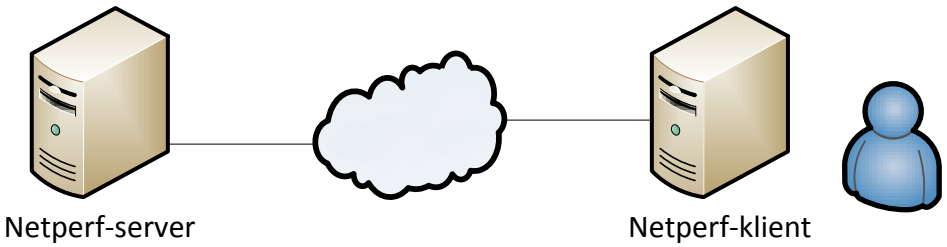


Figur 4.4: Skjermdump av Iometer

Netperf

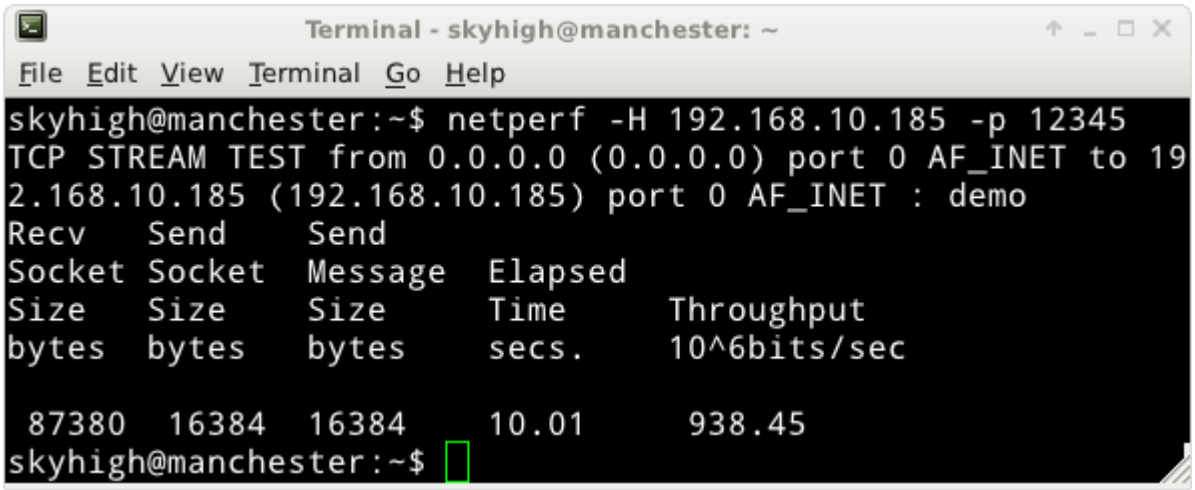
Netperf [34] er et verktøy som hovedsakelig brukes til å måle nettverksytelse med fokus på overføring av større datamengder, hvor resultatene blir oppgitt i antall Megabit per sekund (Mbit/s). Det er også mulig å utfø-

re "Request/Response"-tester der det sendes TCP-pakker kontinuerlig over nettverket mens testen pågår. Når den mottar et svar sendes neste pakke, og totalt antall slike pakker per sekund måles. En rekke andre muligheter finnes, blant annet testing av ytelse for UDP, men det er ikke spesielt relevant i forhold til ytelsen til hverken iSCSI eller OpenStacks moduler, da begge benytter TCP for kommunikasjon ettersom dette er mer pålitelig.



Figur 4.5: Illustrasjon av Netperf sin klient-server-arkitektur

Programmet er utviklet etter klient-server-modellen, der klienten snakker med en annen node som kjører Netperf-server. Mens testene pågår utveksles det imidlertid ingen kontrollinformasjon mellom disse for å unngå at testresultatene blir påvirket i negativ retning som følge av irrelevant pakkeflyt. Etter at målingene er avsluttet får klienten resultatene presentert.



Figur 4.6: Skjermdump av Netperf

Iperf

Et annet alternativ for måling av nettverksytelse er Iperf [35]. Iperf kan bruke både TCP- og UDP-pakker for å måle throughput (maks nettverksflyt) på nettverket, og på samme måte som Netperf bruker det en klient og en server for kommunikasjonen. Ved testing med bruk av UDP-pakker kan brukeren velge størrelsen på pakken og man vil på slutten få presentert den målte hastigheten samt pakketap. Ved bruk av TCP måles også maksimal trafikkflyt, ved at Iperf over en periode på 10 sekunder (med standardinnstillinger) kjører gjennom så mange pakker som mulig, for deretter å gi et resultat på hastigheten i Mbit/s og hvor mye den overførte i løpet av testen i antall MB. Standardinnstillingene er sammenlignbare med Netperf sine. Iperf blir ikke lenger utviklet aktivt og siste versjon kom i 2010.

Diskusjon rundt valg av verktøy

For å komme frem til hvilke verktøy som skulle benyttes til ytelsesmåling er alternativene oppført i nedenstående tabell 4.1, som beskriver tilkortkommenheter og fordeler. Når det kommer til valget mellom Netperf og Iperf, er det flere forhold som fører til at dette ikke var spesielt vanskelig. Iperf oppdateres ikke lenger, og det er usikkert om tråden noensinne vil tas opp igjen. Netperf blir fremdeles utviklet det kommer regelmessige oppdateringer. I studien "I/O Virtualization Bottlenecks in Cloud Computing Today" [36], som er en tilsvarende nettsky-studie hvor de undersøker Eucalyptus [5] blir Netperf benyttet, mens i en annen studie [37] som også går inn på Eucalyptus benyttes derimot Iperf til nettverkstestingen.

Avgjørelsen om testverktøy for lagringsmedier er ikke like triviell. Gruppen har likevel kommet frem til at både FIO og Iometer er gode valg for testing på henholdsvis Linux- og Windows-plattformer. I FIO er testene enkle å forholde seg til, man har et enormt utvalg parametre, verktøyet kan enkelt scriptes til å utføres flere ganger og resultatene produseres uten problemer i et forutsigbart format. Det er heller ikke avhengig av en klient-server-modell, slik som Iometer, noe som gjør det hele mindre komplekst. Verktøyet DD [6] er også et alternativ, brukt i [36], men har ikke like stor fleksibilitet da det i utgangspunktet ikke er myntet på ytelsestesting.

Til tross for at Bonnie++ benyttes i flere studier [36] [37] og er et mer anerkjent verktøy enn FIO, ble det valgt å gå bort i fra det, noe som var basert på Bonnie++ sin manglende mulighet for testing av tilfeldig lesing og skriving til disk, som ses på som vesentlig i forhold til oppgaven og oppdragsgivers

[5] http://www.eucalyptus.com/
[6] http://en.wikipedia.org/wiki/Dd_(Unix)

krav. Bonnie++ har heller ikke muligheten til å kjøre de forskjellige testene (sekvensiell skriving og lesing) individuelt slik som FIO, hvilket gjør at resultatene vil bli svært uforutsigbare og direkte feil når man tester med flere fysiske eller virtuelle maskiner samtidig fordi de etter hvert vil befinne seg på forskjellige tester til forskjellige tidspunkt. I tillegg nekter Bonnie++ å starte med mindre filene spesifiseres til å være dobbelt så store som eller større enn mengden tilgjengelig minne.

I utgangspunktet vil det for Windows kun eksistere ett valg for gruppen angående hvilket av de to diskytelsesverktøyene som vil bli brukt, da FIO ikke finnes for Windows. Iometer blir altså oppgavens verktøy for denne plattformen dersom det gis tidsmessig anledning til å utføre tester på Windows-maskiner. Iometer sin Dynamo-modul kan riktignok også kjøre på Linux, men dette krever en del mer når det kommer til installering og konfigurering, i tillegg til at det kompliserer automatisering av tester for Linux-maskiner, og derfor ble det avgjort å begrense bruk av Iometer til kun å omfatte eventuell Windows-testing.

Verktøy	Fordeler	Ulemper
Bonnie++	• Enkelt verktøy for grunnleggende tester	• Liten individuell kontroll over tester • Tester ikke tilfeldig lesing og skriving • Utvikles ikke lenger
FIO	• Lett å automatisere • Mange konfigurasjonsvalg • Mulig å styre individuelle tester • Hyppige oppdateringer • Gir informasjon om responstider	• Kan virke overveldende
Iometer	• Mange konfigurasjonsvalg • Mulig å styre individuelle tester	• Som FIO kan det virke overveldende • Ikke like trivielt å automatisere
Iperf	• Mange valgmuligheter	• Klient-server-modell • Utvikles ikke lenger
Netperf	• Mange valgmuligheter • Utvikles fremdeles	• Klient-server-modell
Ping	• Finnes på alle plattformer • Mange bruksområder • Enkelt	

Tabell 4.1: Fordeler og ulemper for undersøkte testverktøy

Gruppens valg er derfor å benytte seg av FIO og Iometer til testing av lagringsmedia, mens Netperf vil stå for testing av nettverksytelse på begge plattformer dersom det blir funnet tid og anledning til Windows-testing. Et eget script blir laget for å starte virtuelle maskiner samt måling av hvor kort eller lang tid dette tar. I tillegg vil Ping [7] brukes for å måle responstid

[7] http://en.wikipedia.org/wiki/Ping/

mellom virtuell maskin og henholdsvis hypervisor og lagringsnode.

Verktøy	Informasjon
FIO	MB/s, responstider
Iometer	MB/s, responstider
Netperf	Mbit/s, CPU-bruk
Ping	Responstid
Utrulling av et antall virtuelle maskiner	Tid

Tabell 4.2: De ulike verktøyene og hvilken informasjon man får ut av dem

Kapittel 5

Testmiljø

Maskinvare

Implementasjonen av SkyHiGh blir bestående av servere lånt av IT-tjenesten ved Høgskolen i Gjøvik. Dette er spesifikt HP ProLiant DL320-maskiner [38] med maskinvare lik det som er avtegnet i tabell 5.1

	Compute-noder og controller	Lagringsnode
CPU	Dual-Core Intel Xeon processor 3060 (2.40 GHz, 4MB L2 Cache, 65 Watts, 1066MHz FSB)	Dual-Core Intel Xeon processor 3060 (2.40 GHz, 4MB L2 Cache, 65 Watts, 1066MHz FSB)
Minne	4GB for controller og 2GB for compute-noder	4GB
Lagring	2 stykker Seagate Barracuda ES ST3250620NS eller Western Digital WD2500YS-70SHB1 250GB 7200RPM-disker	12 stykker Seagate Barracuda ES ST3250820AS 250GB 7200RPM
RAID-nivå	RAID 1	RAID 6 og 1
RAID-kontroller	HP SmartArray E200i	HP SmartArray P400
Nettverk	2 stykker à Intel Corporation 82571EB Gigabit Ethernet og 2 stykker à Broadcom Corporation NetXtreme BCM5714 Gigabit Ethernet	2 stykker à Broadcom Corporation NetXtreme BCM5714 Gigabit Ethernet
Linux-kjerne	3.2.0-19-generic	3.0.0-16-server
Distribusjon	Ubuntu 12.04 (Precise Pangolin)	Ubuntu 11.10 (Oneiric Ocelot)
Hypervisor	KVM	Ingen

Tabell 5.1: Maskinvare i compute-, controller- og lagringsnoder

RAID-oppsett

Lagringsnoden i infrastrukturen har tilsvarende maskinvare som compute-nodene og controller-noden, men avviker på visse områder, blant annet i forhold til lagringskapasitet og RAID-kontroller. I stedet for to 250GB-harddisker er det totalt tolv Seagate Barracuda ESXSDADS 250GB 7200RPM SATA-disker. 10 av diskene er satt opp i RAID 6 (hvilket gir 8 utnyttbare disker siden to av disse diskene brukes til å lagre paritetsinformasjon i redundansøyemed, totalt ca. 2TB), og to disker brukes i et RAID 1-oppsett til speiling av operativsystemet. Lagringsnoden har ei heller noen hypervisor kjørende, all den tid ingen virtuelle maskiner skal kjøre på denne maskinen – den står kun for iSCSI-basert lagring for compute-nodene.

Compute-nodene og controller-noden derimot, med sine to identiske disker hver, benytter bare RAID 1. Både controller og compute-noder bruker HP sin SmartArray E200i hardware-RAID-kontroller. Lagringsnoden benytter HP sin SmartArray P400 hardware-RAID-kontroller. Sistnevnte er en kraftigere modell enn E200i, og det forventes at RAID-ytelsen på compute-nodene og controller-noden ikke vil fremstå som veldig imponerende. Fordelen med hardware-RAID er at det i teorien skal gi bedre ytelse, siden det i motsetning til software-RAID ikke skaper belastning på prosessoren (RAID-prosessering håndteres av hardware-kontrolleren). I dette tilfellet kan det imidlertid være bedre ytelse å hente ved å bruke software-RAID, men oppsettet av SkyHiGh bestemmes av gruppen som faktisk implementerer OpenStack, slik at det er begrenset hva som kan testes i så henseende.

Nettverk

Nettverket i infrastrukturen er kablet og full-Gigabit, med unntak av en 10/100Mbit ruter. Som switch benyttes en Cisco SG 200-26 [39] 10/100/1000Mbit-modell. Dette er en såkalt "smart-switch" som har et webgrensesnitt for konfigureringsfunksjonalitet. Det er totalt 26 porter med full Gigabit-aksess og switchen har støtte for virtuelle nettverk. For å få til kommunikasjon mellom de virtuelle nettverkene trengs det en ruter, og det er blitt lånt en Catalyst-ruter i 2800-serien. Denne har ikke Gigabit-porter, noe som gjør at virtuelle nettverk ikke kommer til sin fulle rett i et nettverk ellers preget av Gigabit-utstyr, hvilket har begrenset muligheten til å implementere dette og fremdeles få fornuftige resultater.

Det er forespeilet å dele infrastrukturen inn i tre forskjellige virtuelle nettverk. Ett tilhører compute-nodene, ett eksisterer for lagringsserveren, og ett hører til controlleren og ruteren. Controlleren er med i alle tre av de virtelle

nettverkene; compute-nodene er representert i lagringsnettverket også, mens lagringsnoden kun holder til i sitt eget virtuelle nettverk.

Operativsystem

Instanser i OpenStack startes som kloner av forskjellige typer "images", som er ferdigbygde operativsystem av en viss type. Når instanser rulles ut blir de unike i form av tildelt maskinnavn og deltakelse i forskjellige nettverk og prosjekter.

Gruppens tester er utført på virtuelle maskiner basert på et image av Linux-distribusjonen Ubuntu 11.10 (kodenavn Oneiric Ocelot), som er den nyeste stabile utgaven. Dette er ikke samme operativsystem som OpenStack for SkyHiGh er implementert med, grunnet problemer med å få nåværende utgave av OpenStack operasjonelt på versjon 11.10. En oversikt over mulige instanstypevalg finnes i tabell 3.3.

Programvare

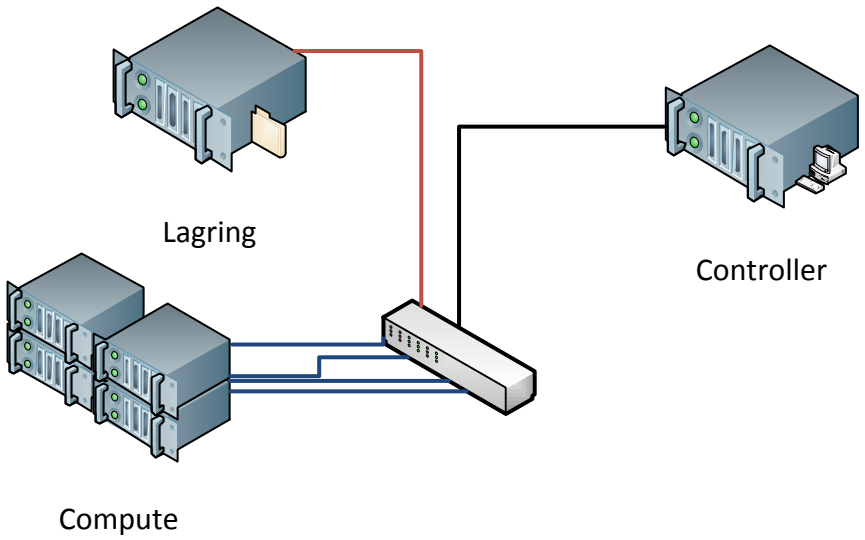
Se tabell 5.2 under for oversikt over versjonene av relevant programvare som blir benyttet.

Verktøy	Versjon
Flexible IO Tester	1.57
Netperf	2.4.5
OpenStack	Essex RC1
Open-iSCSI	2.0.871
iSCSI-Target	1.4.20

Tabell 5.2: Versjoner av programvare i infrastrukturen

Logisk infrastruktur

Figur 5.1 inneholder en illustrasjon av den logiske infrastrukturen. Fargene indikerer de ulike virtuelle nettverkene. I praksis er alle nodene deltakere i det samme nettverket i mangel på en ruter som kan understøtte et Gigabit-nettverk, som er nødvendig for full hastighet når det skal rutes mellom virtuelle nettverk, noe som selvsagt er det ønskede oppsettet hvis SkyHiGh blir satt i produksjon.



Figur 5.1: Logisk infrastruktur

Kapittel 6

Praktisk gjennomføring av ytelsestesting

Hovedhensikten med denne bacheloroppgaven er å se hvordan I/O-ytelse for virtuelle maskiner i OpenStack påvirkes av hvor filene de kjører fra fysisk befinner seg: på de lokale harddiskene til compute-nodene, eller på nettverksbasert ekstern lagring via iSCSI, og hvordan den når opp til hypervisorens ytelse. Nettverksytelse vil også bli testet.

Målepunkter

Ytelsen som rapporteres fra verktøyene gruppen har valgt rapporteres i MB/s (Megabyte per sekund hvor 1MB tilsvarer 2^{20} byte) og Mbit/s (Megabit per sekund hvor 1Mbit tilsvarer 10^6 bit). I tillegg vil responstid for operasjoner rapporteres i millisekunder. Verdiene vil deretter ses på i lys av nedenstående beskrivende statistikk [40].

Gjennomsnitt

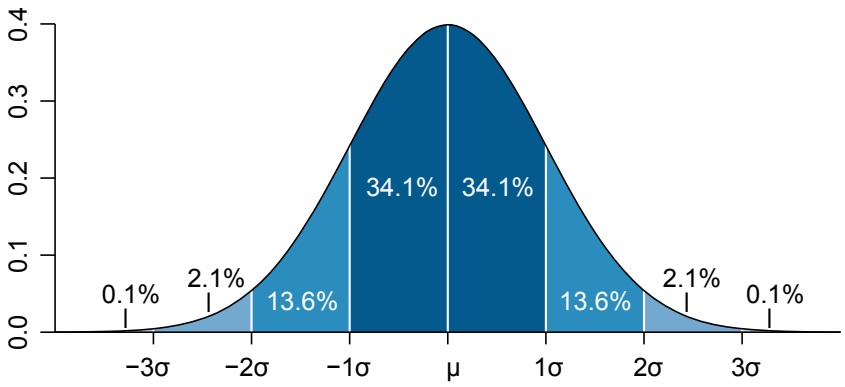
Gjennomsnitt blir funnet ved å summere alle verdiene til alle testgjennomføringer kjørt under en viss konfigurasjon og dele på antall gjennomføringer. Fremkommet verdi kan være farget av stor variabilitet i dataene og sier ikke nødvendigvis noe om et typisk resultat. Noen få ekstreme verdier kan gjøre at gjennomsnittet trekkes kunstig opp, noe som ødelegger for hvor representativt og beskrivende det i realiteten er.

Standardavvik

Standardavviket er definert som kvadratroten av variansen i et datasett. I boken "Statistikk for universiteter og høgskoler" [40] kommer det frem at det er det vanligste av alle spredningsmål, "et typisk avvik fra gjennomsnittsverdien". Det forteller noe om spredning, usikkerhet og variabilitet i dataene - hva det gjennomsnittlige avviket er fra middelverdien (normalen). Dermed får man vite hvilket område ytelsen sannsynligvis kommer til å befinne seg i. I oppgavens grafer blir ett standardavvik vist ved hjelp av såkalte "errorbars" i stolpediagrammene. Standardavviket er også opplistet i resultattabellene i resultatkapittelet i parenteser.

Normalfordeling

Oppdragsgiver ønsket at et lite utvalg av resultatene også skulle bli undersøkt på en måte som viser om de er normalfordelt, altså at de er noenlunde symmetriske og har kun én distinkt topp. Normalfordelingen er regnet for å være den viktigste fordelingen innen statistikk og det nevnes at "når antallet tester blir høyt, blir fordelingen tilnærmet klokkeformet" [40]. Antallet gjennomføringer for majoriteten av denne oppgavens tester er 100, et antall som regnes som tilstrekkelig i følge boken. Se figur 6.1 for en illustrasjon av et normalfordelt datasett [1].



Figur 6.1: Illustrasjon av normalfordeling.

[1] http://no.wikipedia.org/wiki/Fil:Standard_deviation_diagram.svg lisensiert under Creative Commons

Gjennomgang av tester

Gruppen skal i denne bacheloroppgaven gjennomføre tester som måler disk- og nettverksytelse for fysiske og virtuelle maskiner. For nettverksytelsestestene skjer dette ved at en eller flere klientmaskiner av ulik type kobler seg til en maskin som har én eller flere Netperf-servere kjørende, og deretter enten sender eller mottar TCP-pakker så raskt den eller de klarer. Det måles også gjennomsnittlig responstid i med ping.

Når det kommer til diskytelsestestingen, skjer dette ved at én eller flere fysiske eller virtuelle maskiner gjør følgende operasjoner parallelt, målt med programmet Flexible IO Tester (FIO):

- Sekvensiell skriving av en fil
- Sekvensiell lesing av en fil
- Tilfeldig skriving av en fil
- Tilfeldig lesing av en fil

Årsaken til at disse testene kjøres er fordi de er de fire mest vanlige operasjonene som utføres på et filsystem. En kan også kjøre blandede oppgaver, men gruppen føler det blir utenfor omfanget av denne bacheloroppgaven. De tilfeldige skrive- og leseoppgavene kjører i 100 sekunder, mens sekvensielle tester kjører inntil de er ferdige. Årsaken er et ønske om å få et tilstandsbilde – skulle de tilfeldige oppgavene kjøre til de ble ferdige ville omfanget av testingen blitt forlenget betydelig, altså er det et kompromiss.

Filen er i alle tilfeller utenom ett på 4GB for å unngå hurtiglagringseffekter. Siden dette er syntetiske tester gjennomført med programmet "Flexible IO Tester" er det ønskelig å se hva den reelle diskytelsen er uten at hurtiglagring influerer i betydelig grad. På denne måten sammenlignes også fysisk og virtuell ytelse i større grad. Gruppen har også en test hvor det forsøkes å påvise hurtiglagringseffekter; filstørrelsen er i disse unntakstilfellene redusert til henholdsvis 128MB, 256MB og 512MB. Små nok til å være relevante for eventuelle sluttbrukere, men store nok til å se forskjeller mellom oppsettene.

En overordnet liste over de ulike testene følger under:

- Testing av lokal diskytelse for fysiske maskiner i infrastrukturen
- Testing av nettverksytelse for fysiske og virtuelle maskiner i infrastrukturen, med og uten link aggregation på lagringsnoden

- Testing av iSCSI-ytelse for fysiske maskiner i infrastrukturen, med og uten jumbo frames og flere iSCSI-tråder
- Testing av diskytelse for virtuelle Linux-maskiner i infrastrukturen, med og uten jumbo frames og flere iSCSI-tråder
- Testing av utrullingshastighet for virtuelle maskiner
- Testing av mulige hurtiglagringseffekter for virtuelle maskiner avhengig av iSCSI-oppsett på compute-noden

Detaljert informasjon om alle overnevnte tester, herunder script benyttet, maskiner involvert og forklarende figurer finnes i vedlegg A.

Begrunnelse for valg av parametre for testing

Alle diskytelsestester vil kjøres på filsystemet "Ext4". Dette er oppfølgeren til et populært filsystem for Linux [41], og i tillegg standard for Ubuntu og andre Debian-baserte filsystemer. Gruppen ser selvsagt begrensningene i å kun teste med ett filsystem, da filsystemet har innvirkning på syntetiske mikrobenchmarks såvel som makrobenchmarks, men har vurdert det som nødvendig som følge av pragmatikk og tidsbegrensninger. Filformatet de virtuelle maskinene er lagret med er QCOW2, og ved bytting fra fileio til blockio på lagringsnoden ble LVM-volum og filsystem opprettet på nytt for å sikre konsistens.

Diskytelstester vil utføres med blokkstørrelse lik 4KB. Blokkstørrelse er den minste bestanddelen en fil deles inn i. En fil som i utgangspunktet bare rommer 2KB data vil derfor likevel okkupere en blokk på 4KB. Årsaken til dette valget er oppdragsgivers og veileders ønske om 100 gjennomføringer av hver test, noe som skaper tidsmessige hensyn og begrensninger. Et annet forhold er at 4KB er standardblokkstørrelsen for standardfilsystemet Ext4.

Linux-distribusjonen det vil kjøres tester på er "Ubuntu Oneiric Ocelot 11.10". Dette er den nyeste stabile versjonen av Ubuntu, og velges grunnet kompatibiliteten med OpenStack. Det er i tillegg anbefalt distribusjon fra utviklerenes side, og innebærer mindre arbeid å få operasjonelt enn andre distribusjoner. I utgangspunktet er det imidlertid ingen spesielle begrensninger som setter føringer for hvilke distribusjoner som er mulige å implementere.

Størrelsen på filene som testes er, med unntak av én test, satt til å være 4GB. Årsaken til dette er at man ved å velge en filstørrelse som er dobbelt så stor – eller større – enn mengden minne (dvs større enn det man får plass til i buffer cache) på maskinen det testes fører til en negasjon av hurtiglagringseffekt [42]. På denne måten ser man faktisk oppnåelig I/O uten betydelig

påvirkning av hurtiglagring, noe som gir et bedre bilde av forskjellene mellom konfigurasjonsalternativer, og mellom fysisk og virtuell maskin.

Visualisering og representasjon av testdata

Gruppen kommer hovedsaklig til å benytte stolpediagram med tilhørende representasjon av standardavvik etter ønske fra oppdragsgiver. Noe tid gikk med på undersøkelse av hvordan dette lettest mulig kunne bli implementert. Gnuplot [2] stilte seg først i rekken av aktuell programvare, og det finnes et ferdigutviklet perl-script utviklet av Derek Bruening [43] som gjør akkurat det som var ønsket. Bruk av dette scriptet frigjorde tid til andre presserende oppgaver, og gjorde genereringen av grafer på ønsket format enkelt. Til enkelte grafer brukes imidlertid Gnuplot uten Dereks script, da dét kun er tilrettelagt for stolpediagram.

Prinsipper

Det finnes en rekke prinsipper man burde følge ved ytelsestesting. Følgende regnes på som vesentlige [44]:

- **Mengdetesting:** Det bør samles inn store nok mengder data til at man kan med en viss grad av sikkerhet fastslå at resultatene kan sies å ha en form for validitet, altså at det man måler ikke bare er tilfeldigheter. Gruppen erkjenner imidlertid at man aldri kan vite om empiriske data [3] representerer virkeligheten i alle tilfeller, slik at en feilmargin er å regne med.
- **Repeterbarhet:** Det er også viktig at målingene man utfører er repeterbare - av oss så vel som andre som skulle ønske å forsøke å gjenta de samme eksperimentene. Dette medfører at målingene gjøres på en fast og kanskje scriptet måte. I tillegg bør også eventuell kildekode og fremgangsmåte legges ut offentlig. Dette vil gi personer som eventuelt kommer over resultatene i ettertid en mulighet til å utføre de samme testene og dermed kunne ha et sammenligningsgrunnlag
- **Automatisering:** Automatisering henger sammen med punktene over. For å unngå repetitiv, interaktiv testing bør man automatisere så mye som mulig, såfremt det ikke introduserer unødigvendig kompleksitet. Dette sparer ikke bare tid, men er også med på å skape forutsigbarhet og konsistens

[2] http://www.gnuplot.info/
[3] http://no.wikipedia.org/wiki/Vitenskap/

Artikkelen "Linux Performance Tuning" [44] peker også på noen andre relevante prinsipper som burde følges ved ytelsestesting:

- Undersøk grunnytelsen til systemet før eventuelle forsøk på optimaliseringer
- Forsøk på å øke ytelse er meningsløst uten ytelsestester - test derfor både før og etter eventuelle optimaliseringer

Forskjellige variabler som kan påvirke resultatene bør også kontrolleres. Dette betyr at compute-nodene for eksempel ikke skal utføre andre ikke-kritiske oppgaver mens tester utføres, slik at resultatene ikke blir influert i positiv eller negativ retning og kompromitterer integriteten til disse.

Usikkerhetsmomenter og fallgruver ved testing

Selv om ytelsesmålingsprinsippene nevnt er fornuftige og viktige i forskningsøyemed, er det ikke alltid de blir etterfulgt. I studien "Benchmarking File System Benchmarking: It *IS* Rocket Science" [45] kommer det frem at bruk av ad-hoc ytelsesmålingsverktøy var til stede i 56,7% av 527 ytelsesmålingrapporter i perioden 1999 til 2010, mens det nest mest brukte på listen, kompilering (av Apache, Linux-kjernen etc), ble brukt i 9,8% av tilfellene. Det er ikke galt å bruke ad-hoc-verktøy – altså noe som ikke er anerkjent eller vanlig – i seg selv, men flere av rapportene artikkelen tar for seg viser ikke tegn til å ha noen spesiell grunn til å ikke ha brukt av de allerede eksisterende produktene der ute som er laget for nettopp de målingene de ønsket å utføre. I tillegg viste få av rapportene standardavvik for resultatene og hvordan de hadde utført målingene, slik at det blir det veldig vanskelig å repetere disse målingene i fremtiden for å etterprøve eller sammenligne.

Det er en rekke forhold som kan påvirke ytelsen som observeres under måling for denne oppgaven [45]:

- Versjoner av programvare (Linux-distribusjon, Linux-kjerne, OpenStack, hypervisor, ytelsesmålingsverktøy, iSCSI med mer)
- Valg av filsystem (og om man bruker filsystem i det hele tatt), blokkstørrelse, konfigurasjon ved oppmontering, I/O-kødybde, scheduler-innstillinger med mer
- Hypervisor (KVM, Xen) og konfigurering av denne
- Parametre for TCP og nettverk (window size), Ethernet jumbo frames med mer

- Andre tjenester som kjører samtidig på maskinene og skaper belastning på systemet eller nettverket

Gruppen anerkjenner derfor at alle resultater som fremkommer må tas med stort forbehold, og kan kun sies å gjelde for den maskinvaren som er tilgjengelig og presentert, med de og for akkurat de testene som kjøres. Det skal imidlertid likevel være mulig å hente frem og utlede noen generelle pekepinn på hva som påvirker ytelse, og hvordan man kan forholde seg til de måleresultatene som faktisk observeres for å gi konkrete anbefalinger.

Automatisering og testflyt

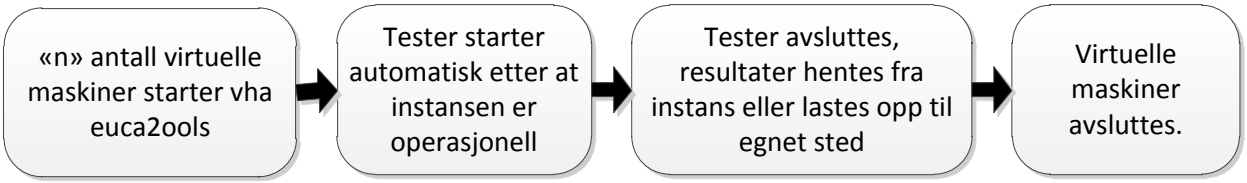
Virtuelle maskiner i OpenStack kan startes ved hjelp av et intuitivt webgrensesnitt, eller ved å benytte kommandolinjeverktøyene "novaclient" – som er OpenStack sitt egenutviklede verktøy – eller "euca2ools" [46] som har en del flere valgalternativer. For å automatisere testene så langt det praktisk lar seg gjøre, slik at de blant annet kan kjøre om nettene, var det et å ønske benytte "euca2ools". På denne måten kan man kjøre opp et gitt antall virtuelle maskiner med ønsket instanstype, som deretter starter testingen egenhendig, sletter disse virtuelle maskinene og deretter gjentar syklusen ved å starte et annet gjenstående sett tester.

En scheduler avgjør hvilken compute-node en virtuell maskin starter på, og standardmetoden er at kapasiteten på compute-noder fylles opp én etter én, hvilket betyr at tilgjengelig mengde maskinvare på en compute-node avgjør hvor man som kan kjøre på hver. Ingen andre metoder er undersøkt.

Gruppen skrev en rekke script til ytelsestestingsformål som finnes i vedlegg C. Disse starter de virtuelle maskinene og starter så kjøring av ytelsestester. For at testene skal starte automatisk, måtte imidlertid fastsatte testverktøy installeres og egenutviklede script skrevet i bash og perl som skulle benytte seg av disse kopieres over på et ferdiglaget Ubuntu-image klart for bruk i OpenStack. OpenStack har implementert en "snapshot"-funksjon som lager en fullverdig klone av kjørende instanser – denne funksjonaliteten er selvsagt prisgitt hypervisoren. Ved å installere alt som trengs for å kjøre testene på en kjørende instans, kan man deretter klone denne instansen og få mulighet til å bruke klonen som utgangspunkt for nye virtuelle maskiner. Dette sparer veldig mye tid på sikt: det eneste som tar tid er førstegangskopieringen av klonen fra controlleren til de andre compute-nodene, men det er et engangstilfelle såfremt en tar med alle verktøy en trenger ved første kloning.

Ikke alle av testene var like praktisk gjennomførbare å få implementert med automatisering uten betydelig ekstraarbeid, hvor timene kunne gått med til

mer matnyttige gjøremål i stedet. Tester som ikke involverte virtuelle maskiner var mer trivielle å utføre ved manuell starting av tester og innsamling av data.



Figur 6.2: Arbeidsflyt involvert i automatisering

Ytelsesresultater fra andre studier

Gruppen har noen antakelser for hvordan ytelsen vil stille seg etter å ha lest flere tilsvarende nettsky-studier. Shafer [37] undersøkte I/O-ytelse (sekvensiell skriving og lesing samt nettverksytelse, ikke tilfeldig aksess) av virtuelle Linux-maskiner med KVM og Xen som hypervisor i 2010, hvor det kommer frem at virtio er den foretrukne nettverksdriveren. Virtio fremstår også som den beste diskdriveren når KVM benyttes. Det overraskende funnet i studien er at det å lagre de virtuelle maskinene som filer gir best ytelse og ikke med EBS som rotfilsystem.

Baun et al [38] undersøkte også ytelsen i Eucalyptus, dog med Xen, men viste forskjeller mellom ulike instanstyper, som nok også vil gjelde for KVM. SkyHiGh I/O vil imidlertid kun teste med én instanstype, hvilket er begrensende, men nødvendig ut i fra tidsmessige hensyn.

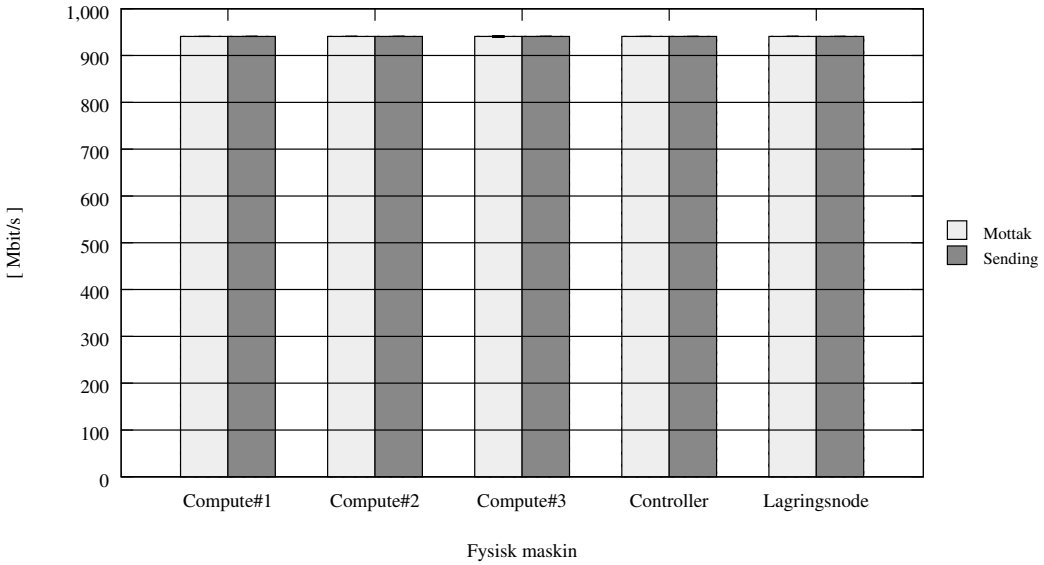
Kapittel 7

Resultater

Resultater som angår throughput for disk og nettverk vil i dette kapittelet oppgis i henholdsvis MB/s og Mbit/s og med to desimaler. Responstider oppgis i millisekunder både for diskytelse og nettverksytelse. Standardavvik for throughput i tabeller fremtrer i parenteser og gjelder for resultatet direkte over seg. For responstid og standardavvik ønsker man så lave resultat som mulig, mens for diskytelse og nettverksytelse er høyere resultat bedre.

Ytelse for nettverk

Fysiske maskiner



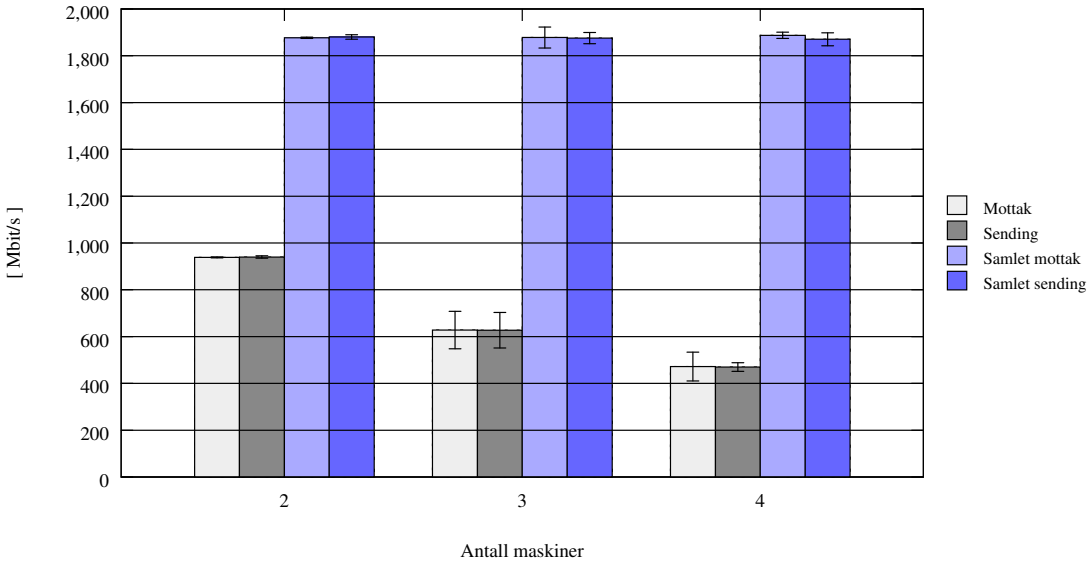
Figur 7.1: Nettverksytelse for fysiske maskiner

	C #1	C #2	C #3	Controller	Lagringsnode
Mottak	941.00 (0.00)	940.99 (0.10)	940.81 (1.89)	941.00 (0.00)	940.99 (0.11)
Sending	940.98 (0.14)	940.99 (0.10)	940.98 (0.14)	941.00 (0.00)	941.00 (0.00)

Tabell 7.1: Nettverksytelse for fysiske maskiner

Figur 7.1 og tabell 7.1 viser ytelse for nettverk (sending og mottak) for de fysiske maskinene i infrastrukturen.

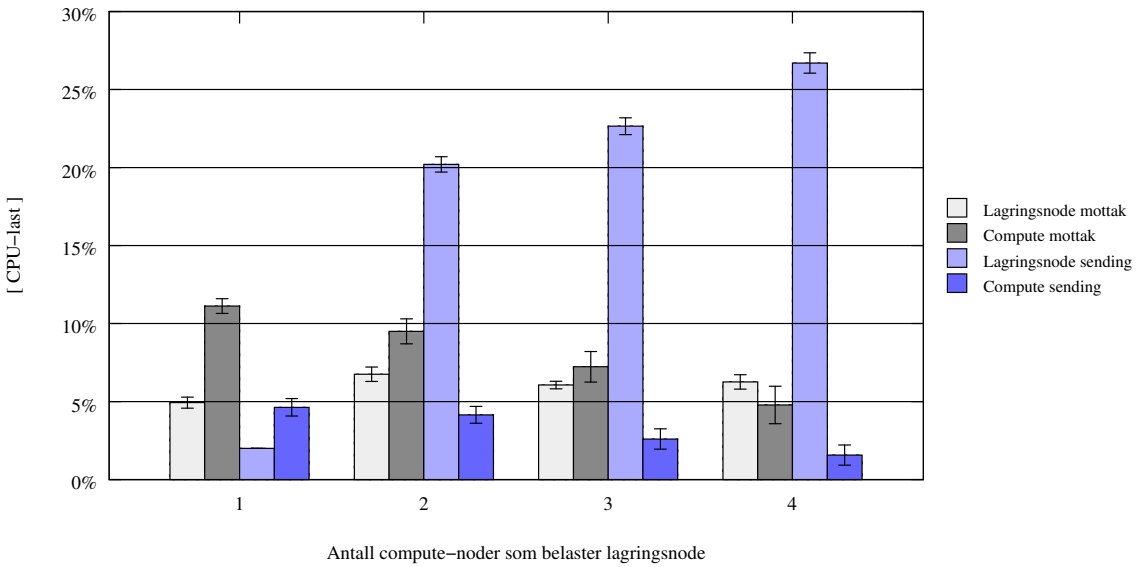
Link aggregation



Figur 7.2: Nettverksytelse med link aggregation på lagringsnoden

# maskiner	Mottak	Sending	Mottak totalt	Sending totalt
2	938.48 (1.91)	940.03 (4.95)	1876.96 (2.52)	1880.06 (9.65)
3	628.07 (80.05)	627.24 (75.98)	1877.95 (44.92)	1875.46 (23.95)
4	471.87 (61.68)	469.98 (18.47)	1887.47 (13.58)	1870.51 (27.90)

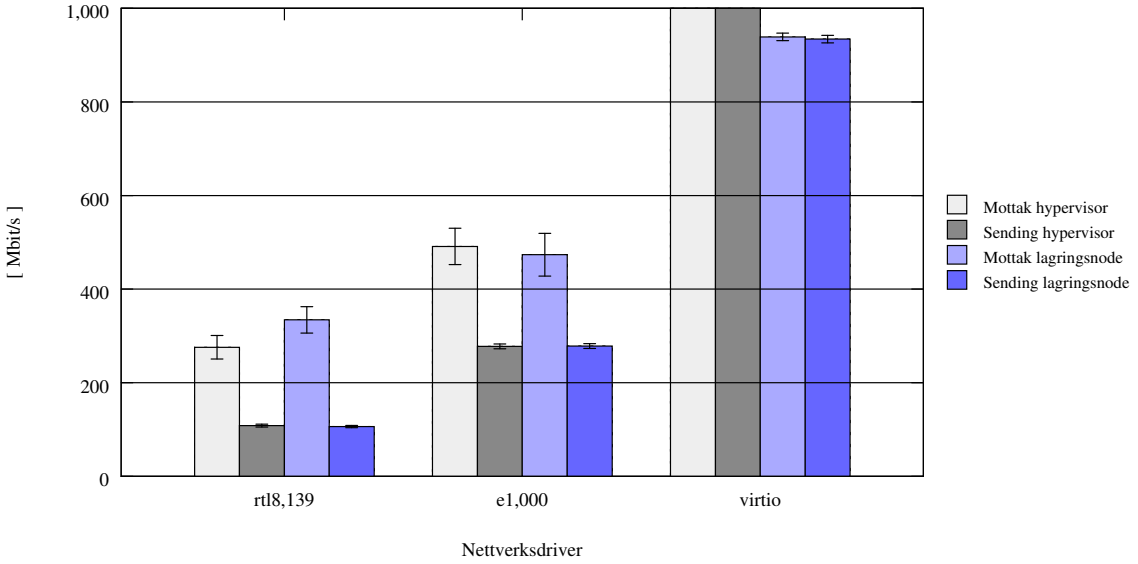
Tabell 7.2: Nettverksytelse med link aggregation på lagringsnoden



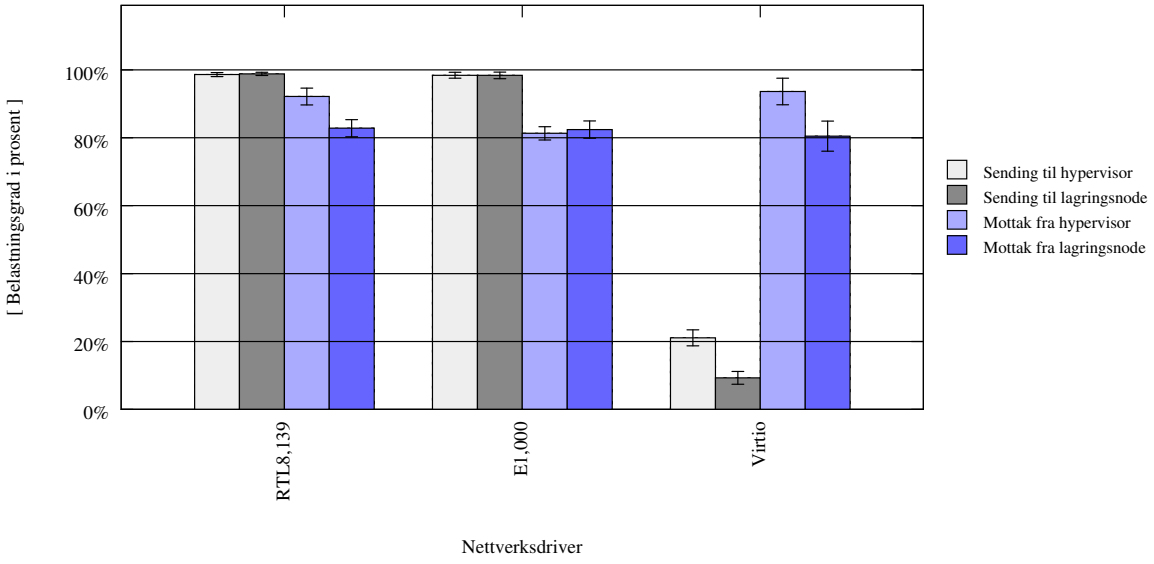
Figur 7.3: CPU-last ved testing av link aggregation

Figur 7.2 og tabell 7.2 viser ytelse for nettverk (sending og mottak) når de fysiske maskinene i infrastrukturen skal kommunisere med lagringsnoden og link aggregation er konfigurert. Figur 7.3 viser forskjellene i prosessorbruk avhengig av hvor mange fysiske maskiner som kommuniserer samtidig med lagringsnoden oppgitt i prosent av totalkapasitet.

Virtuelle maskiner



Figur 7.4: Nettverksytelse for virtuelle maskiner



Figur 7.5: CPU-bruk for virtuelle maskiner ved nettverkstesting

Nettverksdriver	Mål	Mottak	Sending	Responstid
Rtl8139	Hypervisor	275.78 (25.09)	108.39 (2.97)	0.38 (0.03)
E1000	Hypervisor	491.22 (38.76)	277.65 (5.09)	0.20 (0.04)
Virtio	Hypervisor	5521.79 (973.32)	4003.79 (458.73)	0.17 (0.03)
Rtl8139	Lagringsnode	334.32 (28.16)	106.38 (2.43)	0.33 (0.13)
E1000	Lagringsnode	473.54 (45.7)	278.37 (5.24)	0.29 (0.03)
Virtio	Lagringsnode	938.92 (8.08)	934.21 (8.15)	0.26 (0.03)

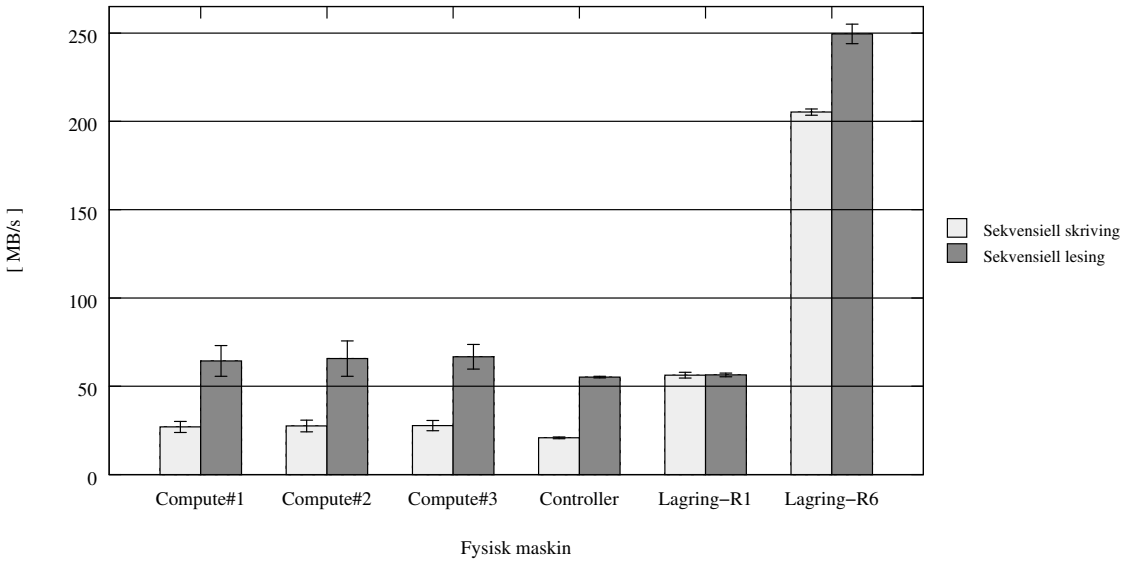
Tabell 7.3: Nettverksytelse for virtuelle maskiner

Figur 7.4 og tabell 7.3 viser ytelse for nettverk (sending og mottak) for virtuelle Linux-maskiner. Forskjellige nettverksdrivere og kommunikasjon både med compute-noden den virtuelle maskinen kjører på og lagringsnoden er representert i resultatene. Se figur 7.5 for oversikt over hvordan de ulike nettverksdriverene påvirker bruk av CPU hos den virtuelle maskinen.

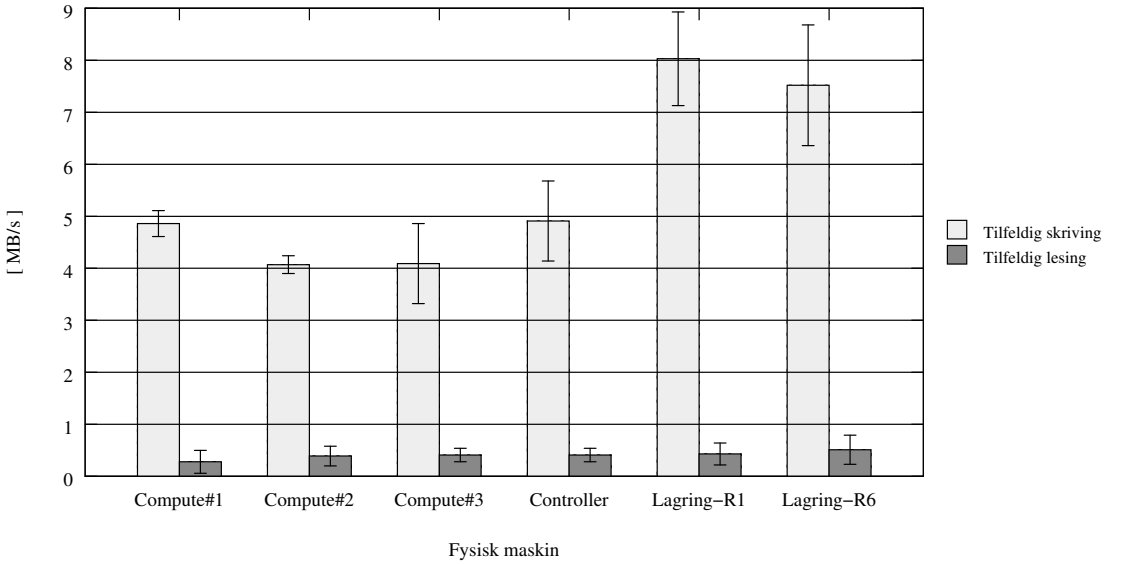
Ytelse for disk

Fysiske maskiner

Den fysiske maskinvarens harddisker



Figur 7.6: Lokal diskytelse for fysiske maskiner: Skriving og lesing



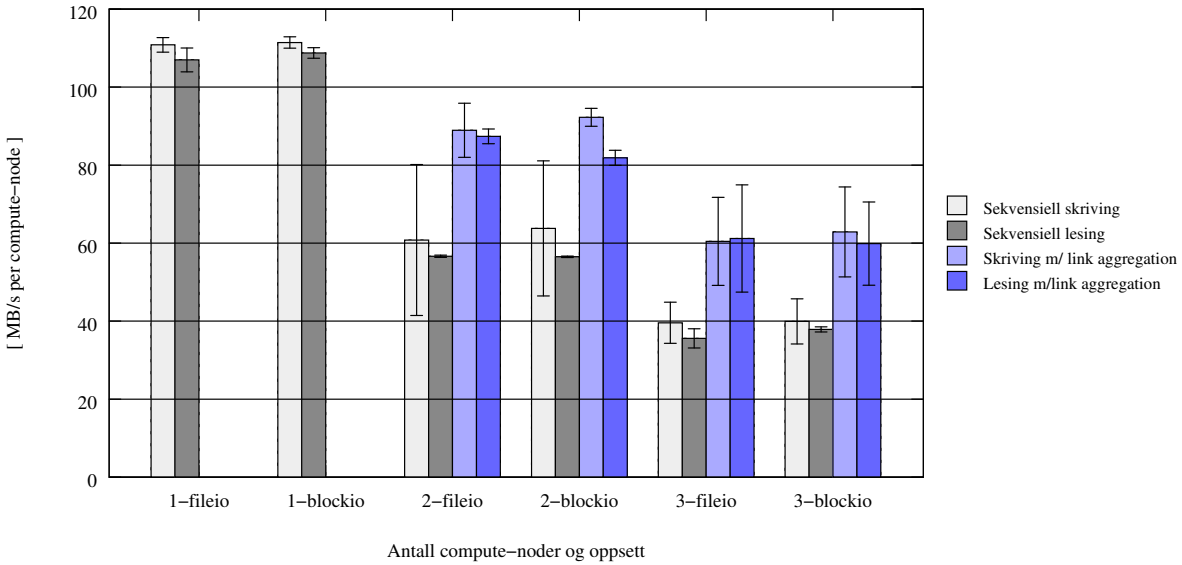
Figur 7.7: Lokal diskytelse for fysiske maskiner: Tilfeldig skriving og lesing

Maskin	Oppsett	Skriv	Les	Tilfeldig skriv	Tilfeldig les
Compute #1	RAID1	26.97 (3.14) 0.21ms	64.37 (8.71) 0.09ms	4.86 (0.25) 0.51ms	0.28 (0.22) 109.58ms
Compute #2	RAID1	27.54 (3.31) 0.29ms	65.69 (10.03) 0.07ms	4.07 (0.17) 0.98ms	0.39 (0.19) 88.40ms
Compute #3	RAID1	27.74 (2.87) 0.25ms	66.69 (7.00) 0.07ms	4.09 (0.77) 0.98ms	0.41 (0.13) 84.55ms
Controller	RAID1	20.82 (1.61) 0.18ms	55.19 (1.05) 0.07ms	4.91 (0.77) 0.58ms	0.41 (0.13) 12.92ms
Lagring	RAID1	56.29 (1.61) 0.07ms	56.43 (1.05) 0.07ms	8.03 (0.90) 0.51ms	0.43 (0.21) 59.69ms
Lagring	RAID6	205.25 (1.78) 0.02ms	249.50 (5.53) 0.02ms	7.52 (1.16) 0.55ms	0.51 (0.28) 67.09ms

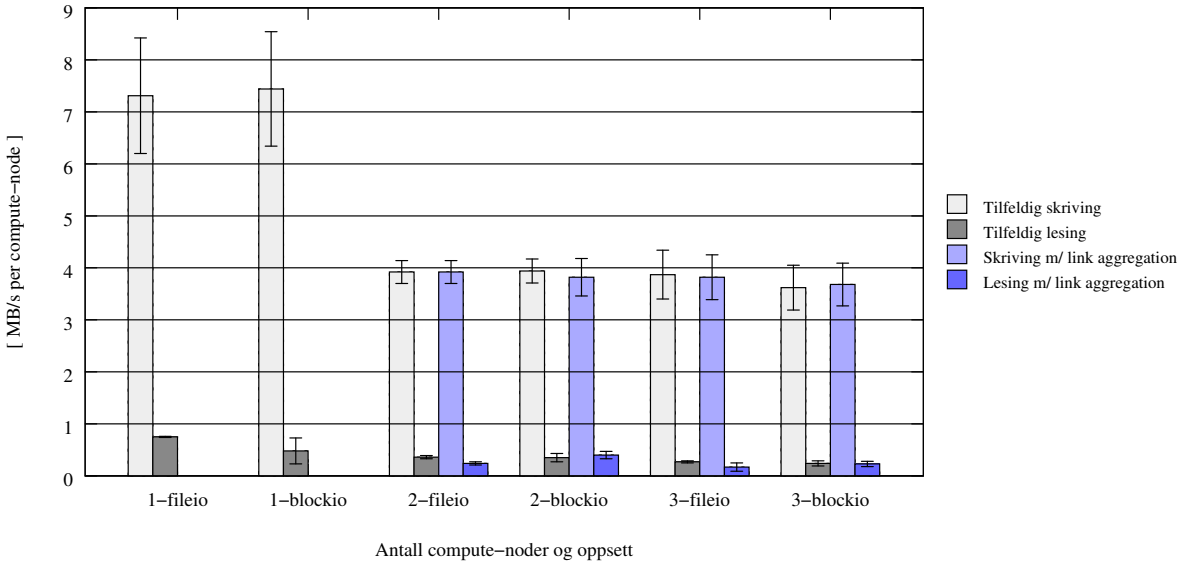
Tabell 7.4: Lokal diskytelse for fysiske maskiner

Figur 7.6 og 7.7 samt tabell 7.4 viser lokal diskytelse for den fysiske maskinvaren i infrastrukturen. De lokale diskene er harddiskene som fysisk befinner seg i de fysiske maskinene.

iSCSI



Figur 7.8: iSCSI-ytelse: Sekvensiell skriving og lesing



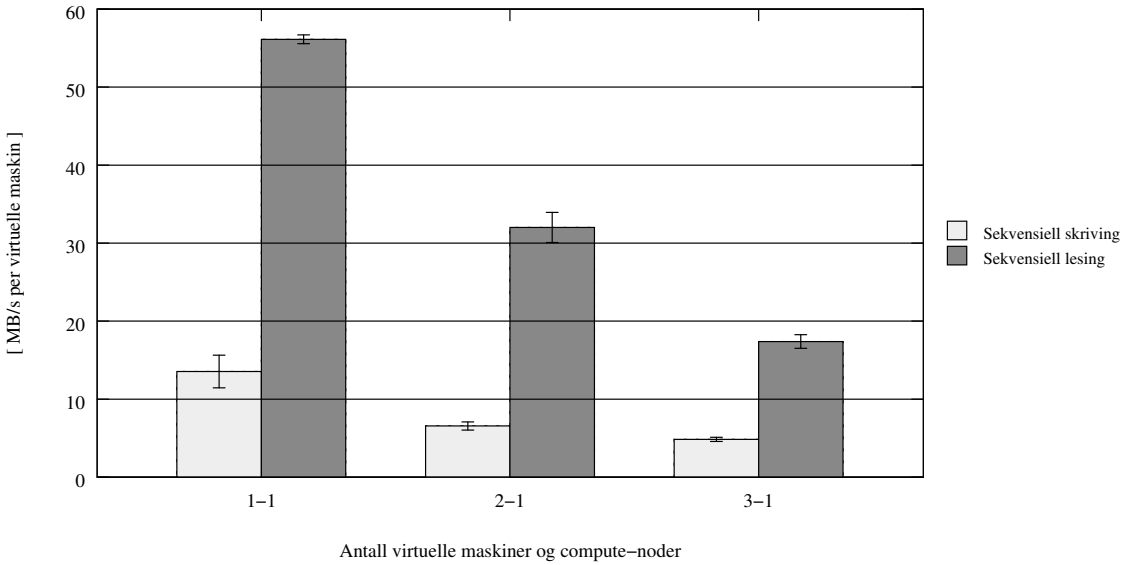
Figur 7.9: iSCSI-ytelse: Tilfeldig skriving og lesing

# Compute	Bonding	Oppsett	Skriv	Les	Tilfeldig skriv	Tilfeldig les
1	Nei	fileio	110.83 (1.87) 0.03ms	106.97 (3.07) 0.04ms	7.31 (1.11) 0.56ms	0.75 (0.01) 3.40ms
2	Nei	fileio	60.8 (9.68) 0.08ms	56.65 (0.14) 0.07ms	3.92 (0.22) 1.03ms	0.36 (0.03) 11.05ms
3	Nei	fileio	39.58 (1.76) 0.10ms	35.58 (0.82) 0.11ms	3.87 (0.47) 1.16ms	0.27 (0.02) 15.82ms
1	Nei	blockio	111.44 (1.46) 0.03ms	108.75 (1.35) 0.03ms	7.44 (1.1) 0.55ms	0.48 (0.25) 31.02ms
2	Nei	blockio	63.79 (8.66) 0.06ms	56.5 (0.09) 0.07ms	3.94 (0.23) 1.03ms	0.35 (0.08) 36.78ms
3	Nei	blockio	39.83 (1.93) 0.10ms	37.89 (0.22) 0.10ms	3.62 (0.43) 1.23ms	0.24 (0.05) 59.9ms
2	Ja	fileio	88.93 (3.47) 0.04ms	87.37 (0.95) 0.04ms	3.92 (0.22) 1.04ms	0.24 (0.03) 11.07ms
3	Ja	fileio	60.46 (3.76) 0.07ms	61.18 (4.58) 0.07ms	3.82 (0.43) 1.15ms	0.17 (0.08) 31.63ms
2	Ja	blockio	92.27 (1.5) 0.04ms	81.87 (0.97) 0.05ms	3.82 (0.36) 1.02ms	0.40 (0.07) 20.41ms
3	Ja	blockio	62.87 (3.85) 0.06ms	59.87 (3.56) 0.07ms	3.68 (0.41) 1.19ms	0.23 (0.05) 67.15ms

Tabell 7.5: iSCSI-ytelse for fysiske maskiner

Figur 7.8 og 7.9 samt tabell 7.5 viser oppnådd iSCSI-ytelse for fysiske maskiner. Det er testet med og uten link aggregation, og med fileio og blockio med inntil tre maskiner samtidig.

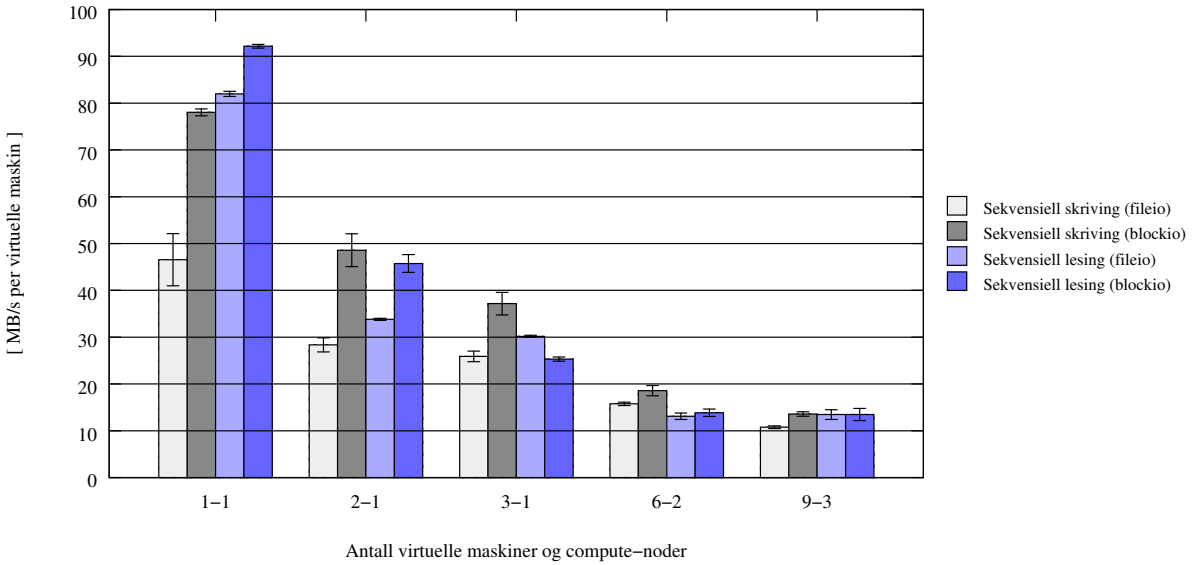
Virtuelle maskiner



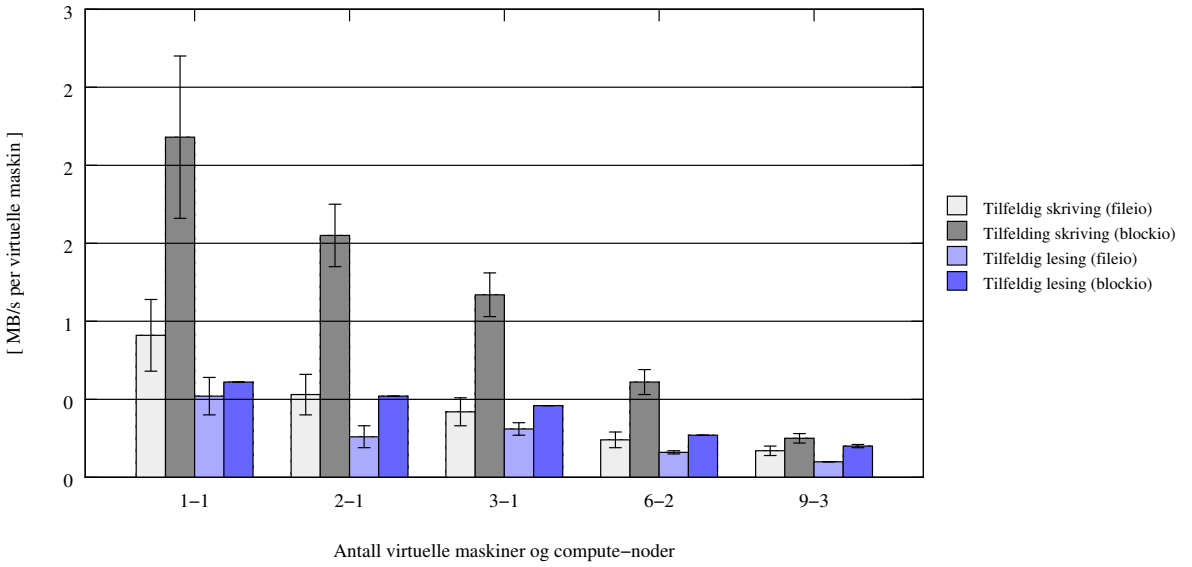
Figur 7.10: Diskytelse VM: Compute-node som oppbevaringssted

# VM	# Compute	Lokal disk	Skriv	Les	Tilfeldig skriv	Tilfeldig les
1	1	Compute	13.53 (2.10) 0.31ms	56.13 (0.56) 0.07ms	0.68 (0.44) 5.75ms	0.47 (0.10) 33.44ms
2	1	Compute	6.56 (0.52) 0.63ms	32.01 (1.93) 0.13ms	0.28 (0.15) 18.17ms	0.38 (0.08) 279.91ms
3	1	Compute	4.84 (0.26) 0.88ms	17.39 (0.88) 0.23ms	0.26 (0.09) 19.30ms	0.23 (0.05) 823.24ms

Tabell 7.6: Diskytelse VM: Compute-node som oppbevaringssted



Figur 7.11: Diskytelse VM: Sekvensiell aksess



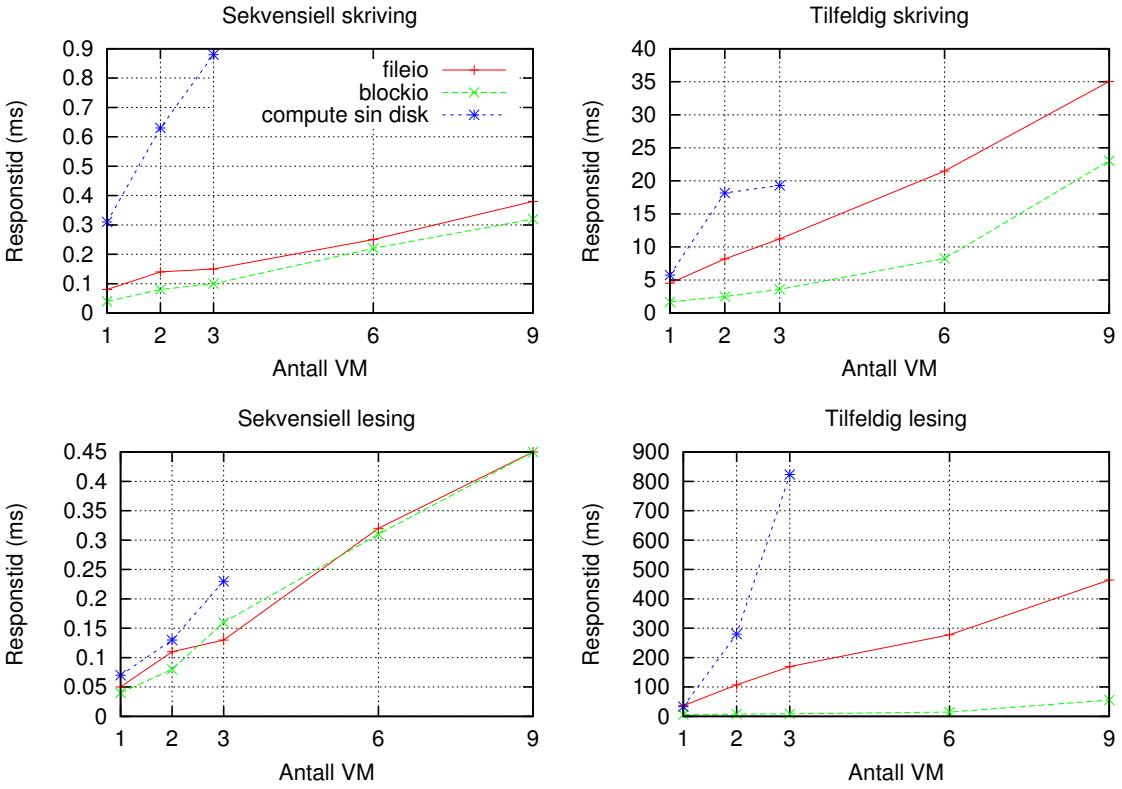
Figur 7.12: Diskytelse VM: Tilfeldig aksess

# VM	# Compute	Lokal disk	Skriv	Les	Tilfeldig skriv	Tilfeldig les
1	1	fileio	47.93 (4.42) 0.08ms	82.29 (0.49) 0.05ms	0.91 (0.23) 4.51ms	0.52 (0.12) 36.73ms
2	1	fileio	28.36 (1.51) 0.14ms	33.81 (0.22) 0.11ms	0.53 (0.13) 8.20ms	0.26 (0.07) 107.54ms
3	1	fileio	25.89 (1.12) 0.15ms	30.18 (0.22) 0.13ms	0.42 (0.09) 11.21ms	0.31 (0.04) 169.60ms
6	2	fileio	15.75 (0.35) 0.25ms	13.11 (0.68) 0.32ms	0.24 (0.05) 21.49ms	0.16 (0.01) 178.05ms
9	3	fileio	10.76 (0.30) 0.38ms	13.47 (1.04) 0.45ms	0.17 (0.03) 35.06ms	0.10 (0.00) 464.23ms

Tabell 7.7: Diskytelse VM: iSCSI-fileio som oppbevaringssted

# VM	# Compute	Lokal disk	Skriv	Les	Tilfeldig skriv	Tilfeldig les
1	1	blockio	75.13 (5.75) 0.04ms	93.70 (1.02) 0.04ms	2.18 (0.52) 1.68ms	0.61 (0.00) 6.56ms
2	1	blockio	48.58 (3.49) 0.08ms	45.74 (1.89) 0.08ms	1.55 (0.20) 2.50ms	0.52 (0.00) 7.68ms
3	1	blockio	37.16 (2.40) 0.10ms	25.32 (0.44) 0.16ms	1.17 (0.14) 3.59ms	0.46 (0.00) 8.59ms
6	2	blockio	18.58 (1.09) 0.22ms	13.84 (0.79) 0.31ms	0.61 (0.08) 8.27ms	0.27 (0.00) 14.54ms
9	3	blockio	13.57 (0.48) 0.32ms	13.48 (1.29) 0.45ms	0.25 (0.03) 23.03ms	0.20 (0.01) 55.66ms

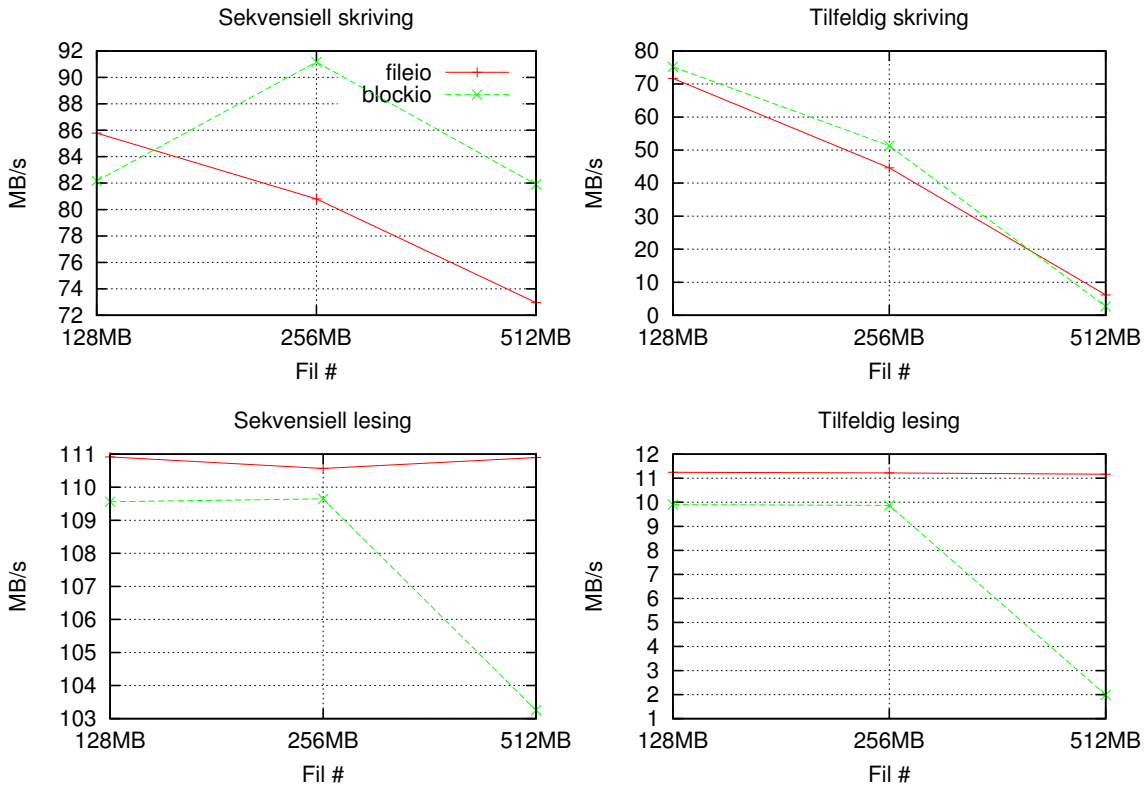
Tabell 7.8: Diskytelse VM: iSCSI-blockio som oppbevaringssted



Figur 7.13: Responstider for disk for virtuelle maskiner

Figur 7.10 og tabell 7.6 viser diskytelsen for et antall (inntil 9) virtuelle Linux-maskiner når de flate filene de ser på som sin lokale disk befinner seg på compute-noden sin fysiske harddisk. Figur 7.11 og 7.12 samt tabell 7.7 og 7.8 viser derimot diskytelsen for virtuelle Linux-maskiner når de flate filene de ser på som sin lokale disk befinner seg på lagringsnoden som kommuniserer med compute-noden over iSCSI. Utvikling i responstid kan observeres i figur 7.13.

Operasjoner på mindre filer



Figur 7.14: Diskytelse VM: Operasjoner på mindre filer

Filstørrelse	iSCSI-oppsett	Skriv	Les	Tilfeldig skriv	Tilfeldig les
128MB	fileio	85.79 (41.29) 0.05ms	110.92 (1.10) 0.03ms	71.69 (32.47) 0.04ms	11.25 (0.13) 0.35ms
256MB	fileio	80.81 (17.85) 0.05ms	110.57 (0.72) 0.03ms	44.61 (7.01) 0.06ms	11.22 (0.09) 0.35ms
512MB	fileio	72.96 (6.87) 0.05ms	110.90 (0.73) 0.03ms	6.16 (1.56) 0.66ms	11.17 (0.14) 0.35ms
128MB	blockio	82.16 (9.19) 0.04ms	109.56 (1.36) 0.03ms	75.21 (9.51) 0.04ms	9.90 (0.10) 0.40ms
256MB	blockio	91.17 (20.15) 0.04ms	109.65 (0.66) 0.03ms	51.30 (9.86) 0.07ms	9.87 (0.14) 0.40ms
512MB	blockio	81.90 (9.29) 0.04ms	103.25 (0.71) 0.04ms	2.68 (0.69) 1.41ms	1.99 (0.04) 2.01ms

Tabell 7.9: Diskytelse VM: Operasjoner på mindre filer

Figur 7.14 og tabell 7.9 viser diskytelse for en virtuell maskin når filene det testes med er mindre, henholdsvis 128MB, 256MB og 512MB.

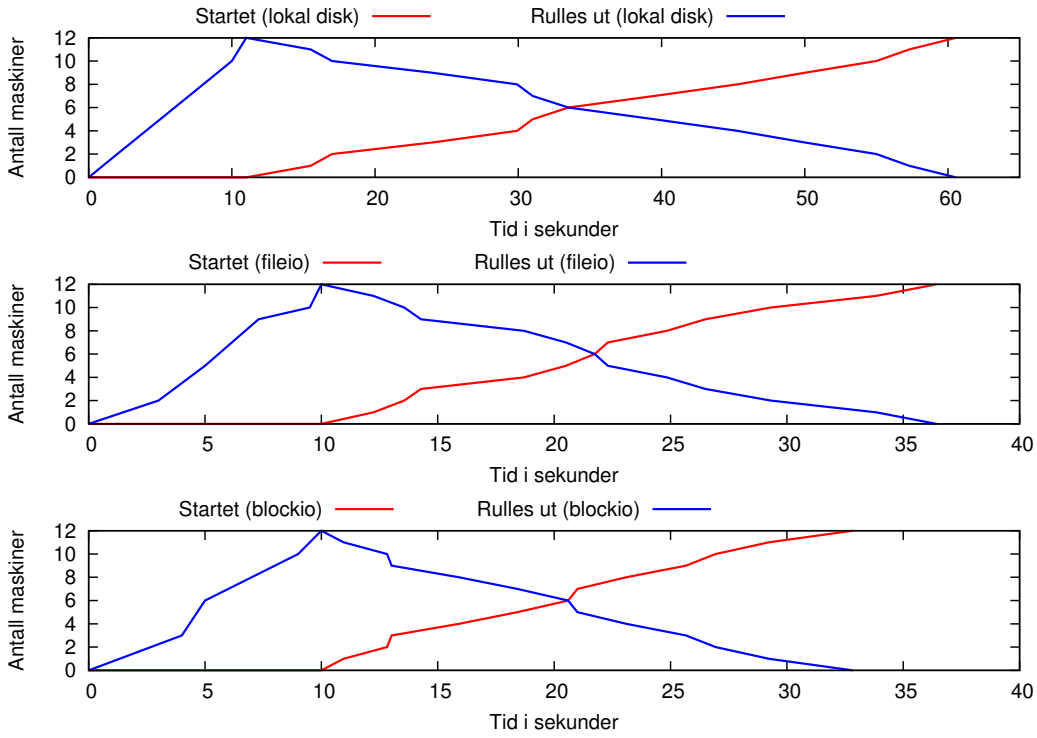
Diskytelse ved ikke-standard konfigurasjonsvalg: jumbo frames og flere iSCSI-tråder

Maskin	Jumbo	iSCSI-oppsett	Skriv	Les	Tilfeldig skriv	Tilfeldig les
1 fysisk	Ja	8 tråder fileio	119.13 (0.27) 0.03ms	115.06 (0.14) 0.03ms	3.95 (1.69) 1.29ms	1.27 (0.49) 4.93ms
	Ja	16 tråder fileio	118.99 (0.16) 0.03ms	114.81 (0.49) 0.03ms	4.32 (0.80) 0.90ms	1.30 (0.53) 1.86ms
	Nei	16 tråder fileio	113.24 (0.56) 0.03ms	109.63 (0.44) 0.03ms	4.43 (0.43) 0.91ms	1.19 (0.31) 1.07ms
1 virtuell	Ja	8 tråder fileio	48.35 (6.58) 0.08ms	92.03 (2.92) 0.04ms	1.35 (0.57) 3.04ms	1.75 (0.48) 2.79ms
	Ja	16 tråder fileio	57.49 (5.78) 0.07ms	90.32 (1.93) 0.04ms	1.35 (0.59) 3.35ms	2.00 (0.29) 2.72ms
	Nei	16 tråder fileio	46.46 (4.37) 0.08ms	81.08 (0.37) 0.05ms	0.88 (0.25) 4.48ms	0.54 (0.08) 18.52ms
1 fysisk	Ja	8 tråder blockio	119.41 (0.21) 0.03ms	114.77 (0.44) 0.03ms	4.45 (0.42) 0.91ms	0.60 (0.12) 8.59ms
	Ja	16 tråder blockio	115.83 (1.02) 0.03ms	114.19 (0.14) 0.03ms	4.41 (0.46) 0.92ms	0.59 (0.00) 6.79ms
	Nei	16 tråder blockio	113.21 (0.15) 0.03ms	109.35 (0.39) 0.03ms	4.21 (0.94) 0.94ms	0.55 (0.13) 10.50ms
1 virtuell	Ja	8 tråder blockio	74.75 (8.03) 0.05ms	96.33 (0.39) 0.04ms	1.54 (0.49) 2.74ms	0.56 (0.08) 7.96ms
	Ja	16 tråder blockio	75.17 (7.92) 0.05ms	97.16 (0.62) 0.04ms	1.56 (0.45) 2.54ms	0.56 (0.08) 7.35ms
	Nei	16 tråder blockio	75.65 (8.17) 0.05ms	93.91 (0.30) 0.04ms	1.54 (0.52) 2.54ms	0.55 (0.07) 7.60ms

Tabell 7.10: Diskytelse: Jumbo Frames og/eller flere iSCSI-tråder)

Tabell 7.10 viser diskytelse for fysiske og virtuelle maskiner når ikke-standard konfigurasjonsvalg er benyttet. Jumbo frames testes med henholdsvis iSCSI-blockio og fileio, og 8 og 16 iSCSI-tråder, for henholdsvis én fysisk maskin og én virtuell maskin.

Utrulling av virtuelle Linux-maskiner



Figur 7.15: Utrullingshastighet for 12 virtuelle Linux-maskiner

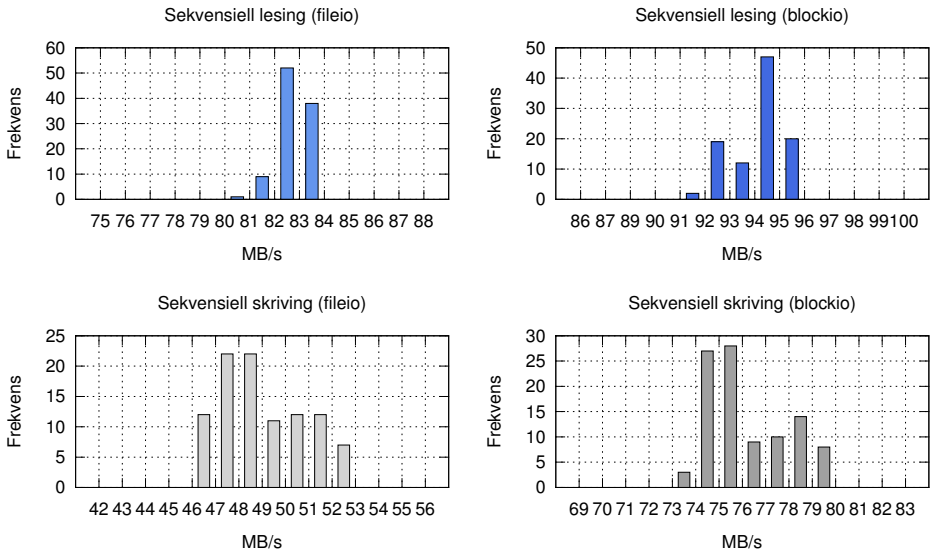
Oppsett	Minstetid	Makstid	Gjennomsnittsutrullingstid	Standardavvik
lokal disk	55s	65s	60.4s	2.5s
fileio	32s	43s	36.4s	2.5s
blockio	28s	44s	32.8s	3s

Tabell 7.11: Utrullingshastighet for 12 virtuelle Linux-maskiner

Figur 7.15 og tabell 7.11 viser gjennomsnittlig utrullingstid for 12 virtuelle Linux-maskiner fordelt på 3 compute-noder. Utrulling vil si kopiering og opprettelse av filene som utgjør de virtuelle maskinene (QCOW2-images),

oppstart av maskinene og et fungerende nettverk. Med andre ord, en maskin definert som "startet" er det mulig å nyttiggjøre.

Undersøkelse av normalfordeling



Figur 7.16: Undersøkelse av normalfordeling

Figur 7.16 viser utvalgte testresultater satt i et histogram med den hensikt å undersøke om resultatene kan sies å være normalfordelte. Testresultatene valgt å representere i de fire histogrammene er sekvensiell skriving og lesing for én virtuell maskin med henholdsvis fileio og blockio som iSCSI-konfigurasjon (se tabell 7.7 og 7.8). Det er ikke brukt noen desimaler, og alle data er innenfor det området på x-aksen som kan observeres. Selv om histogrammene tilsynelatende ikke følger perfekte gauss-kurver som kjennetegner normalfordeling fra lærebøker, ser man ut fra utsnittet at dataene er nærliggende hverandre, med distinkte topper.

Kapittel 8

Diskusjon og anbefalinger

Ytelse for nettverk

Fysisk maskin

Nettverksytelsen for de fysiske maskinene er ikke uventet helt der man skulle forvente den i et Gigabit-svitsjet nettverk. Det er liten variabilitet i resultatene, og man oppnår 941MB/s i skriving og lesing, eller ca. 117.5MB/s. Den teoretisk oppnåelige hastigheten er 125MB/s, men noe går med til såkalt "overhead", altså kontrollinformasjon som ikke er nyttedata. Med dette bekreftet ble deretter link aggregation undersøkt.

Når flere compute-noder belaster lagringsnoden samtidig – og lagringsnoden har link aggregation konfigurert – ser man at standardavviket blir høyere når det er flere enn to maskiner som samtidig kommuniserer med den, både for hastighet per maskin og den maksimalt oppnådde hastigheten, ettersom det da blir en konflikt om ressurser, noe som skaper variabilitet. Det ser imidlertid ut til at en klarer å komme veldig nære å utnytte begge kortene maksimalt, gitt at det mest sannsynlig er noen feilmarginer.

Fra resultatene kan man også se at compute-nodene bruker mindre prosesseringskraft dess flere som belaster lagringsnoden, dette fordi det da blir færre ressurser å utnytte per compute-node. Dette gjelder både for sending og mottak. For lagringsnoden derimot, brukes det omtrent like mye ressurser for mottak, mens det å sende blir mer kostbart som følge av flere compute-noder som belaster.

Det er en kraftig økning i prosessorbelastning på lagringsnode mellom sending til én compute-node og sending til to compute-noder. Deretter ser man en mindre økning for hver nye compute-node som legges til. Den store økningen fra én til to noder kommer sannsynligvis av at ved sending til flere enn

én compute-node bruker lagringsnoden ekstra kraft på prosessering av link aggregation, noe som opptar mer CPU. En annen modus for link aggregation kan potensielt redusere denne lasten noe.

Virtuell maskin

I målingene for nettverksytelse hos individuelle virtuelle Linux-maskiner med forskjellige nettverksdrivere konfigurert kommer det frem at med standard-driveren 'rtl8139', som er en programvare-emulert utgave av et Realtek 8139 10/100Mbit-nettverkskort, oppnås cirka 100Mbit for sending , mens mottak fra lagringsnode er i sjiktet 275Mbit fra hypervisor og 334Mbit fra lagringsnoden.

Den samme testen utført med 'e1000', et programvare-emulert Intel E1000-nettverkskort, gir nærmere 275MBit i sending og omtrent det dobbelte av mottakshastigheten kontra 'rtl8139'. Dette er imidlertid langt fra 1000Mbit som navnet skilter med. Standardavviket for mottak er noe høyere enn sending. Med virtio-driveren, som skal kunne gi ytelse sammenlignbar med den fysiske maskinen, oppnås i praksis full Gbit-hastighet mot lagringsnode. Resultatene for virtio er noe bedre enn funnene i studien til Shafer [37], noe som sannsynligvis kan attribueres annen maskinvare og programvare benyttet. Når man tester mot hypervisor med virtio oppnås kunstig høye resultater, sannsynligvis som følge av hurtiglagring i minnet.

I tabell 8.1 vises resultater for nettverksytelse for sending og mottak til og fra lagringsnoden for for de ulike nettverksdriverene gruppen har undersøkt. Resultatene til de ulike nettverksdriverene er normalisert opp mot den oppnåelige nettverksytelsen på fysiske maskiner (som tilsvarer 1 – 941MBit/s eller 117.5MB/s). For responstider er 1 også best (oppnådd med virtio), mens dårligere (altså høyere rent tidsmessig) resultater oppgis som dårligere enn dette. Dette for å være konsistent med tabellens andre resultater. Denne fremstillingen gjør det meget tydelig at virtio er å foretrekke.

Nettverksdriver	Sending	Mottak	Responstid
RTL8139	0.11	0.36	0.79
E1000	0.30	0.50	0.90
Virtio	0.99	0.99	1

Tabell 8.1: Normaliserte resultater for VM-nettverksytelse

iSCSI-ytelse

Fra resultatene for iSCSI-ytelse for fysiske maskiner kan man se at ingen av alternativene klarer å utnytte det som ble funnet å være maksimal oppnåelig nettverkshastighet, som er 117.5MB/s i praksis med standardinnstillinger. Det er ingen markant forskjell i ytelsen for fileio og blockio. Når tre fysiske maskiner gjør operasjoner parallelt, og link aggregation er konfigurert, klarer man å utnytte ca. 78% av nettverket med fileio og 76% med blockio for lesing, og 77% av nettverket med fileio og 80% med blockio for skriving. På bakgrunn av dette ser det ut til at iSCSI (med standardinnstillinger) kan være flaskehalsen.

I tabell 8.2 ser man utnyttelsen av nettverket for de ulike scenariene. Maksimalt oppnåelig hastighet er 235MB/s (1882Mbit/s) når det er to eller flere maskiner og 117.5MB/s (941Mbit/s) når det er én som kommuniserer med lagringsnoden. Blockio gjør det noe bedre, men det er meget jevnt for alle scenariene.

iSCSI	# fysiske maskiner	Sekvensiell skriving	Sekvensiell lesing
fileio	1	94,32%	91,03%
blockio	1	94,84%	92.55%
fileio	2	75.69%	74.36%
blockio	2	78.53%	69.68%
fileio	3	77.18%	78.10%
blockio	3	80.26%	76.43%

Tabell 8.2: Utnyttelse av maksimalt teoretisk oppnåelig nettverkskapasitet

Diskytelse for fysiske og virtuelle maskiner

Det observeres at ytelsen for compute-noder og controlleren ikke overraskende er veldig lik, på grunn av samme maskinvare. Ytelsen for sekvensiell skriving er forholdsvis lav, noe som skyldes raidkontrolleren som benyttes. Lagringsnoden sin lese- og skriveytelse er god. Leseytelsen er tilsvarende to Gbit-nettverkskort satt opp med link aggregation (250MB/s, som i praksis blir lavere på grunn av overhead).

For alle maskinene er tilfeldig skriving og lesing veldig dårlig, hvilket forklares med at disse operasjonene er mekaniske harddiskers akilleshæl grunnet de fysiske og operasjonelle karakteristika ved slike enheter.

Ved sammenligning av tabellene 7.6, 7.7 og 7.8 ser man at ved oppbevaring av virtuelle maskiner lokalt på compute-nodene er ytelsen veldig lav sammenlignet med oppbevaring over iSCSI på lagringsnoden.

Ved tilfeldig lesing later lagringsnoden til å ha kapasitet til mellom fire og seks VM-er ved fileio og mellom seks og ni VM-er ved blockio, før ytelsen går ned på samme nivå som med tre VM-er lokalt på hver compute. Hvorvidt dette gir gode opplevelser for sluttbrukere må undersøkes nærmere.

Det ser ikke ut til å være stor ytelsesvariabilitet selv om antallet virtuelle maskiner i infrastrukturen økes hvis man skal dømme ut i fra standardavviket. Utnyttelse av iSCSI med blockio når ikke opp til nivået for tre fysiske compute-noder, men øker fra 75MB/s når det er én virtuell maskin; 97MB/s når det er to og ca 111MB/s når det er tre og seks og 122MB/s når det er 9. Fileio ligger en del under hele veien.

Når det kommer til responstider ved tilfeldig lesing, som er den parameteren som tenderer til å øke mest ved økning av antall virtuelle maskiner, er lagring over iSCSI betydelig bedre enn lokalt på compute. Med tre virtuelle maskiner i bruk har fileio 170ms mellom hver gang en operasjon fullføres, blockio 8.59ms og lokal disk på compute-noden 823ms. Blockio har på dette punktet langt bedre responstider enn begge alternativene, og dette gjenspeiler seg også for sekvensiell lesing og skriving samt tilfeldig skriving.

	Compute sin disk	iSCSI-fileio	iSCSI-blockio
Sekvensiell skriving	49% av 27.42MB/s	42% av 110.83MB/s	70% av 111.44MB/s
Sekvensiell lesing	86% av 65.58MB/s	77% av 106.97MB/s	85% av 108.75MB/s
Tilfeldig skriving	11% av 5.34MB/s	12% av 7.31MB/s	29% av 7.44MB/s
Tilfeldig lesing	130% av 0.36MB/s	69% av 0.25MB/s	127% av 0.48MB/s

Tabell 8.3: Normaliserte resultater for VM-diskytelse

I tabellen 8.3 vises resultater for diskytelse for én virtuell Linux-maskin avhengig av hvor den virtuelle maskinen er lagret. Gjennomsnittet for den virtuelle maskinen er delt på gjennomsnittet for de fysiske maskinenes ytelse (compute-nodene) for angitt konfigurasjon. Tallene gjelder for testing med filer på 4GB. For tilfeldig lesing kan det med det blotte øye se ut som at den virtuelle maskinen er bedre. Standardavviket er derimot jevnt over høyere for de virtuelle maskinene, slik at resultatene må tas med forbehold. Man ser også at blockio kommer vesentlig bedre ut enn fileio ved større filer.

Jumbo frames og flere iSCSI-tråder

I tabell 7.10 ses en ytelsesgevinst for sekvensiell skriving på cirka 7% for både fileio og blockio ved bruk av jumbo frames fra fysisk maskin. Dette gjør at man utnytter nettverket bedre, men responstiden vil bli desto dårligere. For de virtuelle maskinene er det kun fileio som gir en liten økning i ytelse ved bruk av jumbo frames. Når det kommer til dobling av antall iSCSI-tråder fra

8 til 16 for fysiske maskiner er det ingen større forskjeller å spore hva gjelder ytelse.

Diskytelse for virtuelle maskiner ved operasjoner på mindre filer

Fileio er som forventet av gruppen raskere enn blockio ved sekvensiell og tilfelding lesing av små filer. Ved skriving ser man mindre forskjell mellom de to, som skyldes at det med våre tester er nytt innhold som skrives hver gang, mens det ved lesing er det samme innholdet for hver iterasjon. Fileio holder seg på samme nivå for alle tre filstørrelsene ved lesing, mens det kan se ut som blockio blir preget av at den går tom for minne når filstørrelsen er 512MB. Det kan man også se spor av i responstidene, hvor fileio har betydelig lengre responstid enn blockio, da spesielt ved tilfeldig lesing, som muligens kommer av introduseringen av et ekstra lag som samtidig gjør liten nytte for seg ved tilfeldig skriving.

Utrullingshastighet for 12 virtuelle Linux-maskiner

Å starte instanser over iSCSI er hurtigere enn fra lokalt på compute. Spesielt interessant er at det fortsatt er raskere selv om det da er 12 instanser som skal starte fra iSCSI, kontra 4 og 4 på hver compute (i overkant av 30 sekunder over iSCSI mot 60 sekunder lokalt og ikke-sentralisert). Det er marginale forskjeller mellom blockio og fileio.

Mulige flaskehalser

I infrastrukturen er det ikke antydning til at nettverket er noen flaskehals, annet enn hvis man velger en ugunstig nettverksdriver i den virtuelle maskinen. iSCSI klarer kun å mette nettverket ved noen tilfeller for fysiske maskiner når jumbo frames benyttes, men ikke ellers, noe som tyder på at iSCSI i seg selv kan være en flaskehals. Andre innstillinger og versjoner av programvaren kan imidlertid gjøre at dette stiller seg annerledes. I ingen av tilfellene klarer de virtuelle maskinene å tangere de fysiske når det kommer til diskytelse, se for eksempel tabell 8.3.

Det må også nevnes at compute-nodene ikke har det ypperste hva gjelder maskinvare, slik at det kan være en begrensende faktor for en implementasjon av SkyHiGh i større skala. I forhold til CPU-bruk på compute-nodene mens I/O er testet med ulike antall virtuelle maskiner, er det jevnt over mest I/O-venting å spore. Dette betyr at CPUen ikke gjør prosessering men venter

på at I/O skal fullføre. Dette har imidlertid med at oppgaven har benyttet syntetisk ytelsestesting, hvor det ikke er annen prosessering (vitenskapelige beregninger eller kalkulasjoner for eksempel) som foregår.

Anbefalinger

Selv om de eksakte resultatene ikke kan sies å ha gyldighet for andre infrastrukturer, vil likevel trendene for de ulike alternativene som er testet i denne bacheloroppgaven også kunne gjelde for andre, gitt den samme programvaren. Videre vil gruppen gi noen konkrete anbefalinger basert på de resultatene som er fremkommet, slik at en eventuell senere videreutvikling av SkyHiGh har beste praksis-råd som kan understøtte implementasjonen.

Å lagre de virtuelle maskinene på lagringsnoden som filer for så å aksessere dem over iSCSI er et klart bedre valg enn å plassere dem lokalt på compute-nodene slik infrastrukturen er i nåværende implementasjon. Dette kommer av at diskytelsen lokalt på compute-nodene er meget lav kontra iSCSI, hvilket kan være et resultat av at raidkontrolleren i disse ikke er like kraftig som lagringsnoden sin. Undersøkelse software-raid på compute-nodene kan avdekke om dette er tilfellet. Det er imidlertid ikke i sikkert man alltid ønsker sentraliserte filer, slik man får med iSCSI.

I forhold til hvor mange compute-noder og virtuelle maskiner som kan kommunisere mot samme lagringsnode kommer dette an på en forståelse for når ytelsen blir så lav at brukeren må vente lenger enn ønsket ved utrulling av ønsket maskin, oppstart av programmer på denne maskinen og daglig virksomhet på maskinen. Det har ikke vært del av oppgaven å beregne hvilke hastigheter som trengs for å gi tilfredstillende ytelse per virtuelle maskin, siden realistisk ytelsestesting ikke er utført. Hvorvidt noe oppleves som tregt vil måtte undersøkes ved hjelp av reelle brukersituasjoner. Det er imidlertid ikke sikkert at resultatene fra ytelsestestingen oppnås når brukerene også bruker maskinene allokert til annen enn ren I/O. Oppgaven har ikke testet ytelsen for I/O når også annen prosessering belaster de virtuelle maskinene samtidig, noe som potensielt vil influere resultatene i negativ retning.

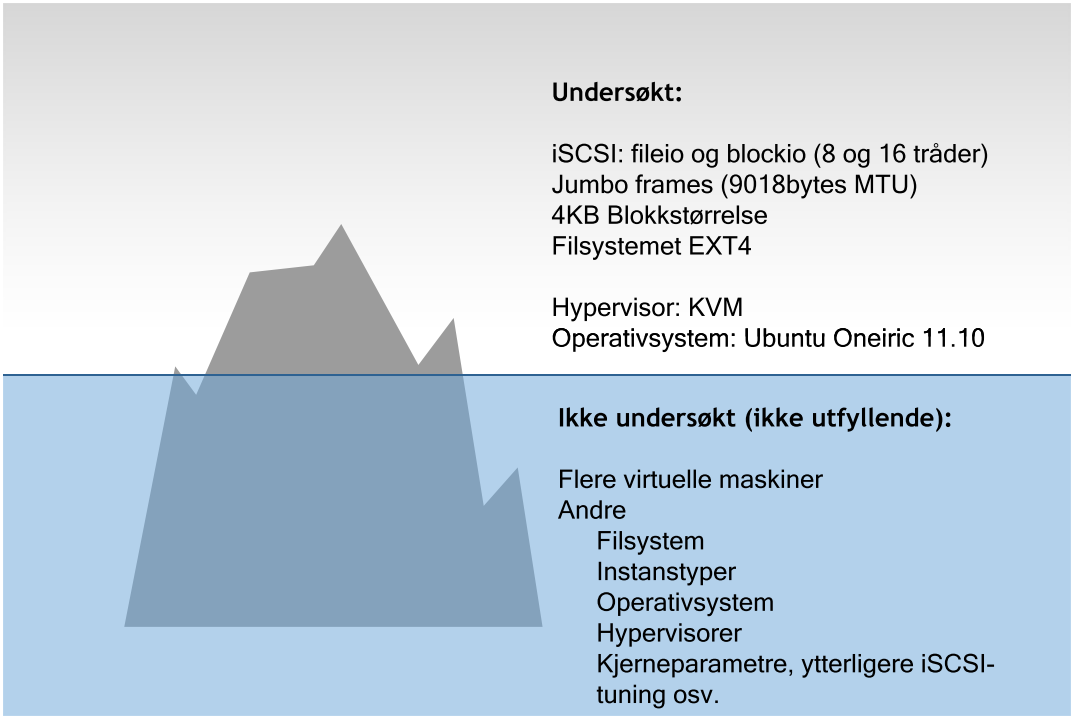
- **Link Aggregation:** Dersom ikke Gbit-nettverk strekker til og man har mulighet til å mette nettverket (altså at disk ikke er flaskehalsen), vil link aggregation være fornuftig å implementere, gitt at man har flere nettverkskort tilgjengelig. Ytelsesmessige fordeler vil avhenge av konfigurasjon og lagringsnodens maskinvarekarakteristika
- **Virtio:** Studier og praksis har demonstrert at virtio er en utmerket nettverksdriver. Shafer [37] viste også at den yter godt som diskdriver,

hvilket forklarer hvorfor det er standarddriveren i KVM sine maler. Virtio anbefales derfor for virtuelle Linux-maskiner i OpenStack, og mest sannsynlig også for Windows-maskiner

- **iSCSI:** Dersom compute-noders fysiske harddisker ikke gir tilstrekkelig god ytelse og man ønsker sentralisering av filene som tilhører de virtuelle maskinene, har gruppen demonstrert at iSCSI med fordel kan brukes som oppbevaringsplass for virtuelle maskiner. Flere tråder ser ikke ut til å gi god gevinst, noe som bekrefter utviklerenes råd om at standardinnstillingene i de fleste tilfeller er gode nok. Fileio kan gi bedre ytelse og responstider dersom lagringsnoden har tilstrekkelige mengder minne og prosesseringskraft til at data som ofte leses fra hurtiglagres. Blockio gir bedre ytelse og betydelig bedre responstider over hele datasettet for virtuelle maskiner, muligens på grunn av mindre overhead. En utredning av hva som defineres som høy responstid kan være fornuftig i forhold til hvor mange compute-noder og virtuelle maskiner som kan brukes av hver lagringsnode
- **Jumbo Frames:** Jumbo frames kan gi bedre throughput ved at forholdet mellom nyttedata og kontrollinformasjon forskyves til at det blir færre pakker totalt sett og dermed desto mer nyttedata per pakke. Dette kan derimot øke responstiden, slik at eventuell implementasjon vil avhenge av ytelsesmessige behov

Oppgavens avgrensninger

I figur 8.1 vises en fremstilling av hva som er testet i denne oppgaven samt elementer som man kan ha nytte av å undersøke i fremtiden.



Figur 8.1: Fremstilling av den avgrensede oppgavens testinnhold

Kapittel 9

Konklusjon og evaluering

Evaluering av arbeid

Organisering

Deltakerene har hatt jevn og tilfredsstillende kontakt med både oppdragsgiver og veileder gjennom hele oppgaveløpet. Dette har vært bidragsytende til forståelse for oppgavens innhold, i tillegg til at det har vært en stor motivasjonsfaktor.

Gruppearbeid

Arbeid i grupper kan både være positivt og negativt. Av positive effekter kan nevnes at man får utnyttet hver enkelt gruppedeltakers sterke sider og en større kunnskapsbase i bunn, i kraft av at man er flere. Det kan også være negativt i form av konflikter, men gruppen har over det hele samarbeidet godt uten noen tilløp til konflikt. Gruppemedlemmene har også samarbeidet på prosjekter tidligere med stort hell, slik at prosjektgjennomføringen har vært uproblematisk i så henseende. De tildelte prosjektrollene viste seg å ikke bli nyttiggjort i stor grad, slik at det aller fleste oppgaver ble utført samarbeidsvis.

Hva kunne vært gjort bedre

En kjensgjerning observert i løpet av oppgavens gang er at mye kunne ha vært gjort annerledes. Ett betydelig moment er at det innledningsvis ble brukt store mengder tid på kildesøk og litteraturstudier på teknologier som til syvende og sist ikke var relevante for den etter hvert avgrensede oppgaven.

Dette hadde mye med at oppgaven ikke var ferdigkonkretisert og nokså åpen før etter en del tid var gått. I så måte kunne betydelige mengder tid vært spart og fokus rettet mot andre, viktigere områder. Samtidig var ikke denne tiden fullstendig bortkastet, ettersom gruppen kom over en hel del interessante artikler og forskningslitteratur som ellers ikke ville ha blitt oppdaget. Den andre gruppen brukte også halvannen måned lenger enn forventet på å få installert infrastrukturen. Dette skyldtes forhold som lav modningsgrad for OpenStack med oppdragsgivers prefererte hypervisor og Linux-distribusjon, hvilket førte til store stabilitetsmessige problemer rundt implementasjonen.

Et annet etterpåklokskapsmoment er at det ble skrevet unødvendig mange script på grunn av usikkerhet i forhold til hvilket diskytelsestestingsverktøy som skulle benyttes. Tidlig valgte gruppen Bonnie++ til å fylle denne rollen basert på tidligere kjennskap til programvaren, men det ble oppdaget at parallellkjøring (flere virtuelle maskiner som kjører tester simultant) fører til resultater som ikke kan stoles på. I forkant av oppdagelsen av Flexible IO Tester laget gruppen også script for automatisk testing med verktøyet "dd", men dette lar deg ikke teste tilfeldig aksess. Gruppen lærte imidlertid mye av denne – om noe unødvendige – ekstraarbeidsbelastningen. Det ble også gjort en del scripting for automatisering av verktøyet "Seekmark", som er et program for testing av responstider for harddisker, men disse scriptene ble også forkastet ettersom FIO sine utdata allerede inneholder informasjon om dette. Når FIO ble valgt som ytelsestestingsverktøy for disk, kunne imidlertid mye bli gjenbrukt av den tidligere skrevne koden.

Det at gruppen gikk fra å bruke masse tid på å prøve å automatisere alt av testing og grafgenerering i starten til deretter å gjøre ting mer manuelt mot slutten ga også nyttig lærdom om at automatisering ikke nødvendigvis alltid er best, selv om det er effektivt når det først fungerer. Gruppens erfaring er at man bør automatisere de trivielle oppgavene som utføres flest ganger. Overstadig automatiseringsiver kan føre til unødig kompleksitet.

Gruppen ønsket også å kjøre tester med Windows, som kun ville bli testet dersom tidsmessige hensyn tillot det, men kom til kort med dette. Forklaringen er at Windows ikke er like trivielt som Linux å få implementert med OpenStack, slik at det ble gitt opp av den andre gruppen helt til det gjenstod to uker før innlevering, til fordel for mer prekære og umiddelbare hensyn tilknyttet oppgaven. Fokus ble derfor dreid helhjertet på testing av Linux-maskiner. Det ble heller ikke anledning til å teste med rotfilsystemet til en virtuell maskin på et LVM-volum eller en partisjon grunnet stor motvilje fra OpenStack-versjonen som er implementert, og det var ikke spesielt ønskelig å oppgradere når infrastrukturen først etter to og en halv måned var fullt og helt operasjonell. Det er også underforstått at resultatene fremkommet er gjeldende for den testmetodikken som det ble undersøkt med. Ingen av de virtuelle maskinene var beskjeftiget med annet enn å utføre syntetis-

ke I/O-tester under undersøkelsene. Med høy sannsynlighet ville resultatene blitt dreid i annen, negativ retning dersom også andre oppgaver som belaster andre deler av de underliggende fysiske maskinene kjørte samtidig. Gruppen burde også ha testet med flere forskjellige filstørrelser da M. Satyanarayanan i "A Study of File Sizes and Functional Lifetimes" [48] demonstrerer at de aller fleste filer i filsystemer er veldig små. I tillegg burde vi forsøkt med flere modifiserte instanstyper, se tabell 3.3. Likevel har nyttig informasjon fremkommet som viser definitive forskjeller mellom fysiske og virtuelle maskiner.

Mulige videre arbeid

Denne oppgaven har sett på nettverksytelse samt diskytelse for virtuelle maskiner når de lagres som flate filer og befinner seg på iSCSI eller hypervisoren sin fysiske harddisk. Videre arbeid rettet mot undersøkelse av ytelse bør definitivt kikke på hvordan den påvirkes når de virtuelle maskinene sitt rotfilsystem er et LVM-volum eller en partisjon delt over iSCSi, noe som kan føre til ytelsesforbedringer. Det kan også være lønnsomt å forsøke med andre filformater som de virtuelle maskinene lagres i.

Andre interessante og mulige områder å undersøke for å se hvilke negative eller positive påvirkninger de har:

- **Hypervisor:** I oppgaven er KVM benyttet, men også Xen og Vmware er muligheter
- **Operativsystem/Linux-distribusjon:** Gruppen fikk ikke testet med Windows, da det ikke ble implementert før innlevering nærmet seg og annet arbeid var prioritert. Årsaken til vanskene med implementering skyldes dårlig dokumentasjon som ikke rekker å oppdateres mellom hver gang det skjer store omveltninger i koden til OpenStack. Det er også kun testet med versjon 11.10 av Ubuntu
- **Grafisk fjernaksess:** Grunnet problemer med å få til Windows-maskiner og fjernaksess i form av VNC og RDP ble ikke grafisk klientaksess til de virtuelle maskinene undersøkt. Dette var imidlertid ikke en prioritering for oppdragsgiver, og det ville kun blitt allokert tid til dette hvis det pragmatisk sett var mulig. Likevel er det et viktig område, da ikke alle oppgaver på komfortabelt vis kan utføres med kommandoskall, og ikke alle brukere er like fortrolige i systemmiljøer som helhjertet kontrolleres på denne måten

Oppgaveutførelse i lys av prosjektmål

Gruppen vil her ta frem prosjektmålene fra innledningen og vurdere hvorvidt det er klart å komme frem til gode svar og løsninger på disse.

Effektmål

Av effektmål var det ønsket å redegjøre for om sluttbrukere ville få en bedre opplevelse hvis man implementerer anbefalinger utledet fra oppgaveresultatene, og om denne nettskyen kan betraktes som "moden". Svarene på disse spørsmålene er sammensatte. Gruppen har imidlertid bekreftet flere beste praksis-anbefalinger, blant annet knyttet til nettverksdriver. Valg av den anbefalte nettverksdriveren vil gjøre at ytelsen er sammenlignbar med fysiske maskiner hva gjelder nettverk. I forbindelse med diskytelse er det også redegjort for beste praksis for den gjeldende infrastrukturen, noe som har potensiale til å øke ytelsen sluttbrukeren opplever. Som innledningen foreslo finnes det imidlertid ingen svar en kan sette to streker under.

Implementering av anbefalingene vil gi bedre ytelse enn standardoppsettet, men at skyløsningen sett fra et I/O-synspunkt er moden for produksjon er ikke gitt. For dette finnes det ikke noe klart svar og mye av opplevelsen sluttbrukere opplever vil avhenge av hvilken type og mengde arbeidsbelastning som vil forekomme.

Læringsmål

Læringsmålene var relatert til tilegning av kunnskaper innen teknologi relevant for oppgaven, samt erfaringer rundt det å planlegge og gjennomføre større prosjektoppgaver. Gruppen har økt sin kunnskap på de tilsiktede områdene, og blitt mer bevandret innen nettskyteknologi, virtualisering, ytelse- og ytelsesvurderinger og statistikk samt scripting. Prinsippene beskrevet i kapittel 6 knyttet til den praktiske gjennomføringen av ytelsestestingen er fulgt samvittighetsfullt, og deltakerene har i tillegg ervervet verdifull kunnskap rundt prosjektoppgavegjennomføring.

Resultatmål

Resultatmålene er et produkt av effektmålene, hvor gruppen skulle komme frem til beste praksis-anbefalinger for SkyHiGh for å utnytte potensialet i nettskyens arkitektur best mulig i forhold til lagring og nettverk, og å vurdere om ytelsen er god nok til at sluttbrukere ikke vil se til andre alternativer.

For det første målet har gruppen kommet frem til et oppsett av nettverks- og harddiskdrivere, samt innstillinger for nettverk og iSCSI som har et potensiale til å forbedre ytelsen i forhold til standardoppsettet.

Hva gjelder opplevelsen til sluttbrukere kan gruppen konstatere at sluttbrukere må forvente et tap av opplevd ytelse ved virtualisering kontra fysiske maskiner på bakgrunn av de testdataene som har fremkommet.

Konklusjon

Oppgaven har resultert i en dypere forståelse for de I/O-messige utfordringene som eksisterer for virtualisering og nettskyer. De bakenforliggende teknologiene er også gjennomgått etter fordypning i aktuell litteratur og forskningsmateriell.

Gruppen hadde et åpent sinn i forhold til hvilke resultater som ville fremkomme, og har utført praktisk ytelsestesting og demonstrert at ytelse for disk og nettverk i den typen virtuelle maskiner som er testet er tilkortkommen målt opp mot hva som er oppnåelig på den ikke-virtualiserte maskinvaren de befinner seg på. Ytelsen som oppnås er tydelig sensitiv til nettverksdriver og hvilken konfigurasjon som velges for lokal disk. Relevante beste praksisanbefalinger fra studier og den gjennomførte ytelsestestingen er samlet inn og redegjort for, slik at ytelse kan optimaliseres ved videreutvikling av SkyHiGh.

Bibliografi

[1] D.E.Y. Sarna. *Implementing and Developing Cloud Computing Applications*. An Auerbach book. Auerbach Publishers, Incorporated, 2010.

[2] Timothy Grance Peter Mell. The nist definition of cloud computing. `http://csrc.nist.gov/publications/nistpubs/800-145/SP800-145.pdf`, April 2012.

[3] Kynetix. Cloud computing - a strategy guide for board level executives. `http://download.microsoft.com/download/1/5/D/15DA1ED7-6005-4D18-A592-12EA315A3F4A/KynetixCloudComputingStrategyGuide.pdf`, April 2012.

[4] Gerald J. Popek and Robert P. Goldberg. Formal requirements for virtualizable third generation architectures. *Commun. ACM*, 17(7):412–421, July 1974.

[5] Simon Crosby and David Brown. The virtualization reality. *Queue*, 4(10):34–41, December 2006.

[6] IBM. Virtualization. `http://download.microsoft.com/download/1/5/D/15DA1ED7-6005-4D18-A592-12EA315A3F4A/KynetixCloudComputingStrategyGuide.pdf`, April 2012.

[7] IBM. Best practices for kvm. `http://publib.boulder.ibm.com/infocenter/lnxinfo/v3r0m0/topic/liaat/liaatbestpractices_pdf.pdf`, April 2012.

[8] linux kvm.org. Virtio - kvm. `http://www.linux-kvm.org/page/Virtio`, April 2012.

[9] Paul Barham, Boris Dragovic, Keir Fraser, Steven Hand, Tim Harris, Alex Ho, Rolf Neugebauer, Ian Pratt, and Andrew Warfield. Xen and the art of virtualization. *SIGOPS Oper. Syst. Rev.*, 37(5):164–177, October 2003.

[10] Todd Deshane, Zachary Shepherd, J N Matthews, M Ben-Yehuda, A Shah, and B Rao. Quantitative comparison of xen and kvm. *Interna-*

tional Conference On Architectural Support For Programming Languages And Operating Systems Asplos, (Cli):23–25, 2008.

[11] Johan De Gelas. Hardware virtualization: the nuts and bolts. `http://impact.asu.edu/cse591sp11/HardwareVirtualizationAnandTech.pdf`, April 2012.

[12] Haining Wang Duy Le, Hai Huang. Understanding performance implications of nested file systems in a virtualized environment. `http://static.usenix.org/events/fast12/tech/full_papers/Le.pdf`, April 2012.

[13] David A. Patterson, Garth Gibson, and Randy H. Katz. A case for redundant arrays of inexpensive disks (raid). In *Proceedings of the 1988 ACM SIGMOD international conference on Management of data*, SIGMOD '88, pages 109–116, New York, NY, USA, 1988. ACM.

[14] Tldp.org. Lvm howto. `http://tldp.org/HOWTO/LVM-HOWTO`, April 2012.

[15] Andrew S. Tanenbaum. *Modern Operating Systems*. Pearson Education, Inc., Pearson Prentice Hall, Upper Saddle River, NJ 07458, USA, 3rd edition, 2009.

[16] Carl Waldspurger Mendel Rosenblum. I/o virtualization. `http://delivery.acm.org/10.1145/2080000/2071256/p30-rosenblum.pdf?ip=128.39.80.76&acc=OPEN&CFID=98948161&CFTOKEN=39795245&__acm__=1335264107_887c25beba1dcf77baab3555df88c542`, April 2012.

[17] Pctechguide.com. Hard disk (hard drive) performance – transfer rates, latency and seek times. `http://www.pctechguide.com/hard-disks/hard-disk-hard-drive-performance-transfer-rates-latency-and-seek-times`, April 2012.

[18] Linuxinsight. How fast is your disk? `http://www.linuxinsight.com/how_fast_is_your_disk.html`, April 2012.

[19] Intel. Serial ata ii native command queuing overview. `http://download.intel.com/support/chipsets/imsm/sb/sata2_ncq_overview.pdf`, April 2012.

[20] Satran J. Meth K.Z., IBM Haifa Research Lab. Features of the iscsi protocol. `http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=1222720`, April 2012.

[21] Yongjian Zhang. difference between fileio and blockio in iet. `http://old.nabble.com/newbie-question%`

3A-difference-between-fileio-and-blockio-in-IET-td27080295.html, April 2012.

[22] Wikipedia.org. Ethernet. http://en.wikipedia.org/wiki/Ethernet, April 2012.

[23] Wikipedia.org. Link aggregation. http://en.wikipedia.org/wiki/Link_aggregation, April 2012.

[24] Ceragon.com. Jumbo frames: The microwave perspective. http://www.ceragon.com/files/Ceragon%20-%20Jumbo%20Frames%20-%20Technical%20Brief.pdf, April 2012.

[25] Openstack.org. Open source software for building private and public clouds. http://www.openstack.org/, April 2012.

[26] W. Curtis Preston. *Using Sans and NAS*. O'Reilly & Associates, Inc., Sebastopol, CA, USA, 1st edition, 2002.

[27] Dean Hildebrand, Anna Povzner, Renu Tewari, and Vasily Tarasov. Revisiting the storage stack in virtualized nas environments. In *Proceedings of the 3rd conference on I/O virtualization*, WIOV'11, pages 4–4, Berkeley, CA, USA, 2011. USENIX Association.

[28] Openstack.org. Instance type management. http://docs.openstack.org/diablo/openstack-compute/starter/content/Instance_Type_Management-d1e2734.html, April 2012.

[29] Wikipedia.org. Benchmark (computing). http://en.wikipedia.org/wiki/Benchmark_(computing), April 2012.

[30] Wikipedia.org. Iops. http://en.wikipedia.org/wiki/IOPS, April 2012.

[31] Russel Coker. Bonnie++ documentation. http://www.coker.com.au/bonnie++/readme.html, April 2012.

[32] Bluestop.org. Fio howto. http://www.bluestop.org/fio/HOWTO.txt, April 2012.

[33] Iometer.org. Iometer. http://csis.pace.edu/~lombardi/sciences/computer/systems/windows/docs/iometer.pdf, April 2012.

[34] Rick Jones. Care and feeding of netperf. http://www.netperf.org/svn/netperf2/trunk/doc/netperf.pdf, April 2012.

[35] Iperf. http://iperf.sourceforge.net/, April 2012.

[36] Jeffrey Shafer. I/o virtualization bottlenecks in cloud computing today. In *Proceedings of the 2nd conference on I/O virtualization*, WIOV'10, pages 5–5, Berkeley, CA, USA, 2010. USENIX Association.

[37] C. Baun and M. Kunze. Building a private cloud with eucalyptus. In *E-Science Workshops, 2009 5th IEEE International Conference on*, pages 33 –38, dec. 2009.

[38] HP.com. Hp proliant dl320 g5 server series - overview and features. `http://h10010.www1.hp.com/wwpc/ca/en/sm/WF05a/15351-15351-3328412-241475-241475-3201178.html?dnr=1`, April 2012.

[39] Cisco.com. Cisco 200 series smart switches. `http://www.cisco.com/cisco/web/solutions/small_business/products/routers_switches/200_series_switches/index.html-tab-Models`, April 2012.

[40] Gunnar G. Løvås. *Statistikk for universiteter og høgskoler*. Oslo : Universitetsforl., 2004.

[41] Wikipedia.org. ext3. `http://en.wikipedia.org/wiki/Ext3`, April 2012.

[42] Kevin Lai and Mary Baker. A performance comparison of unix operating systems on the pentium. In *Proceedings of the 1996 annual conference on USENIX Annual Technical Conference*, ATEC '96, pages 22–22, Berkeley, CA, USA, 1996. USENIX Association.

[43] Derek Bruening. Clustered/stacked filled bar graph generator. `http://www.burningcutlery.com/derek/bargraph/`, April 2012.

[44] Atlantic Linux. Linux performance tuning. `http://www.atlanticlinux.ie/training/performance-tuning/slides-with-notes.pdf`, April 2012.

[45] Saumitra Bhanage Vasily Tarasov. Benchmarking file system benchmarking: It *is* rocket science. `http://static.usenix.org/event/hotos11/tech/final_files/Tarasov.pdf`, April 2012.

[46] Eucalyptus.com. Euca2ools user guide. `http://open.eucalyptus.com/wiki/Euca2oolsGuide_v1.3`, April 2012.

[47] Automatic storage tiering limitations. `http://www.theiostorm.com/automatic-storage-tiering-limitations/`, May 2012.

[48] M. Satyanarayanan. A study of file sizes and functional lifetimes. *SIGOPS Oper. Syst. Rev.*, 15(5):96–108, December 1981.

[49] The iscsi enterprise target. `http://iscsitarget.sourceforge.net/`, May 2012.

[50] Open-iscsi. `www.open-iscsi.org`, May 2012.

[51] Ubuntu bonding. https://help.ubuntu.com/community/UbuntuBonding, May 2012.

Vedlegg A

Oversikt over tester

Test av nettverksytelse for den fysiske maskinvaren

Formål

Undersøkelse av hvilken grunnytelse en kan forvente å oppnå på den fysiske maskinvaren som SkyHiGh implementeres på når det kommer til nettverkshastighet.

Potensielle flaskehalser

Ved hjelp av disse dataene ser man om nettverksytelsen for den fysiske maskinvaren når opp til det forventede nivået, slik at dét ikke er en flaskehals.

Maskiner involvert

- Controller
- Compute-noder
- Lagringsnode

Verktøy for ytelsestesting og overvåking

- Netperf

Netperf utfører sending av TCP-pakker over nettverket, og måler hvor fort senderen kan sende, eventuelt hvor fort mottakeren kan motta. Data om belastning av prosessoren for klient og server kommer også frem.

Script involvert

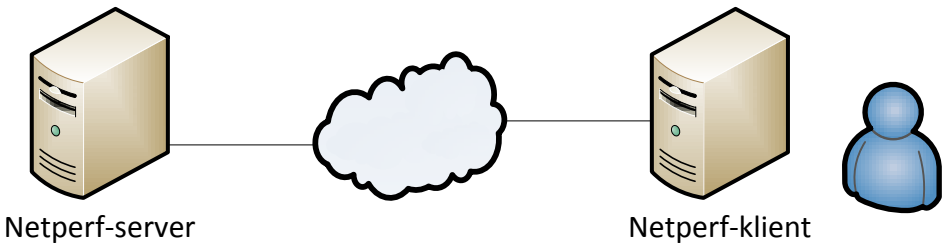
runNetperf.sh

controlNetperf.sh

Antall testgjennomføringer

De ulike testene utføres 100 ganger for å få et signifikant nok datagrunnlag til å kunne vurdere eventuelle årsakssammenhenger og se potensielle trender.

Figur



Figur A.1: Nettverksytelsestesting med Netperf

Test av link aggregation konfigurert på lagringsnode

Formål

Undersøkelse av hvilken ytelse en kan forvente å oppnå på den fysiske maskinvaren som SkyHiGh implementeres på når det kommer til nettverkshastighet når link aggregation er satt opp på lagringsnoden.

Potensielle flaskehalser

Lagringsnoden har to Gigabit nettverkskort, slik at maksimal teoretisk oppnåelig hastighet til og fra noden når disse to er koblet sammen er 2Gbit. Ved å kjøre disse testene oppnås en oversikt over hvorvidt hardware-raidet, CPU eller nettverket vil være potensielle flaskehalser for lagringsnoden når det kommer til den sekvensielle lagringsytelsen. Oppsettet for link aggregation kan også være en flaskehals, ved at man ikke får utnyttet begge kortene fullt når flere enn én maskin kommuniserer med lagringsnoden.

Scenarier

Gruppen vil i denne testen se på om fysiske maskiner klarer å utnytte nettverket når lagringsnoden har link aggregation konfigurert.

Det vil skapes belastning med henholdsvis 2, 3 og 4 maskiner av fysisk art.

Maskiner involvert

- Mellom to og fire fysiske maskiner

Verktøy for ytelsestesting og overvåkning

- Netperf

Netperf utfører sending av TCP-pakker over nettverket, og måler hvor fort senderen kan sende, eventuelt hvor fort mottakeren kan motta. Data om belastning av prosessoren for klient og server kommer også frem. I denne testen må det startes inntil fire instanser av server-programmet til Netperf, hver med ulik port slik at inntil fire maskiner kan koble seg til samtidig:

```
root@lagringsnode:  netserver -p <port>
```

Script involvert

runNetperf.sh

controlNetperf.sh

Antall testgjennomføringer

De ulike testene utføres 100 ganger for å få et signifikant nok datagrunnlag til å kunne vurdere eventuelle årsakssammenhenger og se potensielle trender.

Test av nettverksytelse for virtuelle Linux-maskiner

Formål

Undersøkelse av hvilken grunnytelse en kan forvente å oppnå på virtuelle Linux-maskiner i SkyHiGh-infrastrukturen når det kommer til nettverkshastighet.

Potensielle flaskehalser

Ved hjelp av disse dataene ser man om nettverksytelsen for virtuelle maskiner når opp til det forventede nivået, som er nettverksytelse tilsvarende det en fysisk maskin oppnår. En får i tillegg sett om CPU for den virtuelle maskinen er tilstrekkelig.

Scenarier

I denne testen vil det ses på nettverksytelsen som kan oppnåes mellom virtuelle Linux-maskiner og henholdsvis compute-noden den kjører på og lagringsnoden i infrastrukturen. Det vil testes med tre typer nettverksdrivere: rtl8139, e1000 og virtio.

- Virtuell maskin til og fra compute-node
- Virtuell maskin til og fra lagringsnode

Maskiner involvert

- En virtuell maskin og enten lagringsnoden eller en compute-node

Verktøy for ytelsestesting og overvåking

- Netperf

Netperf utfører sending av TCP-pakker over nettverket, og måler hvor fort senderen kan sende, eventuelt hvor fort mottakeren kan motta. Data om belastning av prosessoren for klient og server kommer også frem.

Script involvert

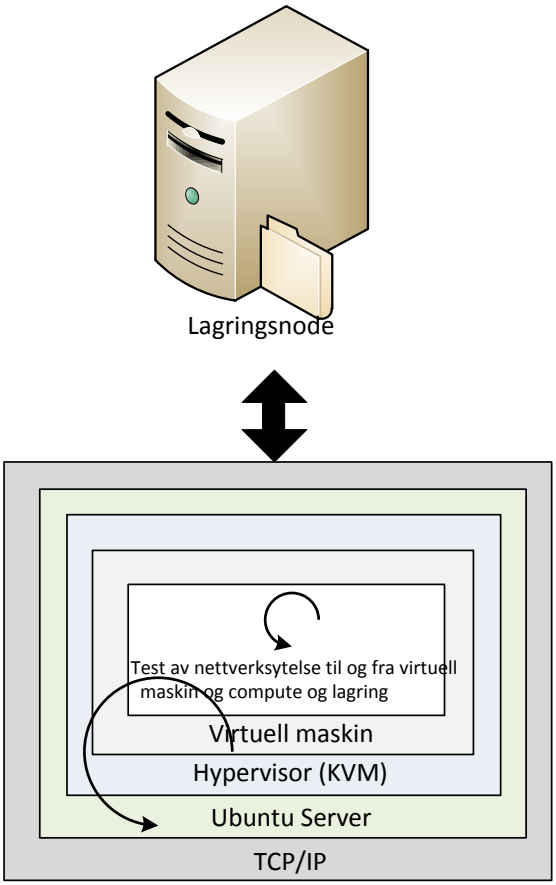
runNetperf.sh

controlNetperf.sh

Antall testgjennomføringer

De ulike testene utføres 100 ganger for å få et signifikant nok datagrunnlag til å kunne vurdere eventuelle årsakssammenhenger og se potensielle trender.

Figur



Figur A.2: Nettverksytelsestesting for virtuelle maskiner

Test av diskytelse for den fysiske maskinvaren

Formål

Undersøkelse av hvilken grunnytelse en kan forvente å oppnå på den fysiske maskinvaren som SkyHiGh implementeres på når det kommer til lokale lagringsenheter.

Potensielle flaskehalser

Ved hjelp av disse dataene ser man om lagringsytelsen for diskene i den fysiske infrastrukturen kan sies å være tilfredsstillende målt opp mot iSCSI-ytelse.

Maskiner involvert

- Controller
- Compute-noder
- Lagringsnode

Verktøy for ytelsestesting og overvåking

- Fio

Fio gir informasjon om ulike målepunkter for ytelse på lagringsenheter. Programmet forteller også om responstider.

Script involvert

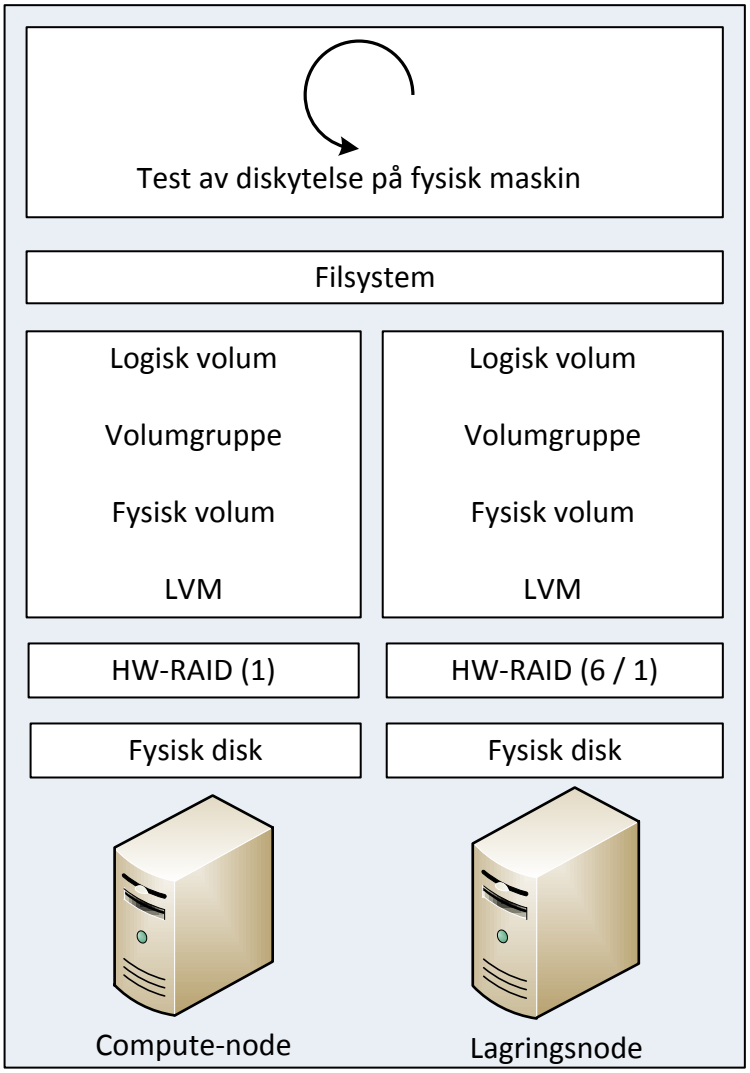
runFio.sh

automatedTest.sh

Antall testgjennomføringer

De ulike testene utføres 100 ganger for å få et signifikant nok datagrunnlag til å kunne vurdere eventuelle årsakssammenhenger og se potensielle trender.

Figur



Figur A.3: Diskytelsestesting for fysiske maskiner

Test av iSCSI-ytelse for fysiske maskiner

Formål

Undersøkelse av hvilken diskytelse som kan oppnås med iSCSI for de fysiske maskinene i fire ulike scenarier.

Potensielle flaskehalser

Testdataene fra disse kjøringene vil kunne fortelle hvorvidt nettverk, iSCSI eller lagringsnode er flaskehalser i infrastrukturen og vil også kunne bidra til å skape en anbefaling for hva som sannsynligvis vil være fornuftig å gå for i forhold til konfigurasjon.

Scenarier

De fire konfigurasjonsalternativene gruppen vil undersøke i denne testen er følgende:

- iSCSI-target fra lagringsnode med fileio-oppsett, med og uten link aggregation på lagringsnoden
- iSCSI-target fra lagringsnode med blockio-oppsett, med og uten link aggregation på lagringsnoden

For hver av disse scenariene vil det kjøres tester hvor henholdsvis 1, 2 og 3 compute-noder samtidig belaster lagringsnoden.

Maskiner involvert

- Compute-nodene
- Lagringsnoden

Verktøy for ytelsestesting og overvåking

- Fio

Fio gir informasjon om ulike målepunkter for ytelse på lagringsenheter. Programmet forteller også om responstider.

Script involvert

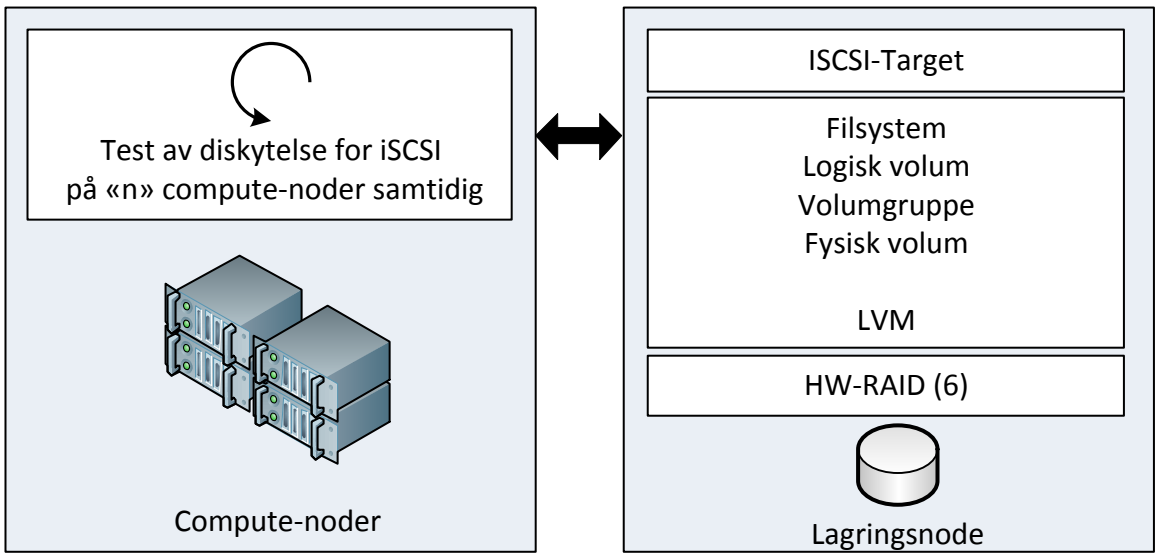
runFio.sh

automatedTest.sh

Antall testgjennomføringer

De ulike testene utføres 100 ganger for å få et signifikant nok datagrunnlag til å kunne vurdere eventuelle årsakssammenhenger og se potensielle trender.

Figur



Figur A.4: Ytelsestesting av iSCSI

Test av diskytelse for virtuelle Linux-maskiner

Formål

Undersøkelse av hvilken ytelse for lagringsenheter som kan forventes i en virtuell maskin når et antall okkuperer nettskyen samtidig og i tillegg utfører mikroytelsestester ved hjelp av eksisterende verktøy.

Potensielle flaskehalser

I testene vil maskinene først kjøre 100 tester med sekvensiell skriving samtidig, deretter 100 tester sekvensiell lesing samtidig, deretter 100 tester tilfeldig skriving samtidig, og til sist 100 tester tilfeldig lesing samtidig. På denne måten observeres hva slags ytelse, for hver av de fire operasjonene, som kan oppnås for hver virtuelle maskin når det er et visst antall virtuelle maskiner som gjør operasjonen samtidig. Flaskehalsene kan potensielt være nettverket, for dårlig maskinvare i compute-nodene eller iSCSI-oppsett. Andre flaskehalser kan også finnes, f.eks dårlig optimalisert iSCSI-versjon e.l.

Scenarier

De tre lagringsløsningene gruppen vil undersøke i denne testen er følgende:

- Compute-noder bruker lokal lagring
- Compute-noder bruker ekstern lagring: iSCSI med fileio-oppsett
- Compute-noder bruker ekstern lagring: iSCSI med blockio-oppsett

For hvert av iSCSI-scenariene vil det startes henholdsvis 1, 2, 3, 6 og 9 virtuelle Linux-maskiner i infrastrukturen. Disse virtuelle Linux-maskinene kjører deretter parallelt sekvensiell skriving, dernest sekvensiell lesing, deretter tilfeldig skriving og til sist tilfeldig lesing. Det samme vil skje for henholdsvis 1, 2 og 3 virtuelle Linux-maskiner når compute-noden bruker lokal lagring som ikke er sentralisert. Link aggregation vil være konfigurert på lagringsnoden.

Maskiner involvert

- Compute-nodene
- Et antall virtuelle Linux-maskiner

Verktøy for ytelsestesting og overvåking

- Fio

Fio gir informasjon om ulike målepunkter for ytelse på lagringsenheter. Programmet forteller også om responstider.

Script involvert

runFio.sh

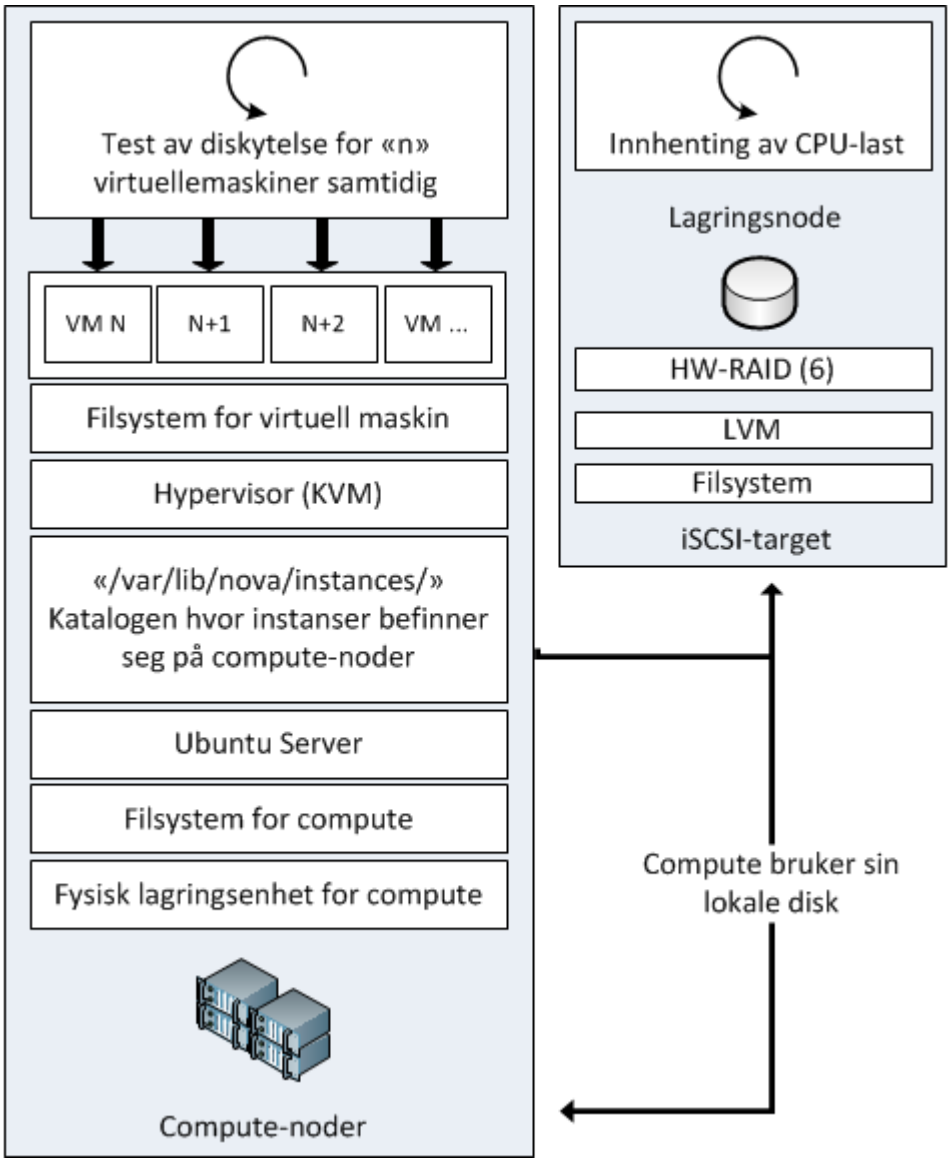
virtualFio.sh

automatedTest.sh

Antall testgjennomføringer

De ulike testene utføres 100 ganger for å få et signifikant nok datagrunnlag til å kunne vurdere eventuelle årsakssammenhenger og se potensielle trender.

Figur



Figur A.5: Diskytelsestesting av virtuelle maskiner

Test av jumbo frames og flere iSCSI-tråder

Formål

Undersøkelse av hvorvidt ytelsesøkning oppnås ved bruk av jumbo frames og/eller øking av antall iSCSI-tråder per tilkobling mellom compute-noder og lagringsnoden.

Potensielle flaskehalser

Jumbo-frames øker potensielt responstiden i nettverket, og kan således bli forringende for kvaliteten som oppleves på de virtuelle maskinene. Med flere iSCSI-tråder vil også mer CPU bli benyttet på lagringsnoden, som kan føre til dårligere tjenesteopplevelse.

Scenarier

- 8- og 16-trådet iSCSI fileio for både fysisk og virtuell maskin med jumbo frames
- 16-trådet iSCSI fileio for både fysisk og virtuell maskin uten jumbo frames
- 8- og 16-trådet iSCSI blockio for både fysisk og virtuell maskin med jumbo frames
- 16-trådet iSCSI blockio for både fysisk og virtuell maskin uten jumbo frames

Scenario én og tre vil bli kjørt i to omganger, med henholdsvis én og to fysiske og virtuelle maskiner. Av tidshensyn har gruppen valgt å kun bruke 9018 bytes jumbo frames.

Maskiner involvert

- 1 compute-node eller 1 virtuell maskin
- Lagringsnoden

Verktøy for ytelsestesting og overvåking

- Fio

- Vmstat

Fio gir informasjon om ulike målepunkter for ytelse på lagringsenheter. Programmet forteller også om responstider.

Script involvert

runFio.sh

virtualFio.sh

automatedTest.sh

Antall testgjennomføringer

Av tidsmessige hensyn reduseres dette antallet: de ulike testene utføres 50 ganger, til forskjell fra 100, for å få et signifikant nok datagrunnlag til å kunne vurdere eventuelle årsakssammenhenger og se potensielle trender og fortsatt spare tid.

Test av mulige hurtiglagringsfordeler for virtuelle Linux-maskiner

Formål

Undersøkelse av hvilke forskjeller som foreligger for cache-fordeler mellom fileio og blockio for iSCSI. For å undersøke dette vil gruppen se på diskytelse når filene som manipuleres (skrives til og leses) har størrelsene 128MB, 256MB eller 512MB. Caching [1] i denne konteksten betyr at man ved lesing eller skriving lagrer unna så mye man kan på internminnet, fordi man forventer å bruke dette igjen senere. Når man leser fra cache går dette mye raskere enn det man kan forvente fra disk alene. Fileio skal kunne vise sin sterke side her, siden dette alternativet lar en bruke internminnet på iSCSI-serveren til caching.

Potensielle flaskehalser

Gruppen tester kun med én instanstype hvor virtuelle maskiner har 512MB ram og én logisk kjerne. Dette er en konsekvens av at compute-nodene kun har 2GB ram tilgjengelig totalt, i tillegg til oppgavemessige tidshensyn. Siden ønsket er å kjøre flere virtuelle maskiner per compute-node, er det maksimalt plass til to eller tre stykker avhengig av minneforbruket på compute-noden.

Scenarier

Sekvensiell skriving og lesing samt tilfeldig lesing og skriving til filer på henholdsvis 128MB, 256MB og 512MB. Dette vil testes for virtuelle maskiner som oppbevarer seg på et iSCSI-target konfigurert med henholdsvis fileio og blockio. Filstørrelsene 128MB og 256MB er valgt fordi de er store nok til å faktisk kunne vise forskjeller mellom fileio og blockio, og 512MB fordi vil vise begrensningen til hurtiglagringen (512MB er mengden minne som instanstypen gruppen bruker har tilgjengelig).

Maskiner involvert

- 1 virtuell maskin

[1] http://en.wikipedia.org/wiki/Cache_(computing)

Verktøy for ytelsestesting og overvåking

- Fio

Fio gir informasjon om ulike målepunkter for ytelse på lagringsenheter. Programmet forteller også om responstider.

Script involvert

runFio.sh

virtualFio.sh

automatedTest.sh

Antall testgjennomføringer

De ulike testene utføres 100 ganger for å få et signifikant nok datagrunnlag til å kunne vurdere eventuelle årsakssammenhenger og se potensielle trender.

Test av utrullingshastighet for virtuelle Linux-maskiner

Formål

Undersøkelse av hvor hurtig det går å starte et antall virtuelle Linux-maskiner i infrastrukturen, avhengig av hva slags lagringsløsning de skal ende opp på. Virtuelle Linux-maskiner kan enten befinne seg fysisk på en lagringsenhet som er lokal for compute-noden de kjører på, eller på en ekstern lagringsenhet som vises for compute-noden ved hjelp av for eksempel NFS eller iSCSI.

Scenarier

De tre lagringsløsningene gruppen vil undersøke i denne testen er følgende:

- Lagringsenhet lokal for compute-noden
- iSCSI-target fra lagringsnode med fileio-oppsett
- iSCSI-target fra lagringsnode med blockio-oppsett

For hver av disse scenariene vil gruppen ta tiden på å rulle ut 12 virtuelle Linux-maskiner i infrastrukturen; 4 på hver compute-node. Den begrensede mengden virtuelle maskiner og variasjoner av mengde er grunnet infrastrukturelle begrensninger, i tillegg til tidshensyn.

Maskiner involvert

- 1 virtuell maskin

Verktøy for ytelsestesting og overvåking

- Egenutviklet script i bash

Scriptet starter et valgt antall maskiner av en spesifisert instanstype med et bestemt operativsystem og henter inn data om statusen til disse inntil alle er operasjonelle.

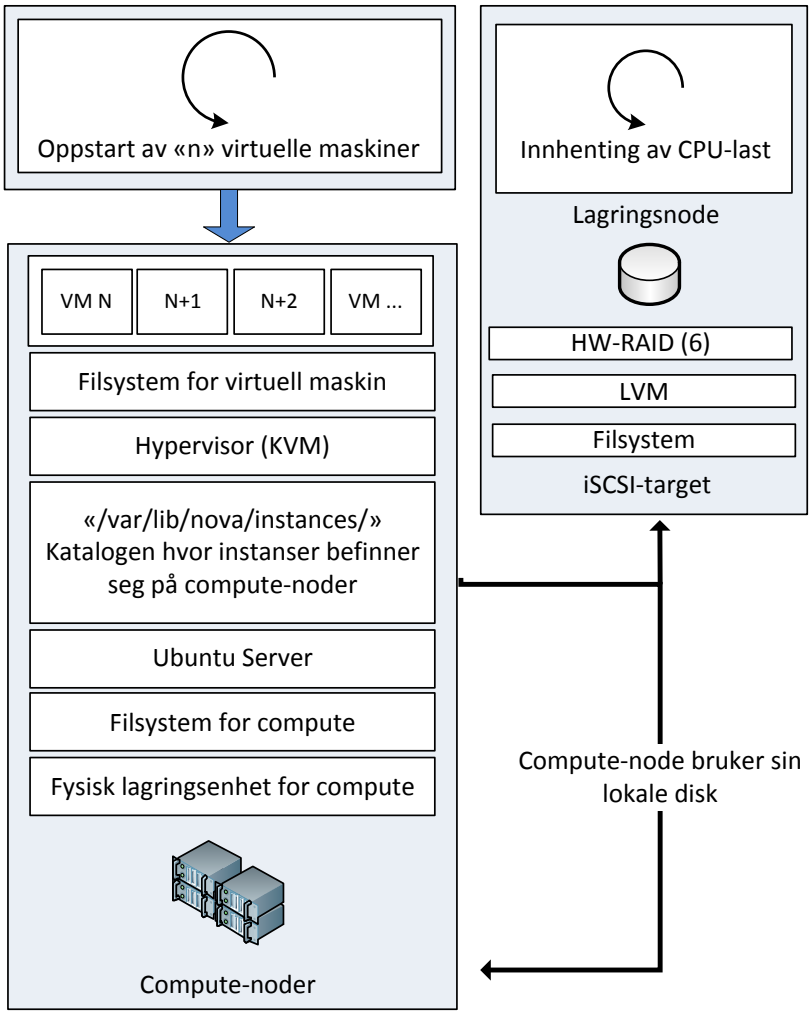
Script involvert

provisionTest.sh

Antall testgjennomføringer

De ulike testene utføres 100 ganger for å få et signifikant nok datagrunnlag til å kunne vurdere eventuelle årsakssammenhenger og se potensielle trender.

Figur



Figur A.6: Utrulling av virtuelle maskiner

Vedlegg B

Oppsett av LVM, iSCSI, Link Aggregation og Jumbo Frames

Installasjon og oppsett av LVM og iSCSI

Det finnes flere ulike implementasjoner av iSCSI-funksjonalitet for Linux-distribusjoner. For å kunne bruke iSCSI, må man ha "target"-funksjonalitet installert på serveren som skal fungerere som iSCSI-server, og en "initiator"-klient på de maskinene som man ønsker å dele ut blokkbasert lagringsplass til.

For lagringsnoden ble det avgjort å installere "iscsitarget" [49]. Arbeidet på den tok utgangspunkt i et annet prosjekt for "target"-implementasjon, Ardis. Dette prosjektet bedømte utviklerene som tilfredsstillende, men mente likevel at det manglet visse grunnelementer (blant annet støtte for Linux-kjerner over versjon 2.6 og 64bit-arkitektur).

Som klientprogramvare gikk gruppen for "open-iscsi" [50]. Dette prosjektet fusjonerte med et annet i 2005, og er i etterkant blitt den desidert mest anerkjente implementasjonen tilgjengelig på Linux-plattformen.

De overnevnte pakkene finnes tilgjengelige i pakkebrønnene til Ubuntu 11.04 som compute-nodene på SkyHiGh sin infrastruktur er bestående av. Lagringsnoden sitt operativsystem er imidlertid Debian "Wheezy", som fremdeles er i en testfase. Versjon 1.4.20.2 av iscsitarget, som ligger i pakkebrønnen til "Wheezy" lot seg ikke installere grunnet noen manglende kodelinjer som var utelatt ved en feil. Gruppen fant, etter litt søking på Internett, en oppdatering som løser dette. Versjonen som ble installert var derfor 1.4.20.2-9. I etterkant ble Ubuntu 11.04 installert på lagringsnoden også, etter inkompabilitetsproblemer mellom Debian og Ubuntu sine OpenStack-versjoner, slik

at dette ikke lenger var et problem. For oppsett av LVM ble den offisielle dokumentasjonen fulgt [14].

Installasjon og oppsett på lagringsnode

Før installasjon av iSCSI ble LVM benyttet til å dele opp raidet (med nivå 6) på 2TB i fire deler. Først ble raidet delt i to partisjoner, begge på 900GB. Deretter ble det laget to fysiske volum av disse partisjonene. Etterpå ble det opprettet logiske volumgrupper av de fysiske volumene. Den ene volumgruppen, kalt "nova-volumes", ble i kraft av navnevalget satt opp til å kunne fungere som "nova-volume"-lagringsplass ved å starte "nova-volume". Den andre logiske volumgruppen på 900GB deles inn i tre helt like logiske volum på omtrentlig 300GB hver, slik at compute-nodene er i stand til å lagre sine kjørende og avslåtte virtuelle maskiner på lagringsnoden (montering av iscsi-target i mappen `/var/lib/nova/instances` hos compute-nodene)

Oppsett av LVM

Først identifiseres hva raidet tilkjennegir seg som ved hjelp av verktøyet "fdisk" [1]:

```
root@kingston:# fdisk -l
```

Dette gav oss følgende informasjon:

```
Disk /dev/cciss/c0d1: 2000.2 GB, 2000205045760 bytes
```

Deretter ble raidet partisjonert i to deler, også ved hjelp av fdisk, med følgende sekvens:

```
root@kingston:# fdisk /dev/cciss/c0d1
Command (m for help): n
Select (default p): <ENTER>
Partition number (1-4, default 1): <ENTER>
First sector: <ENTER>
```

[1] http://en.wikipedia.org/wiki/Fdisk/

```
Last sector: +900G
```

Den samme sekvensen utføres én gang til, med "Last sector" lik "+900G" i stedet for "+900G" hvorpå man trykker t og velger den heksadesimale koden "8e" som svarer til "Linux LVM". Man skriver deretter inn "w" for å lagre endringene utført.

Etterpå kan man lage de fysiske volumene:

```
root@kingston:# pvcreate /dev/cciss/c0d1p1
root@kingston:# pvcreate /dev/cciss/c0d1p2
```

De logiske volumgruppene følger deretter:

```
root@kingston:# vgcreate lvmcompute /dev/cciss/c0d1p1
root@kingston:# vgcreate nova-volumes /dev/cciss/c0d1p2
```

Den logiske volumgruppen "nova-volumes" er deretter klar til bruk for "nova-volume", som lager logiske volum etter behov. Nå kan man lage de logiske volumene som skal fungere som "targets" for compute-nodene på 290GB hver i "lvmcompute"-volumgruppen.

```
root@kingston:# lvcreate -L 290G -n compute1 lvmcompute
root@kingston:# lvcreate -L 290G -n compute2 lvmcompute
root@kingston:# lvcreate -L 290G -n compute3 lvmcompute
```

Disse kommandoene gir deg følgende blokkenheter som nå kan annonseres som targets av iSCSI: */dev/lvmcompute/compute{1,2,3}*

Installasjon og oppsett av iSCSI-target

For installasjon av iSCSI-target ble følgende sekvens fulgt på lagringsnoden:

```
root@kingston:# apt-get install iscsi-target
```

Når denne kommandoen er kjørt, må det sørges for at iSCSI tillates å annonsere sine "targets" ved å endre fra "false" til "true" i linjen med "ISCSI-

TARGET_ENABLE"i filen /etc/default/iscsitarget:

```
root@kingston:# vim /etc/default/iscsitarget
:%s/false/true
```

Når dette er utført, må man spesifisere hvilke lagringsenheter som skal deles ut som targets i filen `/etc/iet/ietd.conf`. Denne filen inneholder alle targets, sammen med eventuelle spesifikke alternativer som skal gjelde for disse. Alternativene man kan velge blant er blant annet antall tråder per tilkoblet maskin (standard er 8), antall kommandoer i kø (standard er 32) og hvorvidt man skal bruke "fileio" eller "blockio".

Gruppens standardkonfigurasjonsfil er som følger:

```
# Target for /var/lib/nova/instances for compute-node #1
Target skyhigh.kingston:compute1.instances
Lun 0 Path=/dev/lvmcompute/compute1,Type=fileio

# Target for /var/lib/nova/instances for compute-node #2
Target skyhigh.kingston:compute2.instances
Lun 0 Path=/dev/lvmcompute/compute2,Type=fileio

# Target for /var/lib/nova/instances for compute-node #3
Target skyhigh.kingston:compute3.instances
Lun 0 Path=/dev/lvmcompute/compute3,Type=fileio
```

Her er det mye slingringsmonn i forhold til å gjøre optimaliserende endringer hva gjelder antall tråder per forbindelse og kommandoer i kø. I etterkant av at alle targets er lagt til med ønskede alternativer, restarter man "iscsi-target" og har en fungerende iSCSI-server som både har "nova-volume"-funksjonalitet og deler ut tre LVM-volum:

```
root@kingston:# /etc/init.d/iscsitarget restart
```

For å endre antall tråder fra standarden som er 8, legger man til en linje etter "Target" hvor man skriver "Wthread <antall tråder>":

```
Target skyhigh.kingston:compute1.instances
Lun 0 Path=/dev/lvmcompute/compute1,Type=fileio
Wthread 16
```

Installasjon og oppsett på compute-nodene

For iSCSI-klienter er det hele mye enklere. Infrastrukturens compute-noder kjører Ubuntu 11.04, som gjør at installasjon skjer med følgende kommando:

```
root@compute:# apt-get install open-iscsi
```

Deretter kan man kontakte lagringsnoden for å se hvilke targets den annonserer og deler ut:

```
root@compute:# iscsiadm -m session -t sendtargets -p <lagringsnode-IP>
<lagringsnode-IP>:3260,1 skyhigh.kingston:compute3.instances
<lagringsnode-IP>:3260,1 skyhigh.kingston:compute2.instances
<lagringsnode-IP>:3260,1 skyhigh.kingston:compute1.instances
```

Lagringsnoden deler ut alle tre targets definert i konfigurasjonsfilen. Fra en compute-node #1, kjøres denne kommandoen for å erverve sitt target:

```
root@compute:# iscsiadm -m node -T skyhigh.kingston:compute1.instances
-p <lagringsnode-IP> -l
```

Etter at denne kommandoen er kjørt, vil det dukke opp en ny blokkenhet som systemet tror er lokalt tilknyttet maskinen. Identifikasjonen til denne finnes ved hjelp av "fdisk -l" eller "dmesg". Blokkenheten kan for eksempel finne på å identifisere seg selv som "/dev/sda". Når den er identifisert, lages et filsystem på blokkenheten. Nyere versjoner av Ubuntu og Debian bruker i dag "ext4" som standard, noe gruppen også vil gjøre. Deretter kan man montere denne disken hvor som helst, for eksempel i `/var/lib/nova/instances`, slik at kjørende virtuelle maskiner lagres på lagringsnoden og ikke lokalt på den enkelte compute-node:

```
root@compute:# mkfs.ext4 /dev/sda
```

```
root@compute:# mount /dev/sda /var/lib/nova/instances
```

Konfigurering av link aggregation (bonding)

Link aggregation (bonding) vil si at man setter av flere nettverkskort til å samarbeide, enten for å gi høyere ytelse, eller for å tilby tjenester som feiltoleranse og lastbalansering. Dette fordrer flere enn ett nettverkskort og like mange ledige porter i switchen som det antall nettverkskort som skal delta, og begrensningen er i utgangspunktet bare antallet nettverkskort du kan sette i maskinen som skal ha link aggregation. Sannsynligvis får man mindre effekt etter et visst antall nettverkskort, eller kanskje en annen flaskehals gjør seg gjeldende (disk, CPU).

Lagringsnoden i OpenStack-arkitekturen kjører på den debianbaserte Linuxdistribusjonen Ubuntu 11.04, som det finnes en åpent tilgjengelig veiledning for hvordan man setter opp bonding på [51]. Det finnes hele seks forskjellige alternativer for hvordan man ønsker link aggregation sin oppførsel å være:

Modus	Beskrivelse
0 (Round-robin lastbalansering)	Round-robin lastbalanseringsmodusen sender pakker sekvensielt fra første til siste fysiske nettverkskort. Tilbyr lastbalansering og feiltoleranse, men kan føre til at pakker kommer frem i feil rekkefølge, slik at TCP/IP sine flytmekanismer gjør at pakker sendes på nytt "unødig".
1 (Active Backup)	Med denne modusen er kun ett fysisk nettverkskort aktivt av gangen. Andre nettverkskort settes kun i drift hvis det aktive på et eller annet vis skulle svikte. Modusen tilbyr kun feiltoleranse.
2 (XOR lastbalansering)	Med modus 2 brukes boolsk XOR til å bestemme hvilket fysisk nettverkskort som skal brukes. Formelen er "(Kilde-MAC XOR-et mot destinasjons-MAC) modulus antall fysiske nettverkskort). Dette vil si at man får både lastbalansering og feiltoleranse, men at samme destinasjon alltid vil bruke samme fysiske sti.
3 (Broadcast)	Denne modusen har feiltoleranse i høysetet, og sender all trafikk ut på alle de fysiske nettverkskortene som er deltagende i den logiske bonding-enheten.
4 (IEEE 802.3ad Dynamic link aggregation)	Link aggregation-grupper opprettes som deler hastighet og duplex-innstillinger. Benytter 802.3ad-spesifikasjonen, og krever en switch som støtter dette.
5 (Balance-TLB)	Tilpasningsdyktig lastbalansering for sending. Trenger ingen spesiell switchstøtte.
6 (Balance-ALB)	Tilpasningsdyktig lastbalansering. Trenger ingen spesiell switchstøtte, og gir lastbalansering og feiltoleranse både for sending og mottak.

Stegene gjennomgått for å sette opp de to nettverkskortene tilgjengelig i lagringsnoden til å representere ett logisk nettverkskort med modus 6 er beskrevet nedenfor. Modus 6 ble valgt fordi det ikke krever spesiell switchstøtte, samtidig som det gir både lastbalansering og feiltoleranse.

Først installeres "ifenslave", et verktøy som tilknytter eller avløser fysiske nettverkskort fra den logiske bonding-enheten:

```
root@kingston:# apt-get install ifenslave
```

Etterpå settes de fysiske nettverkskortene opp til å høre til det logiske nettverkskortet. Filen `/etc/interfaces/networking` redigeres til nedenforstående utgave:

```
auto lo
iface lo inet loopback

auto bond0
iface bond0 inet static
address 192.168.10.185
netmask 255.255.255.0     gateway 192.168.10.1
dns-nameservers 8.8.8.8
post-up ifenslave bond0 eth0 eth1
post-down ifenslave -d bond0 eth0 eth1
bond-mode 6
bond-miimon 100
```

Deretter restartes maskinen og man verifiserer at det logiske nettverkskortet (identifisert som "bond0" er aktivt:

```
root@kingston:# ifconfig bond0

bond0 Link encap:Ethernet HWaddr 00:1e:0b:fd:c8:a6
inet addr:192.168.10.185 Bcast:0.0.0.0 Mask:255.255.255.0
inet6 addr: fe80::21e:bff:fefd:c8a6/64 Scope:Link
UP BROADCAST RUNNING MASTER MULTICAST MTU:1500 Metric:1
RX packets:2812165640 errors:0 dropped:594905 overruns:0 frame:0
TX packets:2147579506 errors:0 dropped:0 overruns:0 carrier:0
collisions:0 txqueuelen:0
RX bytes:7913149363058 (7.9 TB) TX bytes:6620664213234 (6.6 TB)

root@kingston:# cat /proc/net/bonding/bond0
```

```
Ethernet Channel Bonding Driver: v3.7.1 (April 27, 2011)

Bonding Mode: adaptive load balancing

Primary Slave: None

Currently Active Slave: eth1

MII Status: up

MII Polling Interval (ms): 100

Up Delay (ms): 0

Down Delay (ms): 0


Slave Interface: eth0

MII Status: up

Speed: 1000 Mbps

Duplex: full

Link Failure Count: 5

Permanent HW addr: 00:1e:0b:fd:c8:a6

Slave queue ID: 0


Slave Interface: eth1

MII Status: up

Speed: 1000 Mbps

Duplex: full

Link Failure Count: 4

Permanent HW addr: 00:1e:0b:fd:c8:a7

Slave queue ID: 0
```

Konfigurering av Jumbo Frames

Jumbo Frames betyr at en Ethernet-frame sin MTU er større enn standard 1500 bytes, helt opp til maksimalt 9000 bytes. På Linux-baserte operativsystem konfigurerer man det på følgende måte, gitt at man vil ha 9000 bytes på grensesnitt “eth0”:

```
root@compute:# ifconfig eth0 mtu 9000
```

Alle maskinene i infrastrukturen må aktivere Jumbo Frames, i tillegg til switchen, noe som gjøres ved hjelp av webgrensesnittet.

Vedlegg C

Script brukt til testformål

provisionTest.sh

```
1  # This is a script which measures the total amount of time
2  # in seconds it takes to get "n" virtual machines
3  # of flavor "m" with image "o" operational
4
5  # Heavy use of "novaclient" is employed.
6  # Use "nova image-list" and "nova flavor-list" if you are
7  # unsure about the image type and instance type arguments.
8
9  # The format of the output file follow below:
10 # seconds passed;instances building;instances running
11
12 # Example of output file contents - each run ends with a #:
13 # 1;4;0;2;4;0;3;3;1
14
15 #!/bin/bash
16
17 arguments=5
18
19 machines=$1
20 runs=$2
21 image=$3
22 flavor=$4
23 testnr=$5
24
25 user="root"
26 password="XXXX"
27
28 output="$testnr-provisioning"
29
30 iterateRuns=1
```

```
31 sleepAmount=5
32
33 if [[ $# -ne $arguments ]]
34  then
35   echo -e "Usage of $0:\n$0 [number of virtual machines to start]
36             [number of tests] [image type] [flavor] [test number]"
37   exit 0
38  fi
39
40 echo "Starting $machines virtual machines $runs times."
41
42 function boot() {
43  euca-run-instances ami-00000005 --key mykey
44                     -n $machines -t $flavor
45 }
46
47 function delete() {
48  for i in $(nova list | egrep 'ACTIVE|BUILD' | awk {'print $2'})
49   do
50    nova delete $i
51    sleep 1
52   done
53  sleep $sleepAmount;
54 }
55
56 function checkStatus() {
57  unset provisioning
58  totalSeconds=1
59  curACTIVE=0
60  curBUILD=$machines
61
62  while [[ $(mysql -u$user -p$password -Dnova -N -e "select count(*)
63           from instances where vm_state=active;") -ne $machines ]]
64   do
65    update
66   done
67
68  update
69
70  for second in ${provisioning[@]}
71   do
72    echo "$second" >> $output
73   done
74 }
75
76 function update() {
77  checkACTIVE=$(mysql -u$user -p$password -Dnova -N -e "select count
       (*) from instances where vm_state='active';")
78  checkBUILD=$(($machines-$checkACTIVE))
```

```
79
80   if [[ $checkACTIVE -gt $curACTIVE ]]
81    then curACTIVE=$checkACTIVE; fi
82   if [[ $checkBUILD  -lt $curBUILD ]]
83    then curBUILD=$checkBUILD; fi
84
85   tmp="$totalSeconds;$curBUILD;$curACTIVE"
86   provisioning[$totalSeconds]=$tmp
87   sleep 1
88   let totalSeconds=$totalSeconds+1
89  }
90
91  while [[ $iterateRuns -le $runs ]]
92   do
93    clear
94    echo "[$iterateRuns/$runs]"
95    boot &
96    checkStatus
97    delete
98    let iterateRuns=$iterateRuns+1
99   done
100
101 echo "Consult '$output' for results."
```

provisionMachines.sh

```
1  # This script allows you to automatize the launching of a
2  # number of virtual machines and allocate them floating IPs.
3
4  # It is especially useful in situations when you're doing
5  # automated tests and want to do them in quick succession
6  # without having to intervene in between.
7
8  # Example use:
9  # ./provisionMachines.sh 3 192.168.10.129 test ubuntu 0 mykey
10 # The following example will provision 3 virtual machines
11 # with instance type 0 and key 'mykey'. Floating IPs
12 # will be be allocated between 192.168.10.129-131.
13
14 #!/bin/bash
15
16 arguments=6
17
18 launchAmount=$1
19 IP=$2
20 name=$3
21 image=$4
22 flavor=$5
23 key=$6
24
25 instances=1
26
27 # Print this if insufficient arguments are given
28 function usage() {
29  echo -e "Usage of $0:\n$0 [VMs to start]
30           [first floating IP to allocate]
31           [name of batch] [image] [flavor] [key]"
32  exit 0
33 }
34
35 # Function for booting virtual machines
36 function boot() {
37  nova boot --image=$image --flavor=$flavor
38            --key_name=$key "$name-$instances" &> /dev/null
39 }
40
41 # Function to get IDs of virtual machines
42 function getMachines()   {
43  local machines=$(nova list | grep -E "$name.*$1" |
44                   awk {'print $2'})
45  echo $machines
46 }
47
```

```
48 # Function to get number of virtual machines
49 function countMachines() {
50  local count=$(nova list | grep -E "$name.*$1" | wc -l)
51  echo $count
52 }
53
54 # Get the next IP that will be used for floating IPs
55 function nextIP()        {
56  local split=$(echo $IP | tr "." " ")
57  local network=$(echo $split | awk {'print $1 "." $2 "." $3'})
58  local ip=$(echo $split | awk {'print $4'})
59  echo "$network.$(($ip+1))"
60 }
61
62 # Function to allocate a VM a floating IP
63 function allocateIP() {
64  while [[ $(nova list | grep -E "$1.*192" | wc -l) -eq 0 ]]
65  do
66   nova add-floating-ip $(nova list | grep $1 | awk {'print $2'})
67                        $IP &> /dev/null &
68   sleep 3
69  done
70
71  let IPAllocated=$IPAllocated+1
72  echo "Successfully allocated IP '$IP' to instance '$1'."
73 }
74
75 # Function for setting floating IPs
76 function setFloatingIPs() {
77  while [[ $IPAllocated -lt $launchAmount ]]; do
78   for i in $(getMachines ACTIVE); do
79    allocateIP $i; IP=$(nextIP);
80   done
81  done
82 }
83
84 # Function for deploying machines
85 function deployMachines() {
86  j=0;
87
88  while [[ $j -ne $launchAmount && $(countMachines ACTIVE)
89           -ne $launchAmount ]];
90   do
91    boot &
92    let instances=$instances+1;
93    let j=$j+1
94    if [[ $j%10 -eq 0 ]]; then sleep 5; fi
95   done
96
```

```
97    while [[ $(countMachines BUILD) -ge 0 &&
98              $(countMachines ACTIVE) -ne $launchAmount ]];
99     do
100     sleep 1
101   done
102 }
103
104 # Function for starting deployment and allocation of IPs
105 function deployWithFloat() {
106  echo "Attempting to start $launchAmount virtual machines
107        of flavor '$flavor' running image '$image'."
108  echo "Floating IPs will be added from '$IP'"
109
110  deployMachines
111  setFloatingIPs
112
113  echo -e "\nSuccessfully started $launchAmount
114           machines with floating IPs."
115 }
116
117 if [[ $# -ne $arguments ]]; then usage; fi
118
119 deployWithFloat
```

controlNetperf.sh

```
1  # This script controls the script runNetperf.sh
2  # You can use it to transfer to or from a machine running Netserver
3  # using one or more machines running Netperf.
4
5  # Example use:
6  # ./controlNetperf.sh 'manchester;newcastle' 10 100 TX 2_1 kingston
7  # The following example will contact server kingston, open two
8  # sessions of Netserver and have the hosts 'manchester' and
9  # 'newcastle' simultaneously run 100 transfers
10 # each lasting 10 seconds to the server.
11
12 #!/bin/bash
13
14 arguments=7
15
16 computers=$(echo $1 | tr ";" " ")
17 testLength=$2
18 runs=$3
19 mode=$4
20 testNr=$5
21 server=$6
22 key=$7
23
24 path="/root/scripts/netperf/runNetperf.sh"
25 iterate=1
26 portStart=12345
27
28 function usage() {
29  echo -e "Usage of $0:\n$0 [comma-separated list of hosts]
30           [test length] [# number of test runs] [TX/RX]
31           [test number] [server] [key /0]"
32  exit 0
33 }
34
35 if [[ $# -ne $arguments ]];
36  then usage
37 fi
38
39 ssh root@$server 'kill -9 $(pgrep -u root netserver)'
40
41 port=$portStart
42 for i in $computers; do
43  ssh root@$server "netserver -p $port &> /dev/null";
44  sleep 1; let port=$port+1;
45 done
46
47 cpuName="$testNr.$server-$mode.cpu"
```

```
48
49 ssh root@$server "sh -c 'vmstat 1 > '$cpuName' &'" &
50 ssh root@$server "sh -c 'sleep $(($testLength*$runs))
51                  && killall -9 vmstat && scp '$cpuName'
52                  root@192.168.10.190:/tests/'" &
53
54 echo "Waiting for tests to finish"
55
56 d=1
57 for i in $computers
58  do
59   if [[ $key == "0" ]]; then ssh -o UserKnownHostsFile=/dev/null -o
        StrictHostKeyChecking=no root@$i "sh -c 'sleep $d && $path
       $testLength $runs $testNr $server $portStart $mode &> /dev/
       null'" &
60   else ssh -o UserKnownHostsFile=/dev/null -o StrictHostKeyChecking
       =no -i $key ubuntu@$i "sleep $d && /home/ubuntu/scripts/
       netperf/runNetperf.sh $testLength $runs '$testNr-$i' $server
       $portStart $mode &> /dev/null" &
61   let d=$d+1
62   fi
63   let portStart=$portStart+1
64 done
65
66 echo "Sleeping till finished"
67
68 sleep $((($testLength*$runs)+$testLength))
```

runNetperf.sh

```
1  # The following script runs Netperf on an already started
2  # Netserver for the amount of times specified by parameters.
3
4  # Example:
5  # ./runNetperf.sh 10 100 2_1 kingston 12345 TX
6  # The following example will start 100 tests,
7  # each lasting 10 seconds, transferring to 'kingston'
8
9  #!/bin/bash
10
11 arguments=6
12
13 testLength=$1
14 runs=$2
15 testNr=$3
16 remoteHost=$4
17 remotePort=$5
18 mode=$6
19
20 iterate=1
21
22 if [[ $mode == "RX" ]]
23  then output="$testNr-$HOSTNAME-$mode-fra-$remoteHost"
24  else output="$testNr-$HOSTNAME-$mode-til-$remoteHost"
25 fi
26
27 function usage() {
28  echo -e "Usage of $0:\n$0 [each test's length (=> 10sec)]
29           [# number of runs] [test number] [remote host]
30           [port] [TX/RX]"
31  exit 0
32 }
33
34 if [[ $# -ne $arguments ]]
35  then usage
36 fi
37
38 if [[ -e $output ]]
39  then rm $output
40 fi
41
42 if [[ $mode == "TX" ]]
43  then mode="TCP_STREAM"
44  else mode="TCP_MAERTS"
45 fi
46
47 while [ $iterate -le $runs ]
```

```
48  do
49   echo "Test in progress: [$iterate/$runs]"
50   netperf -l $testLength -P 0 -c -C -t $mode -H
51           $remoteHost -p $remotePort >> $output
52   let iterate=$iterate+1
53  done
54
55 echo "Finished with $runs runs. Consult '$output' for result data."
```

automatedTests.sh

```
1  # This script controls the interaction
2  # of tmux for use with automated tests with FIO
3
4  #!/bin/bash
5
6  arguments=5
7
8  testnr=$1
9  command=$2
10 vmstat=$3
11 key=$4
12
13 sleepAmount=5
14
15 storage="192.168.10.185"
16 user="ubuntu"
17
18 machines=$(echo $5 | tr ";" "\n")
19
20 # Print this if insufficient arguments are given
21 function usage() {
22  echo -e "Usage of $0:\n$0 [testnr] [script and arguments]
23           [vmstat 0/1] [key 0/key]
24           [machine n;machine n+1;machine..]"
25 }
26
27 if [[ $arguments -ne $# ]]
28  then usage
29  exit 0
30 fi
31
32 # Create new session for test run
33 tmux new-session -d -s $testnr
34
35 echo -e "\nRunning command '$command' on machines specified."
36
37 # Run the command on each of the specified machines
38 for machine in $machines; do
39  tmux new-window -t $testnr -a -n $machine "ssh -o
       UserKnownHostsFile=/dev/null -o StrictHostKeyChecking=no -i
       $key $user@$machine '$command; exit';exit"
40 done
41
42 tmux kill-window -t $testnr:0
43
44 # If specified, run vmstat on storage node
45 if [[ $vmstat -eq 1 ]]; then
```

```
46   ssh root@$storage "vmstat 1 > '$testnr.cpu' &"
47  fi
48
49  # Sleep until tests are finished
50  while tmux has-session -t $testnr
51   do sleep $sleepAmount
52  done
53
54  echo "$runs finished on remote hosts."
```

virtualFio.sh

```
1  # This script will create the specified number of virtual
2  # machines and start running FIO tests.
3
4  # It is especially useful in situations when you're doing
5  # automated tests and want to do the in quick succession
6  # without having to intervene in between.
7
8  # Example use:
9  # ./virtualFio.s '1 2 3 6 9' 8.1 mykey 129 0
10 # The following example will provision 1, 2, 3, 6 and 9
11 # virtual machines with instance type 0 and key 'mykey'.
12 # Floating IPs will be be allocated between 129-137
13 # before testing with FIO commences.
14
15 #!/bin/bash
16
17 # Print this if insufficient arguments are given
18 function usage() {
19  echo -e "Usage of $0:\n$0 [machines='1 2 4'] [ID of test]
20           [key] [last octet of first floating ip] [flavor]"
21  exit 0
22 }
23
24 arguments=5
25
26 testID=$2
27 key=$3
28 IPStart=$4
29 flavor=$5
30
31 provisionPath="/root/scripts/provisioning/provisionMachines.sh"
32 autoPath="/root/scripts/tmux/automatedTests.sh"
33 fioPath="/home/ubuntu/scripts/fio/"
34 testPath="/home/ubuntu"
35
36 numMachines=$1
37 sleepShort=5
38 network=192.168.10
39 image="newfio"
40 fioJobfile="belastning"
41 memory="4g"
42 blocksize="4k"
43 vmstat=1
44 runs=100
45
46 declare -a runtime=(0 0 100 100)
47 declare -a work=(write read randwrite randread)
```

```
48
49 if [[ $# -ne $arguments ]]
50  then usage
51 fi
52
53 # Function to delete running virtual machines.
54 function delete() {
55  while [[ $(nova list | egrep 'ACTIVE|BUILD|ERROR' | wc -l)
56            -gt 0 ]]
57   do
58    for instance in $(nova list | egrep 'ACTIVE|BUILD|ERROR' |
59                      awk {'print $2'});
60     do nova delete $instance
61      sleep 1
62     done
63   done
64 }
65
66 iteration=0
67
68 for startMachines in $numMachines; do
69  iter=0
70  j=0
71  nextIP=$IPStart
72  while [[ $j -lt $startMachines ]]
73   do
74    IPs[$j]=$network.$nextIP
75    let nextIP=$nextIP+1
76    let j=$j+1
77   done
78
79  # Start the required virtual machines
80  $provisionPath $startMachines $network.$IPStart
81                 testmachines $image $flavor $key
82
83  IP=$(echo ${IPs[@]} | tr " " ";")
84  thisID="$testID.$(($iteration+1))"
85
86  sleep $sleepShort*3
87
88  # Run tests
89   while [[ $iter -lt 4 ]]; do
90  $autoPath "$thisID.${work[$iter]}" "cd $fioPath;./runFio.sh
       $fioJobfile $thisID $memory $blocksize ${runtime[$iter]} ${
       work[$iter]} $testPath $runs" $vmstat /root/$key.priv "$IP"
91   let iter=$iter+1
92  done
93
94  sleep $sleepShort
```

```
95
96   # Delete virtual machines
97   delete
98
99   sleep $sleepShort
100
101  let iteration=$iteration+1
102
103 done
```

runFio.sh

```
1  # This script allows you to automate FIO runs.
2
3  # Example use:
4  # ./runFio.sh belastning test-1.1 4g 4k 0 write /test 100
5  # The following example will sequentially write to a
6  # 4GB file with blocksize 4kb 100 times.
7
8  #!/bin/bash
9
10 arguments=8
11
12 input=$1
13 testname=$2
14 filesize=$3
15 bs=$4
16 runtime=$5
17 work=$6
18 directory=$7
19 runs=$8
20
21 output="$testname-$HOSTNAME.$work"
22 iterate=1
23
24 programPath="/usr/bin/fio"
25 vmstatInterval=5
26
27 # Print this if insufficient arguments are given
28 function usage() {
29  echo -e "Usage of $0:\n$0 [input jobfile] [test name]
30           [file size] [block size] [run time] [work]
31           [directory] [number of runs]"
32  exit 0
33 }
34
35 if [[ $# -ne $arguments ]]
36  then usage
37 fi
38
39 files="$output.raw $output.vmstat $output"
40
41 # Delete files if they exist
42 for file in $files; do
43  if [[ -e $file ]]
44   then rm $file
45  fi
46 done
47
```

```
48 `vmstat $vmstatInterval >> "$output.vmstat" &`
49
50 # Until the number of specified runs are complete, run FIO
51 while [[ $iterate -le $runs ]]
52  do
53   clear
54   echo "Test in progress: [$iterate/$runs]"
55   WORK=$work RUNS=$runtime SIZE=$filesize BS=$bs DIRECTORY=
         $directory $programPath $input --minimal >> "$output.raw"
56   let iterate=$iterate+1
57  done
58
59 if [[ $work == "read" || $work == "randread" ]]; then
60  grep "belastning" "$output.raw" | tr ";" " " |
61        awk {'print $6'} >> $output
62 else
63  grep "belastning" "$output.raw" | tr ";" " " |
64        awk {'print $26'} >> $output
65 fi
66
67 kill -9 $(pidof vmstat)
68
69 echo "Finished with $runs runs. Consult '$output.raw',
70       '$output' and '$output.vmstat' for result data."
```

Vedlegg D

Arbeidsfil for FIO

```
1 [belastning]
2 rw=${WORK}
3 size=${SIZE}
4 directory=${DIRECTORY}
5 bs=${BS}
6 runtime=${RUNS}
7 time_based
```

Vedlegg E

Dagslogg

Uke	Beskrivelse	Dato	Erik timer	Eirik timer
2	Forprosjektrapportarbeid innledes, hjemmeside i stor grad utarbeidet, kilder letes frem fra internett.	09/01	7	7
	Forprosjektarbeid skrider fremover. Lynkurs med Tom. Avgrensning, problemstilling, gannt gjenstår - i de grove trekkene. Leser to rapporter hver til 11.1 for å finne frem til gode punkter å ta i bruk senere. Møte med Erik (oppdragsgiver fredag 13/1).	10/01	4	5
	Arbeid på forprosjekt. Lesing av tidligere rapporter. Innhenting og sortering av kilder.	11/01	5	5
	Finne frem til spørsmål til oppdragsgiver ift avgrensning, problemstilling, tips, råd, osv.	12/01	4	3
	Møte med oppdragsgiver. Arbeid på forprosjekt. Søk etter flere kilder og forskningsarbeid.	13/01	7	7
3	Kilder for iSCSI. Møte med oppdragsgiver igjen for å avklare oppgavens innhold.	16/01	5	5
	Kilder. Forprosjekt. Gantt.	17/01	5	5
	Sortert kildematerialet, laget gantt-diagram.	18/01	6	5
	Ferdigstilt forprosjekt. Møte med veileder og oppdragsgiver. Skrevet teori om iSCSI.	19/01	7	7
	Undersøkelse av eksisterende benchmarkingverktøy og studier rundt dette. Søk etter relevant litteratur.	20/01	5	5
	Lest relevant litteratur.	22/01	2	2

Uke	Beskrivelse	Dato	Erik timer	Eirik timer
4	Diskutert struktur for sluttrapport. Tegnet. Sett mer på benchmarking.	23/01	6	6
	Diskutert struktur, tegneprogram og logo. Lest relevante arbeid.	24/01	8	8
	Tegnet infrastruktur. Lest relevante arbeid. Skrevet om litt på forprosjekt.	25/01	8	7
	Møte med oppdragsgiver. Kurs i akademisk skriving. Laget diagrammer.	26/01	7	7
	Levert forprosjekt og prosjektavtale. Hentet utstyr. Skrevet om nettskyer.	27/01	3	3
5	Skrevet om virtualisering. Lest studier.	30/01	5	5
	Lest studier og om benchmarking.	31/01	6	7
	Skrevet om link aggregation.	01/02	6	6
	Møte med oppdragsgiver. Lest om Openstack.	02/02	4	3
	Skrevet om ytelsesmålingsverktøy. Lest relaterte oppgaver og artikler.	03/02	4	
	Lest artikler. Sortert kilder. Skrevet teori.	04/02	5	
6	Skrevet om benchmarking-verktøy og generelt om benchmarking.	06/02	6	6
	Lest og skrevet om OpenStack.	07/02	5	5
	Litteraturstudie og teoriskriving.	08/02	8	6
	Litteraturstudie og teoriskriving. Møte med oppdragsgiver og veileder. Bedre avklaring av oppgavens innhold. Initiell testing.	09/02	9	7
	Lest om og utforsket gnuplot.	10/02	6	4
7	Skrevet script for bonnie++ og jobbet på sluttrapport.	13/02	9	8
	Skrevet script for bonnie++ og jobbet på sluttrapport.	14/02	6	6
	Skrevet script for bonnie++ og jobbet på sluttrapport.	15/02	7	3
	Skrevet innledning for rapporten.	16/02	5	
	Lest om og eksperimentert med gnuplot og bonnie++.	17/02	5	
	Teori - lesing og skriving.	18/02	6	
	Teori - lesing og skriving.	19/02	10	

Uke	Beskrivelse	Dato	Erik timer	Eirik timer
8	Teori om testing av nettverksytelse.	20/02	4	7
	Script for netperf. Teori.	21/02	6	6
	Script for netperf. Teori.	22/02	9	6
	Script for formatering av resultater fra bonnie og netperf. Teori.	23/02	9	
	Script for formatering av resultater fra bonnie og netperf. Teori.	24/02	3	
9	Teori om I/O.	27/02	6	6
	Teori om virtualisering. Script for netperf og bonnie.	28/02	5	5
	Innhenting av baseline-data for fysiske maskiner. Utrolig dårlig ytelse for compute-nodene med hardware-raid.	29/02	4	3
	Oppsett av lagringsnode, iSCSI og nova-volume. Møte med oppdragsgiver. OpenStack ikke operativ. Diskutert link aggregation.	01/03	4	6
	Skriving av testscenarioer, teori. Oppsett av iSCSI. Sortert testene som skal gjøres for organisering. Laget maler for visualisering av resultater.	02/03	3	3
	Ferdigstilt tester. Jobbet på script for prosessering. Tester for virtuelle maskiner begynner så snart OpenStack er operasjonelt.	04/03	4	
10	Laget grafer og tabeller for hvordan testdata vil bli visualisert.	05/03	5	6
	Jobbet med script. venter på at openstack skal bli operasjonelt. Rapportskrivingsmøte.	07/03	6	6
	Møte med oppdragsgiver. Reinstallert lagringsnode. Satt opp ISCSI. Kjørt ISCSI-tester med fileio.	08/03	10	6
	Fortsatt med fileio-tester for fysiske maskiner. Eirik syk.	09/03	5	
	Kjørt iscsi-blockio-tester for fysiske maskiner.	10/03	3	
	Funnet ut at bonnie++ ikke gir fornuftige resultater når man kjører det på flere maskiner mot ekstern lagring samtidig.	11/03	10	

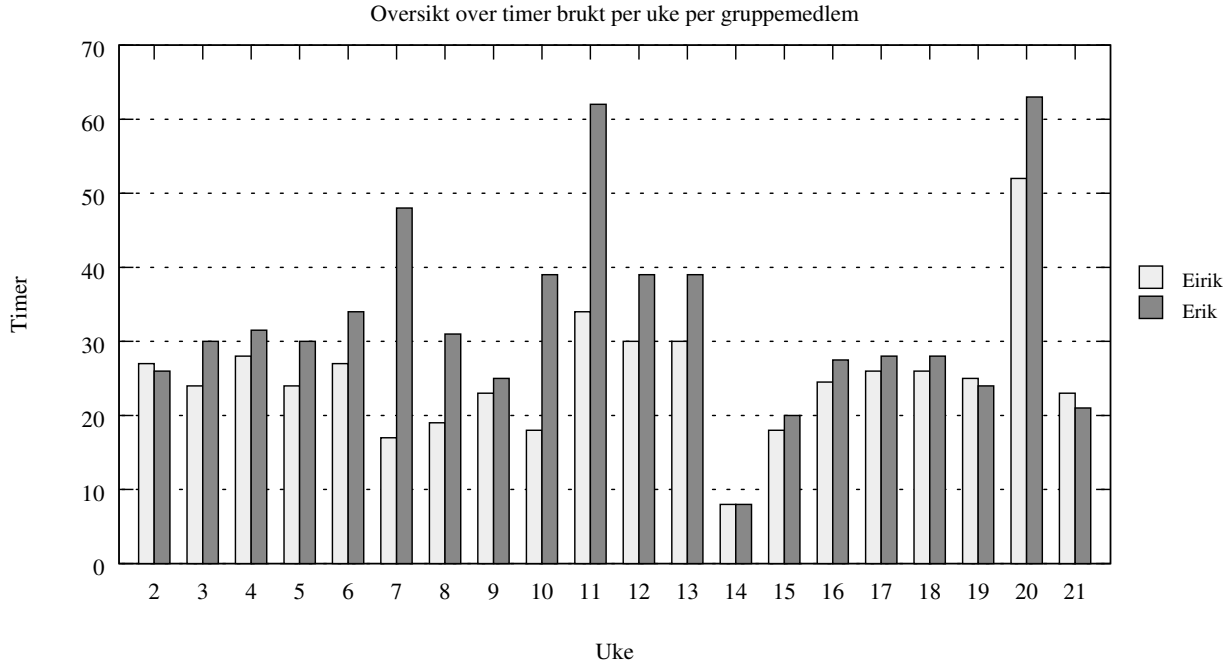
Uke	Beskrivelse	Dato	Erik timer	Eirik timer
11	Script for formatering/analyse av DD-data. Flere iscsi-basetester. Forsøker å sette opp link aggregation. Kjører tester med link aggregation mot iscsi. Knotet mye med nova-volume, som nekter å fungere som lokal og ekstern disk for virtuelle maskiner. Skrevet opp oversikt over testscenarier og laget tilhørende forklarende figurer.	12/03	11	3
	Skrevet script for automatisering av fio- og seekmark-tester, utrulling av maskiner og netperf. Utrullingen av maskiner fortsetter til den har nådd målet, ettersom det er helt tilfeldig om man får startet og allokert IPer eller ei.	13/03	13	11
	Skrevet om ulemper ved virtualisering og I/O. Kjørt tester for iscsi. Funnet ut av hvorfor det er treig nettverksytelse på VMer - nettverksdrivervalg!	15/03	8	5
	Sortering av kilder og testing. Automatisering av testing av diskytelse for virtuelle maskiner.	16/03	7	2
	Satt i gang siste rest av iSCSI-tester. Satt opp lagringsnode for backup av nåværende OpenStack-konfigurasjon.	17/03	5	1
12	OpenStack settes opp på nytt og blir forhåpentligvis mer stabilt. iSCSI på windows.	19/03		6
	Iometer-testkonfigurasjonsfil og output-scripting. Laget grafer og tabeller for resultatene som eksisterer	20/03	7	7
	Ferdigstilt grafer og tabeller for foreløpig utførte tester. OpenStack ser ut til å nesten være operasjonelt. IO-teori.	21/03	12	7
	Skrevet det vi har av sluttrapport i RST. Møte med oppdragsgiver og gjennomgang av foreløpige resultater.	22/03	8	7
	Testing av nettverk hos virtuelle maskiner.	23/03	6	2
	Sluttrapport og tester.	25/03	6	1

Uke	Beskrivelse	Dato	Erik timer	Eirik timer
13	Startet tester av VMer. Forsøkt å få windows-image opp og gå.	26/03	8	6
	Kjørt tester, skrevet på sluttrapport.	27/03	8	9
	Fortsatt test av lokale disker for virtuelle maskiner. Skrevet på sluttrapport.	28/03	6	6
	Skrevet på sluttrapport. Kjørt og scriptet tidstester på starting av instanser.	29/03	5	6
	Startet tester som skal kjøre i påsken. Skrevet på sluttrapport. Startet tester som skal kjøre i påsken.	30/03	4	4
	Sluttrapport, lesing og formatering av tester.	31/03	8	
14	Sett på noen av testresultatene som kjører. Skrevet litt på sluttrapport.	02/04	2	2
	Sørget for at tester enda kjører. Skrevet på sluttrapport. Ferie!	03/04	2	2
15	Skrevet på sluttrapport. Liten snakk med oppdragsgiver om iscsi og jumbo frames.	10/04	7	7
	Tester med flere iSCSI-tråder og jumbo frames kjører. Ført opp responstider i tabeller. Sluttrapport.	11/04	7	7
	Sluttrapport.	12/04	3	3
	Kjørt jumbo frames-tester.	15/04	3	1
16	Kjørt tester, sluttrapport. Forsøk på å få opp Windows image.	16/04	6.5	6.5
	Sluttrapport. Forskjellig antall iSCSI-tråder fra VM (1, 8, 16,32 64) Liten/ingen forskjell.	17/04	8	8
	Testing av cache-ytelse (fileio med små filer), gode resultater.	18/04	6	6
	Cache-tester. Møte. Sluttrapport.	19/04	4	4
	Kjørt tester med mindre filer.	20/04	3	
17	Samlet resultater i grafer og tabeller. Sluttrapport.	23/04	4	4
	Overført innledning og teori til latex. Laget litteraturliste.	24/04	8	8
	Skrevet på sluttrapport. Møte. Kjørt tester med 4 VMer på 1 compute.	26/04	8	8
	Sluttrapport og litteraturliste.	27/04	8	6

Uke	Beskrivelse	Dato	Erik timer	Eirik timer
18	Sluttrapport og tolkning av resultater	30/04	5	5
	Kjørt noen manglende tester	01/05	3	
	Diskusjon av resultatene. Tester.	02/05	2	3
	Endret sluttrapport per oppdragsgivers ønsker og tilbakemelding. Møte med veileder og oppdragsgiver.	03/05	8	8
	Sluttrapport. Utarbeidet grafer/tabeller og kikket på resultater.	04/05	8	8
19	Sluttrapport	07-13/05	24	21
20	Sluttrapport	14-20/05	52	63
21	Sluttrapport	21-23/05	21	23

Vedlegg F

Uke- og timeoversikt



Figur F.1: Oversikt over timer arbeidet per gruppemedlem per uke

	Totalt	Gjennomsnitt
Erik	656	34.52
Eirik	501.5	26.39

Tabell F.1: Oversikt over timer arbeidet per gruppemedlem totalt

Vedlegg G

Møtelogg

Møte med veileder, 05.01

Deltakende: Hanno Langweg, Erik Raassum
Fraværende: Eirik Bakken

Punkter som ble diskutert:

- Best practices og ytelsesvurderingene bør gjelde for en størrelsesorden som er relevant i undervisningssammenheng på HiG (dvs ca. 50-100 virtuelle maskiner).
- Det bør tidlig letes etter relevante kilder, herunder artikler, konferansemateriell, tidsskrift etc, som kan benyttes i oppgaven. Let på områder som www.acm.org, www.usenix.org og www.ieee.org.
- Ikke bruk for mye tid på forprosjektrapporten - denne skal kun leveres inn og godkjennes.
- Det bør tidlig komme frem hvilke avgrensninger, kostnads- og ytelsesmessige behov som finnes, og hvilke behov de ulike emnene har i forhold til ytelse på virtuelle maskiner.
- OpenStack - interne maskiner for skyen eller innleid utenfra (eksternt). Er det andre hensyn ift ytelse, kostnader o.l dersom man har en ekstern sky kontra en intern? Kan de kombineres? Hvilke ulike typer anvendelser passer de til?

Møte med oppdragsgiver, 13.01

Deltakende: Erik Raassum, Eirik Bakken, Erik Hjelmås

Agenda fra gruppedeltakerenes side:

- Konkretisering og avgrensning av oppgaven (hva skal vi gjøre, hva skal vi ikke gjøre - lettere å øke omfang på senere tidspunkt enn å starte bredere enn nødvendig). Hvilke resultatmål er mest vesentlige?
- Hvor ofte skal møter avholdes?
- Er oppgaven av mer vitenskapelig, teoretisk art enn praktisk?
- I/O-begrensninger for skyer generelt, eller OpenStack-spesifikt?
- Noen bacheloroppgaver vi burde lese?
- Fallgruver?
- Tips til kilder, referanser, tidsskrift, bøker – hva bør vi lese på?
- Tips til ytelsestestingsapplikasjoner, hvordan dette bør foregå?

Punkter som ble diskutert:

- Verktøy for IO-ytelse bør letes frem.
- Testene bør være lange nok til å teste faktisk ytelse, i tillegg til at det bør kjøres mange nok til at det blir
- statistisk signifikante målinger å hente.
- ISCSI ses på som et spesielt viktig punkt for oppgaven.
- Let etter relevante kilder og begynn tidlig med å skrive på sluttrapporten.

Møte med oppdragsgiver, 16.01

Deltakende: Erik Raassum, Eirik Bakken, Erik Hjelmås

Punkter som ble diskutert:

- Identifiser når og hvor flaskehalser kan oppstå. Oppgaven avgrenses til nettverksoppsett og lagringsløsning (klientaksess droppes).
- Møter avholdes hver torsdag, med både veileder og oppdragsgiver til stede.
- På møte med Hanno hver torsdag deltar også ErikH.
- Oppgaven i et nøtteskall: avdekke flaskehalser/ytelsesproblemer på infrastrukturen som settes i drift.
- Ha dette i bakhodet: oppgaven kan være nyttig for OpenStack!

Møte med oppdragsgiver og veileder, 19.01

Deltakende: Erik Raassum, Eirik Bakken, Erik Hjelmås, Hanno Langweg

Agenda fra gruppedeltakerenes side:

- Baseline. Teste ytelse internt på disker og nettverk på både compute og storage-noder (gjerne flere, for å sjekke evt variabilitet) før vi gjør noen endringer. Deretter treffe optimaliserende tiltak for å sjekke hva eller hvorfor det ble bedre eller dårligere / diskutere dette; så nye forsøk, osv.
- Tips til ytelsestestingsapplikasjoner, hvordan dette bør foregå, osv?
- Noen bacheloroppgaver vi burde lese?
- Fallgruver?
- Avgrensning og konkretisering.
- Utarbeide testscenarier?

Punkter som ble diskutert:

- Hvor trengs mer hardware?
- Hvor vil flaskehalser oppstå (CPU, RAM, lagring). Hvilke emner vil det være relevante for?
- Bli enige med den andre gruppen om en felles infrastruktur.
- Tegne scenarier.

- Ha like instanser som i public clouds – lag private instanser med samme CPU+RAM+disk, hvilken er best?
- Virtuelle desktops. Mange emner ønsker ferdig oppsatte virtuelle desktops.
- Trenger instanser som er sammenlignbare med Amazon-instanser.
- Påvirkning av virtualiseringslaget – hypervisor vs virtuell maskin.
- Målinger, vil de kjøre i virtuelle omgivelser? Kan de bli automatisert sammen med starting av instanser?
- Nettverk- og harddiskmålinger (I/O) mellom virtualisert-virtualisert, intern-ekstern.
- Virtual desktop measurements. Can it replace a desktop?
- Virtuelle desktop-målinger. Kan det erstatte fysiske desktops?
- Type RAID?
- Ett iscsi target pr virtuell pc? Antall threads - etc.
- Sammenligne mellom en compute node (ikke optimalisert) og en lagringsnode med optimalisering.

Møte med oppdragsgiver og veileder, 26.01

Deltakende: Erik Raassum, Eirik Bakken, Erik Hjelmås, Hanno Langweg

Agenda fra gruppedeltakerenes side:

- Begynt på selve oppgaven, ettersom forprosjektet leveres inn i løpet av morgendagen.
- Funnet relevante kilder.

Punkter som ble diskutert:

- Baseline. Teste ytelse internt på disker og nettverk på både compute og storage-noder (gjerne flere, for å sjekke evt variabilitet) før eventuelt gjøre noen endringer. Deretter treffe optimaliserende tiltak for å sjekke hva eller hvorfor det ble bedre eller dårligere / diskutere dette. Så nye forsøk, osv.

- Scenarier
- Er det vi måler nødvendig for å svare på spørsmålene vi har stilt?

Det som blir viktig fremover er at den andre gruppen får opp infrastrukturen (nova-compute, nova-volume) slik at vi begynne å kikke på faktisk ytelsestesting.

Møte med oppdragsgiver, 02.02

Deltakende: Erik Raassum, Eirik Bakken, Erik Hjelmås **Fraværende**: Hanno Langweg

Punkter som ble diskutert:

- Driftsspørsmål: Utfordringer: F.eks at det kan bli stor belastning når alle jobber samtidig.
- Start så enkelt som mulig med VMer.
- Start et antall vmer, start og teste - finn en baseline.
- Hvordan samle inn målinger, presentere de, skape belastning.
- Når man tegner og plotter grafer. Gjør enklest mulig. Alle målinger bør gjentas 50 ganger. Plott middelverdien og standardavvik.
- Fil som lagring, iSCSI som lagring, lokal disk som lagring. Hvordan bør man koble lagringa?

Møte med oppdragsgiver og veileder, 09.02

Deltakende: Erik Raassum, Eirik Bakken, Erik Hjelmås, Hanno Langweg

Agenda fra gruppedeltakerenes side:

- Hvor mange omganger per test (50?). Hvordan visualisere det?
- Hvordan skape belastning à la emnene (WWW-tek, ethh, ops) - om vi i det hele tatt skal dette?
- I så fall, bruke den tyngste oppgaven i hvert emne?
- Hvilke basistester skal kjøres?

- Hypervisor uten VMer (diskytelse, nettverksytelse, ISCSI-ytelse)
- Hva slags ytelse skal vi fokusere på, i.e hvilke verdier skal vi hente inn? Det har jo litt å si ift hvilket benchmarking-program vi går for (bonnie++ - den måler sekvensiell input/output og random aksess og CPU-bruk for disse).
- Sekvensiell skriving og lesing?
- Responstider?
- Throughput nettverk?
- Hvor fort instancer starter når mange starter samtidig / flere kjører allerede?

Punkter som ble diskutert:

- Filebench: mulig program for ytelsestester.
- Antall tester avhenger av presisjonsgraden man ønsker - tommelfinger-regel er 30.
- Standardavvik og stolpediagram.
- Ryddig statistisk analyse.
- ISCSI-ytelse som funksjon per tråd. ene aksen med antall tråder du bruker
- Statistisk signifikans - ift hvor mange tester..
- Finne ut om man taper mye med rdp på en ekstern nettsky?
- Tre nivåer finnes:
- Direkte på server, hypervisor - utnyttelsen av det fysiske laget
- Inne i en vm - tjenestekvalitieten, FÅR jeg faktisk tilsvarende en fysisk maskin 4GHz, 512MB ram, eller er det stor variasjon mellom fysisk og virtuelt.
- På en desktop med RDP til en VM - sluttbrukeren.
- Relevant for akseptanse, får brukeren ikke tilstrekkelig ytelse vil den ikke ha behov for løsningen og foretrekker heller en fysisk maskin.

Møte med oppdragsgiver, 16.02

Deltakende: Erik Raassum, Eirik Bakken, Erik Hjelmås **Fraværende**: Hanno Langweg

Dette ble et kort møte på grupperommet som følge av fravær fra Hannos side. Ytterligere avgrensning: klarer vi finne noen flaskehalser, hvor virtuell ytelse er det svake punkt?

Oppdragsgiver ytret i tillegg ønske om å se på om optimaliseringer for filsystem blir borte når man har VM som fil, om mulig. Han ønsker å teste med flere mulige måter å lagre de virtuelle maskinene på, dersom mulig: lokal disk (som fil), lokal disk (LVM), på iSCSI-target (SAN), på EBS (om mulig).

Møte med oppdragsgiver og veileder, 01.03

Deltakende: Erik Raassum, Eirik Bakken, Erik Hjelmås, Hanno Langweg

Agenda fra gruppedeltakerenes side:

- Er sekvensiell skriving/lesing viktig?
- Forkaste Bonnie++?
- Første flaskehals: 1Gbit på lagringsnode.

Punkter som ble diskutert:

- Er det noen tester vi vil se mer på (nøyaktige data: 1000 tester?) - begrunn hvorfor vi kjører testene så mange ganger som vi gjør.
- Forsøk å sette opp link aggregation.
- Computenoder - flaskehals?
- Når man sammenligner: benytt stolpediagram med standardavvik.

Møte med oppdragsgiver og veileder, 08.03

Deltakende: Erik Raassum, Eirik Bakken, Erik Hjelmås, Hanno Langweg

Punkter som ble diskutert:

- Spesifiser alle tre lagene - fysisk, hypervisor og sluttbruker!
- Aksesstid for minne i VM? Ta med informasjon om de tre store ressursene: CPU, Minne, I/O?
- Nei: minne og CPU bør tas i egen oppgave -> bevisst valg med I/O (disk, nettverk)!
- Script for å forsøke å starte VMer utarbeidet.

Møte med veileder, 14.03

Deltakende: Erik Raassum, Eirik Bakken, Erik Hjelmås, Hanno Langweg

Punkter som ble diskutert:

- Begrunne hvorfor vi tester hva vi gjør!
- Trekk frem det spennende på presentasjonen.

Møte med veileder, 12.04

Deltakende: Erik Raassum, Eirik Bakken, Hanno Langweg **Fraværende**: Erik Hjelmås

Punkter som ble diskutert:

- Dokumentér og kjør tester der man tydelig ser caching-effekten (mindre filer - 256MB, 512MB osv)
- Prøv med Windows om mulig.
- Se på 3D-visualisering av resultatene.
- Bedre å øke antall noder eller minnekapasitet?

Møte med oppdragsgiver og veileder, 19.04

Deltakende: Erik Raassum, Eirik Bakken, Erik Hjelmås, Hanno Langweg

Dette ble et kort (15 minutters) møte grunnet få anliggende å gjennomgå.

Punkter som ble diskutert:

- Forsøk å presenter forholdene mellom hastighet, fileio/blockio, filstørrelse osv.
- Viktig å dokumentere valget av å kjøre færre tester på slutten på grunn av liten spredning i dataene (selv om dette er noe sent, da vi allerede har kjørt de fleste testene).

Møte med oppdragsgiver og veileder, 26.04

Deltakende: Erik Raassum, Eirik Bakken, Erik Hjelmås, Hanno Langweg

Punkter som ble diskutert:

- Forrige uke har gått med på å implementere veileders forslag til forbedringer på sluttrapporten.
- Oppdragsgiver leser utkast i løpet av neste uke.
- Hvor mange compute-noder og virtuelle maskiner kan vi kjøre opp før vi trenger en ny lagringsnode?
- Diskuter resultater både i diskusjons- og resultatdelen.
- Hvorfor øker responstiden så kraftig når vi går fra 6 til 9 virtuelle maskiner? Undersøk og resonner!

Møte med oppdragsgiver og veileder, 03.05

Deltakende: Erik Raassum, Eirik Bakken, Erik Hjelmås, Hanno Langweg

Punkter som ble diskutert:

- RAID10 er i realiteten RAID1.
- I etterpåklokskapens navn burde gruppen nok ha tatt statistikk dette semesteret (selv om vi da hadde måtte ta 35 studiepoeng totalt).
- Burde vi bruke standardfeil og konfidensintervall for å bekrefte eventuelle hypoteser, eller skal vi bare beskrive forholdene som er fremkommet med standardavvik/gjennomsnittsverdi?
- Utbedringstilbakemeldinger fra veileder og oppdragsgiver for sluttrapport er nå utarbeidet.

Vedlegg H

Kontrakt

Vedlegg H inneholder kontrakten mellom oppdragsgiver og gruppen.

HØGSKOLEN I GJØVIK

PROSJEKTAVTALE

mellom Høgskolen i Gjøvik (HiG) (utdanningsinstitusjon),

_____

_____ (oppdragsgiver), og

Eivik Raassum (090373)

Eivik Bakken (090382)

_____ (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 10/1-12 til 7/6-12.

   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): Hanno Langweg

Oppdragsgivers kontaktperson (navn): Eirik Hjelmås

Student(er) (signatur): Eirik Raasscum dato 27/1-12

Eirik B[illegible] dato 27/1-12

_____ dato _____

_____ dato _____

Oppdragsgiver (signatur): [signature] dato 26/1-12

IMT Dekan/prodekan (signatur): [signature] dato 15/2-2012

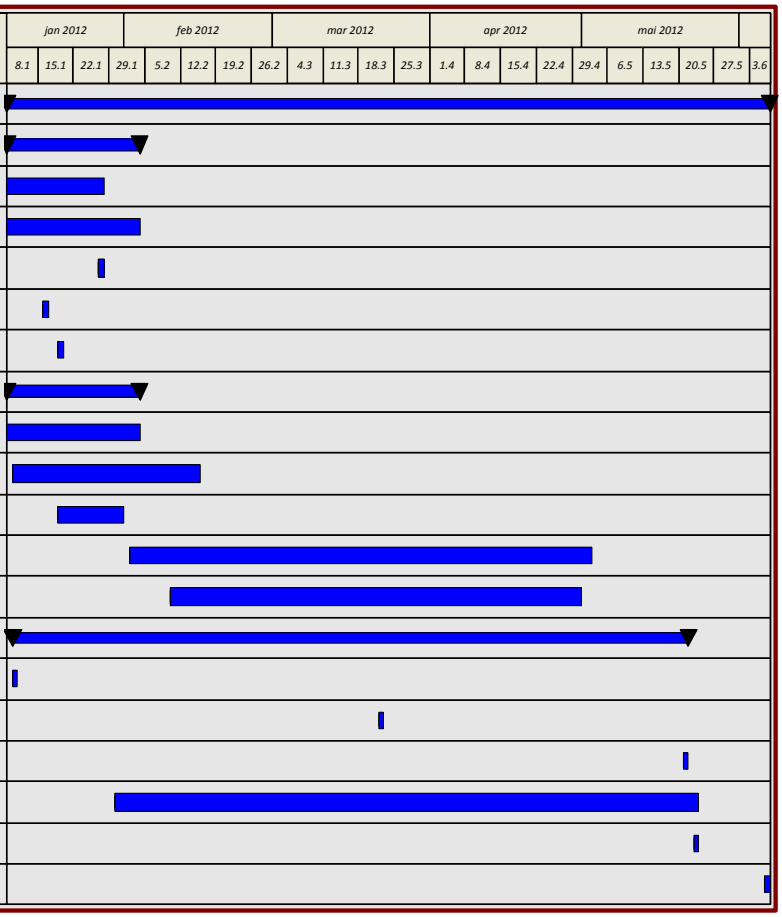
Vedlegg I

Gantt-diagram

Vedlegg I inneholder gantt-diagrammet fra forprosjektplanens innlevering i januar, samt et gantt-diagram som er sist oppdatert i mai med de faktiske gjøremål gjennom hele oppgaveløpets gang. Den overordnede strukturen og oppsettet har holdt seg tilnærmet likt mellom begge diagrammene, men to store forskjeller observeres for undersøkelse av eksisterende ytelsesprogramvare og utarbeiding av testscenarier. Disse delene av oppgaven viste seg å være mer kompliserte enn forventet, og derfor ble det brukt lenger tid på å finne hensiktsmessige verktøy og fremgangsmåter enn det som først var forespeilet og lagt til grunn.

Gantt-diagram fra forprosjektet

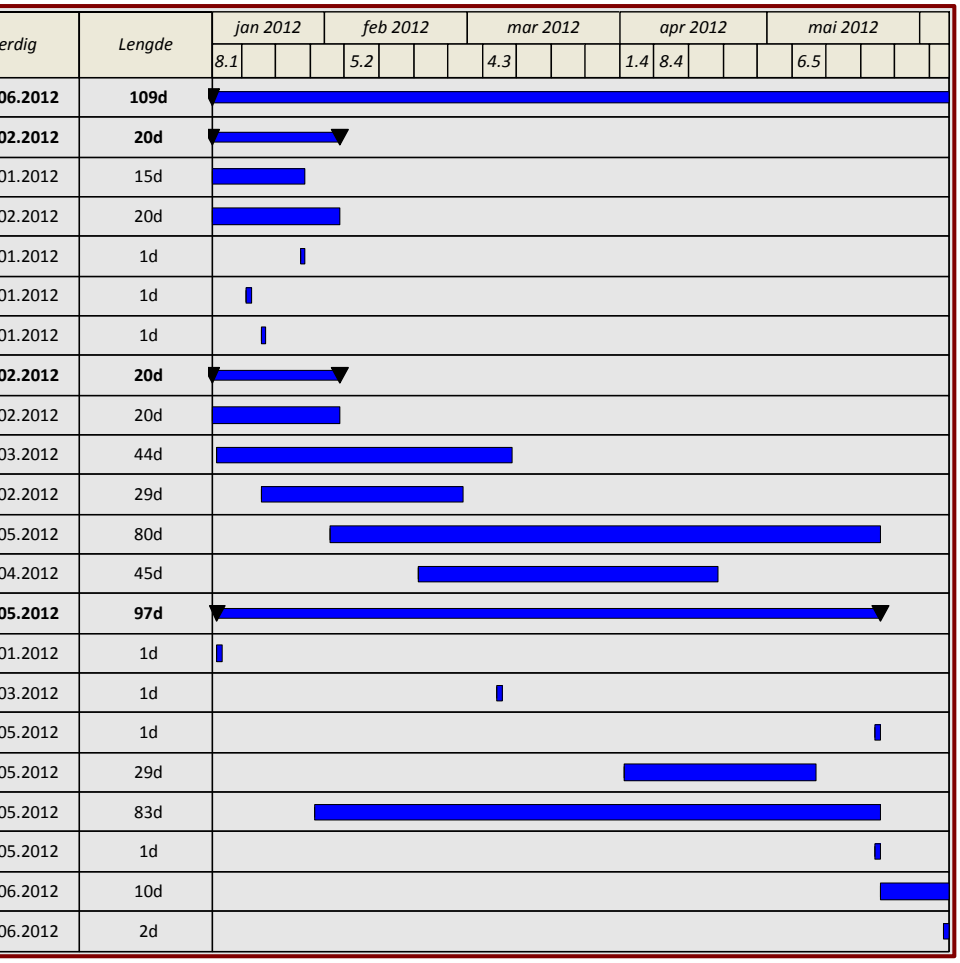
ID	Oppgave	Begynt	Ferdig	Lengde
1	**Bacheloroppgave**	**09.01.2012**	**06.06.2012**	**108d**
2	**Forprosjekt**	**09.01.2012**	**03.02.2012**	**20d**
3	Arbeid på forprosjekt	09.01.2012	27.01.2012	15d
4	Etablere webside	09.01.2012	03.02.2012	20d
5	Innlevering forprosjekt	27.01.2012	27.01.2012	1d
6	Møte med oppdragsgiver	16.01.2012	16.01.2012	1d
7	Møte med veileder	19.01.2012	19.01.2012	1d
8	**Leting etter lignende arbeid**	**09.01.2012**	**03.02.2012**	**20d**
9	Kildesortering	09.01.2012	03.02.2012	20d
10	Undersøke ytelsesprogramvare	10.01.2012	15.02.2012	27d
11	Utarbeide scenarier	19.01.2012	31.01.2012	9d
12	Litteraturstudie	02.02.2012	02.05.2012	65d
13	Ytelsestesting	10.02.2012	30.04.2012	57d
14	**Kurs**	**10.01.2012**	**21.05.2012**	**95d**
15	Lynkurs i prosjektplanlegging	10.01.2012	10.01.2012	1d
16	Lynkurs i rapportskriving	22.03.2012	22.03.2012	1d
17	Lynkurs i presentasjonsteknikk	21.05.2012	21.05.2012	1d
18	Rapportskriving	30.01.2012	23.05.2012	83d
19	Innlevering bacheloroppgave	23.05.2012	23.05.2012	1d
20	Presentasjon bacheloroppgave	06.06.2012	06.06.2012	1d



Figur I.1: Gantt-diagram fra forprosjektet

Gantt-diagram ferdigstilt i mai

ID	Oppgave	Begynt	Ferdig	Lengde
1	**Bacheloroppgave**	**09.01.2012**	**07.06.2012**	**109d**
2	**Forprosjekt**	**09.01.2012**	**03.02.2012**	**20d**
3	Arbeid på forprosjekt	09.01.2012	27.01.2012	15d
4	Etablere webside	09.01.2012	03.02.2012	20d
5	Innlevering forprosjekt	27.01.2012	27.01.2012	1d
6	Møte med oppdragsgiver	16.01.2012	16.01.2012	1d
7	Møte med veileder	19.01.2012	19.01.2012	1d
8	**Leting etter lignende arbeid**	**09.01.2012**	**03.02.2012**	**20d**
9	Kildesortering	09.01.2012	03.02.2012	20d
10	Undersøke ytelsesprogramvare	10.01.2012	09.03.2012	44d
11	Utarbeiding av scenarier	19.01.2012	28.02.2012	29d
12	Litteraturstudier	02.02.2012	23.05.2012	80d
13	Ytelsestesting	20.02.2012	20.04.2012	45d
14	**Kurs**	**10.01.2012**	**23.05.2012**	**97d**
15	Lynkurs i prosjektplanlegging	10.01.2012	10.01.2012	1d
16	Lynkurs i rapportskriving	07.03.2012	07.03.2012	1d
17	Lynkurs i presentasjonsteknikk	23.05.2012	23.05.2012	1d
18	Kompetanse innen Latex	02.04.2012	10.05.2012	29d
19	Rapportskriving	30.01.2012	23.05.2012	83d
20	Innlevering bacheloroppgave	23.05.2012	23.05.2012	1d
21	Forberede presentasjon	24.05.2012	06.06.2012	10d
22	Presentasjon av bacheloroppgave	06.06.2012	07.06.2012	2d



Figur I.2: Gantt-diagram ferdigstilt i mai

Vedlegg J

Forprosjekt

Vedlegg J inneholder forprosjektet som ble levert 27. januar. Gantt-diagram er ikke inkludert, men finnes i vedlegg I.

SkyHiGh I/O, forprosjekt bacheloroppgave 2012

Eirik Bakken (090382), Erik Raassum (090373)

27. januar 2012

Innhold

1 Bakgrunn, mål og rammer

1.1 Bakgrunn

Bakgrunnen for oppgaven er at sluttbrukerne – studenter og ansatte – som har benyttet seg av virtuelle maskiner i regi av Høgskolen i Gjøvik i dataemner, til tider har hatt problemer med ytelse. I forhold til regnekraft og minnebruk kan ytelse ofte lett forbedres ved å allokere mer maskinvare til problemet, men når det kommer til I/O er ikke svaret like svart-hvitt. Med I/O menes i denne sammenheng primært nettverks og datalagring, men også klientaksess.

Nettskyer (cloud computing) er en spennende teknologi som har fått mye oppmerksomhet de siste årene. Den gjør det mulig for sluttbrukere å kjøre sine virtuelle maskiner på andres maskinvare og infrastruktur. En slik sky vil bli satt opp på HiG i løpet av våren 2012, og det bringer på bane mulige utfordringer i forhold til tilfredsstillende I/O.

Det er flere emner på HiG som kan dra stor nytte av å bli flyttet over i skyen. Fag som i dag krever at studenter enten benytter seg av lab-pcer eller virtuelle maskiner på egne bærbare maskiner, vil være enklere å gjennomføre og forberede undervisningen til, hvis man har ytelse som svarer til fysiske maskiner, og i tillegg mindre strev med oppsett. Fag hvor dette vil være spesielt fordelaktig er "Operativsystemer", "Ethical hacking and penetration testing", "Systemadministrasjon" og "Database- og applikasjonsdrift". Det er også et behov for det i forskningsøyemed, hvor studenter eller ansatte kan få avsatt den ytelsen de trenger.

Oppdragsgiver ønsker derfor en utredning hvor man ser på disse potensielle problemene i en nettskykontekst, vurderer og drøfter ulike årsaker til ytelsesproblemer og forsøker å utlede noen generelle anbefalinger i forhold til å forbedre dette området.

1.2 Rammer

Gruppen må forholde seg til de rammene som er fremsatt av fakultetet og høgskolen i forhold til gjennomføringen av oppgaven. Med dette menes tidsfrister i løpet av våren, blant annet ved innlevering av forprosjektrapport, kontrakt mellom oppdragsgiver og gruppen, og sluttrapport i mai. Andre krav til oppgavegjennomføringen er statusmøter og loggføring i løpet av prosjektets gang.

2 Oppgavebeskrivelse

Her følger oppgavebeskrivelsen fra oppdragsgivers side. Den endelige utgaven kan bli videre spisset eller utvidet fra dette utgangspunktet.

- Identifisere hvilke krav de ulike, mest relevante emnene stiller til ytelse i en skykontekst.
- Vurdere og drøfte de ulike alternativene til virtualisert/ekstern lagring og virtuelle nettverk.
- Ytelsestesting for å utlede generelle anbefalinger rundt oppsett, primært av permanent lagring (over iSCSI) og nettverk i en OpenStack-arkitektur.
- Identifisere når og hvor flaskehalser kan oppstå.

2.1 Prosjektmål

De ulike prosjektmålene er effektmål, som viser hvilke positive virkninger det kan bringe på banen; læringsmål, som sier noe om hva gruppen ender opp med å ha lært, mens resultatmål er de faktiske, målbare og konkrete vurderingene og anbefalingene som fremkommer i kjølvannet av denne bacheloroppgaven.

2.1.1 Effektmål

- Sluttbrukerene opplever, om mulig, mer pålitelige, sømløse og tilfredsstillende tjenester ved implementering av frembrakte "best practices".

2.1.2 Læringsmål

- Tilegne oss verdifull kunnskap innenfor yrkesrelevante fagområder (herunder nettskyer, virtualisering, ytelse- og ytelsesvurderinger og overvåkning).
- Utvikle ferdigheter for å gjennomføre prosjekter av større omfang.

2.1.3 Resultatmål

- Komme frem til relevante "best practices", som gjelder for ytelse i forhold til lagrings- og nettverksrelaterte hensyn i en privat OpenStack-arkitektur til akademisk bruk, slik at eventuelle sluttbrukere opplever en så god tjeneste som mulig.
- Tilstrebe at de virtuelle maskinene nærmer seg ytelsen som tilsvarende fysiske maskiner har.

2.2 Avgrensninger

Hovedsaklig vil gruppen i denne oppgaven:

- Undersøke hvilke krav emner stiller til maskinkraft.
- Se på alternativer for lagrings- og nettverksløsninger i OpenStack.
- Teste ytelse på nettverk og ekstern lagring (iSCSI) i en OpenStack-kontekst. Hensynet ligger hele tiden på på hva som er best for skolen, det vil si akademiske formål.
- Finne optimaliseringstiltak for å øke ytelsen.

Oppgaven vil i utgangspuntet ikke omhandle klientaksess til virtuelle maskiner på OpenStack-arkitekturen som implementeres, med mindre det blir tid til overs.

Valg av infrastrukturoppsett bestemmes av en annen bacheloroppgave-gruppe (SkyHiGh Adm), mens denne oppgaven vil se på hvordan I/O kan optimaliseres.

3 Prosjektorganisering

3.1 Ansvarsforhold og roller

Erik Raassum vil være gruppens prosjektleder, mens Eirik Bakken fungerer som dokumentasjons- og webansvarlig. Ved vanlig prosjektvirksomhet vil gruppen i størst mulig grad jobbe sammen i fellesskap, slik at begge har mest mulig utbytte av prosjektets innhold og arbeidet med dette.

Oppgavens oppdragsgiver er Erik Hjelmås (førsteamanuensis ved HiG), studieprogramansvarlig for bachelorprogrammet "drift av nettverk og datasystemer". Veileder for oppgaven er Hanno Langweg (førsteamanuensis ved HiG).

3.2 Grupperegler og rutiner

- Arbeid på bacheloroppgaven gjennomføres i all hovedsak i perioden 8-15 på grupperom A030, alle hverdager, med unntak av tidspunkt for forelesninger og arbeid i faget WWW-teknologi, som tas parallelt av begge gruppemedlemmer. Dersom mer tid viser seg å være nødvendig, avgjøres det fortløpende hvordan dette skal allokeres. I snitt legges det opp til 30 timer arbeid per deltaker, per uke.

- Hver mandag gjennomføres et kort møte, hvor plan legges for inneværende ukes gjøremål. I tillegg vil det hver dag det skal jobbes med prosjektet, bli hentet frem gjøremål hver gruppedeltaker, eller begge, skal arbeide med den dagen.

- Gruppedeltakere skriver logg for hver dag som det er arbeidet med oppgaven, og dokumenterer antall timer jobbet, samt hva som er blitt gjort.

- Uenigheter skal i første omgang forsøkes løst innad i gruppen. Skulle det bli behov, vil uenigheten søkes løst i samarbeid med veileder.

- Fravær må begrunnes, og leveres i muntlig form til gruppeleder.

- Referater skrives etter alle større møter, og ellers der det er behov for oppsummering av aktiviteter knyttet til oppgaven.

- Alle dokumenter og filer knyttet til oppgaven skal lastes opp til gruppens dedikerte filområde. Hver dag tas det også backup til minst én minnepenn.

4 Risikoanalyse

I risikoanalysen gjennomgås relevante situasjoner som kan forekomme i løpet av de neste fem månedene. Disse situasjonene kan komme til å ha stor innvirkning på prosjektes gjennomførbarhet eller kvalitetsmessige innhold. Sammen med hver situasjon, er det foreslått tiltak som kan begrense omfang, eller redusere sjansen for inntreffelse.

4.1 Teknologiske

Beskrivelse	Sannsynlighet	Konsekvens	Tiltak
Openstack blir for vanskelig å sette seg inn i.	Lav	Høy	Sette oss godt inn i dokumentasjonen.
Hardware-feil	Lav	Middels	Ta backup jevnlig. Ha om mulig reservedeler for kritiske komponenter.
Finner ikke relevant programvare.	Lav	Høy	Begynn tidlig med å lete etter programvare.

4.2 Personellrelaterte

Beskrivelse	Sannsynlighet	Konsekvens	Tiltak
Sykdom innad i gruppen	Middels	Middels	Gode rutiner for å dele arbeidsmengde og informasjon. Komme godt i gang tidlig for å ha tilstrekkelig slingringsmonn ved uforutsette omstendigheter.

4.3 Prosjektmessige

Beskrivelse	Sannsynlighet	Konsekvens	Tiltak
Tap av prosjektdokumenter.	Lav	Høy	Ta høyde for datatap ved å ta regelmessig backup, og i tillegg bruke versjonskontroll.
Tidspress eller manglende ferdigstillelse av oppgave.	Lav	Høy	Komme godt i gang tidlig. Dele prosjektet inn i klart definerte faser for å lett kunne se hvordan vi ligger an.

5 Plan for gjennomføring

5.1 Prosessrammeverk

På grunn av bacheloroppgavens store omfang, i tillegg til den lange perioden det skal jobbes med denne, vil det være fordelaktig å få på plass et prosessrammeverk/systemutviklingsmodell man kan ta utgangspunkt i. Dette vil komme oss til gode både i forkant av, og under arbeidet på oppgaven. Selv om man ikke har et utpreget systemutviklingsrelatert prosjekt, finnes det likevel mange egenskaper og fordeler ved de ulike prosessrammeverkene som kan gagne et arbeid i samme fagfelt.

Gruppen har vurdert to spesielt relevante og forskjellige modeller: Fossefallsmodellen og inkrementell modell. Fossefallsmodellen fordrer meget god initiell oversikt over hele løpet til prosjektet, fra start til slutt. Dette egner seg godt hvis man ved oppgavens begynnelse vet eksakt hvilke trinn du må gjennom, og på hvilke tidspunkt. Denne oppgaven er av en mer dynamisk art, som vil kreve hyppige endringer og revisjoner, noe som ikke helt tilfredsstiller fossefallsmodellens karakteristika.

Derfor har gruppen lagt seg på en mer inkrementell linje, der en er friere til å være dynamisk, gå tilbake og se på tidligere funn og oppgaver, og undersøke disse i lys av nye oppdagelser. Likevel vil det forsøkes så godt det lar seg gjøre å dele oppgaveløpet inn i klart definerte faser, slik at en viss ryddighet, struktur og målbar fremgang vises i arbeidet.

5.2 Milepæler

De viktigste datoene for bachelorarbeidet er avtegnet nedenfor.

- Levere prosjektplan og prosjektavtale med oppdragsgiver innen 27.01.12 kl. 12.00
- Nettside skal være etablert innen 03.02.12
- Bacheloroppgaven leveres i Fronter den 23.05.12 innen kl. 12.00

5.3 Statusmøter

Det er lagt opp til ukentlige møter med veileder, Hanno Langweg, med offisiell oppstart torsdag den 19.01. Skulle ytterligere interaksjon være nødvendig, skjer denne etter behov. Møte med oppdragsgiver vil også skje ukentlig, samtidig med møte med veileder. Skulle det ikke være behov for disse møtene, gis det i god tid avklaring per mail eller telefon.

Det legges også opp til statusmøter for å ta temperaturen på arbeidet, for å forsikre deltakerene og de andre ressursene i arbeidet, om at jobben skrider fremover med riktig tempo og i henhold til gjeldende tidsplaner og retningslinjer.

5.4 Dokumentasjon og versjonshåndtering

Arbeidet på den avsluttende rapporten vil – mens arbeidet pågår – bli skrevet i Google Docs for effektiv samhandling mellom gruppemedlemmene. Deretter blir den konvertert til Latex ved slutten av prosjektet av estetiske og praktiske årsaker. Google Docs har funksjonalitet for å få tilbake tapt arbeid gjennom "revisjonshistorie", slik at man ved uhell ikke mister arbeid. I tillegg vil Dropbox bli brukt som backup for andre dokumenter eksterne fra Google Docs. På denne måten slipper man ugunstige situasjoner hvor man taper timer eller dager med arbeid, og i tillegg tilrettelegger for strukturert og samordnet arbeid, selv om den administrative belastningen blir noe større.

Gantt-diagram vil bli produsert i programmet "Microsoft Visio". Gruppen forsøkte en rekke opensource-varianter, blant annet "GanttProject", men det viste seg å bli vanskelig å få til en konsis visuell representasjon som kunne vises på én A4-side på en god måte. Visio er internasjonalt anerkjent i tillegg, som understøttet denne avgjørelsen.

For websiden vil Wordpress benyttes, et open-source bloggrammeverk som kan tilpasses i det uendelige. Dette vil gjøre oppdatering av oppgaven lettere, og det finnes en del gratis, estetisk tilfredsstillende temaer man kan benytte seg av.

Rapportene som produseres i arbeidet med bacheloroppgaven skal forholde seg til fastsatte maler og retningslinjer.