Distributed File Systems
Inzetbaar als data consistentie houder?

Jan van Lith en Maarten Michels

4 juli 2005

Samenvatting

Het consistent houden van data tussen 2 systemen is lastig. Het maken van backups, het opslaan van data op een server terwijl er geen toegang is tot het bedrijfsnetwerk omdat ze onderweg zijn, etc. In dit onderzoek gaan wij ons richten op een manier om het consistent houden van data mogelijk te maken met een Distributed File System (DFS). Bij een gedistribueerd file systeem staan bestanden op een of meerdere servers en worden ze toegankelijk gemaakt voor gebruikers zodat ze als lokale bestanden te zien zijn.

Het onderzoek richt zich op de inzetbaarheid van de volgende gedistribueerde file systemen; Coda, GFS en Lustre. Er is voor elk DFS onderzocht of het inzetbaar is en op welke manier dit kan. Er is ook een Proof of Concept gedaan met Coda in een client-server scenario. Deze proof of concept beschrijft een methode hoe een client-server systeem opgezet kan worden om hierna de performance van het DFS bij verschillende bandbreedtes en latency's te testen.

Als conclusie uit dit onderzoek is naar voren gekomen dat Coda een file systeem is dat geschikt is om in te zetten in een client-server scenario. Op dit moment is er nog geen goed DFS voor het consisitent houden van data tussen 2 servers. Lustre is echter wel druk bezig om dit te realiseren maar het zal nog enige tijd duren voordat het inzetbaar is.

Inhoudsopgave

1 Inleiding

Bij een gedistribueerd file systeem (DFS) staan bestanden op een of meerdere servers en worden ze toegankelijk gemaakt voor gebruikers zodat ze als lokale bestanden te zien zijn. Veranderingen die gemaakt worden op de client, worden als er verbinding is met de server, automatisch gesynchroniseerd. In dit onderzoeksrapport wordt bekeken of het mogelijk is om een DFS in te zetten om de data van een server naar een andere server te synchroniseren op het moment dat er data veranderd wordt. Door dit principe wordt het mogelijk om bijvoorbeeld data met een server op een co-locatie consistent te houden met data op een server in de hoofdlocatie. Hier vandaan kunnen dan backups gemaakt worden zodat geen backup faciliteit op de co-locatie aanwezig hoeft te zijn. Ook word er gekeken of een DFS ingezet kan worden in een client-server model. De data op de client zal dan automatisch gesynchroniseerd worden met de server. In dit rapport zullen drie Distributed File Systemen worden bekeken op hun inzetbaarheid als data consistentie houder. Aangezien een DFS over een netwerk opereert is het handig om te weten hoe een DFS presteert bij een slechte verbinding. De kwaliteit van een netwerk wordt bepaald door de bandbreedte en de latency van de verbinding. Deze kwaliteit kan getest worden met de Proof of Concept die in dit verslag wordt beschreven.

1.1 Opbouw

Dit onderzoeksverslag zal beginnen met een uitleg over het project, de probleemstelling beschrijven en de aanpak met de gehanteerde planning. Daarna zullen drie Distributed File Systemen; GFS, Lustre en Coda aan bod komen. De werking zal uitvoerig worden behandeld. Hierna wordt gekeken of de drie file systemen inzetbaar zijn voor het consistent houden van data in een client-server en server-server scenario. Hierna wordt de Proof of Concept met Coda, in een client-server scenario, omschreven. Na de omschrijving van de Proof of Concept wordt ook daadwerkelijk Coda getest. Er wordt gekeken naar de performance bij verschillende bandbreedtes en latency's. Het resultaat van de uitkomst van deze testen zullen worden behandeld en er zal een analyse van deze resultaten worden gedaan. Als laatste zal er een conclusie worden gegeven.

1.2 Dankwoord

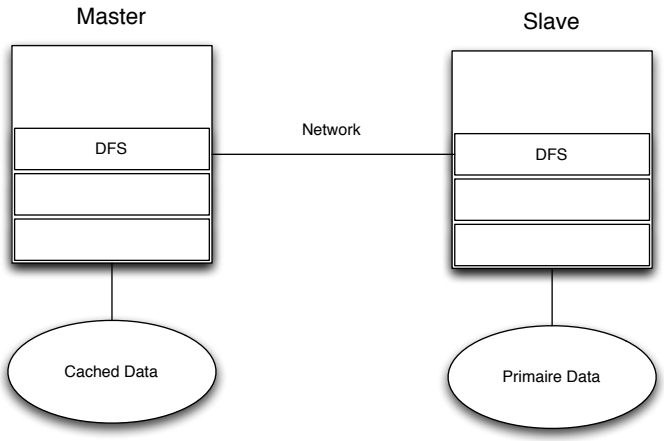
We willen graag Fred Mobach bedanken voor dit interessante onderwerp waarin we onderzoek hebben mogen verrichten. Ook danken wij hem voor het nakijken van dit onderzoeksverslag. We danken de opleiding OS3 voor het beschikbaar stellen van de faciliteiten die nodig waren om dit onderzoek te verrichten.

2 Het project

Een eigenschap van een DFS is dat zodra er data op het distributed file systeem wordt aangepast of aangemaakt dit ook wordt doorgegeven aan andere systemen die DFS draaien. Er zijn verschillende scenario's waarin DFS gebruikt kan worden om de data tussen 2 systemen consistent te houden. In dit onderzoek zullen we ons beperken tot de volgende 2 scenario's:

Scenario 1 client-server Een gebruiker dient bij verlies van netwerk connectiviteit met de file-server zijn data aan te kunnen passen en bij terugkeer van netwerk connectiviteit automatisch deze data consistent met de data op de file-server te krijgen.

Scenario 2 server-server Op een bedrijf staat centraal een file-server. Op een co-location of naast de file-server staat een backup-server. De bestanden op de file-server kunnen dan automatisch gesynchroniseerd worden met de backup-server zodat er consistentie onstaat.



Figuur 1: Opdracht

In figuur 1 is schematisch weergegeven hoe het DFS gebruikt kan worden bij het consistent houden van data. In het geval van scenario 1 zal de client de master zijn en de server de slave. In het geval van scenario 2 zal de master de file-server op de co-locatie zijn en de slave op de centrale locatie. De data-uitwisseling tussen de Master en de Slave zal door het DFS worden afgehandeld. De Slave zal de primaire data bevatten die gebackup-ed kan worden. De data op de Master is de gechachte data. De data hierop aanwezig zal over een netwerk automatisch gesynchroniseerd worden met de Slave.

2.1 Probleemstelling

Het onderzoek zal zich richten op de inzetbaarheid van de volgende DFS-en; Coda, GFS en Lustre, in het client-server scenario en in het server-server scenario. Er zal voor elk DFS onderzocht worden of het inzetbaar is en op welke

manier deze inzetbaar is voor de 2 verschillende scenario's. Er zal een proof of concept worden gedaan met Coda voor het client-server scenario. Deze proof of concept zal een methode beschrijven hoe een client-server systeem opgezet kan worden om hierna de performance van het DFS bij verschillende bandbreedtes en latency's te testen.

2.1.1 Onderzoeksvragen

In dit onderzoek willen we antwoord geven op de volgende onderzoeksvragen:

- Kunnen Coda, GFS of Lustre ingezet worden voor het consistent houden van de data op een client in een client-server omgeving?
- Kunnen Coda, GFS of Lustre ingezet worden voor het consistent houden van data op de servers in een server-server omgeving?
- Op welke manier kunnen de Distributed File Systems ingezet worden.
- Wat is de performance van Coda mbt bandbreedte en latency in een client-server scenario?

2.1.2 Randvoorwaarden

De onderzoeksperiode van dit project is 2 weken. In deze 2 weken kan niet alles onderzocht worden en dus zijn er enkele randvoorwaarden aan het onderzoek gesteld. Geen onderdeel van dit project zullen zijn:

- Het maken van een gedetailleerde HOW-TO voor het opzetten van Coda.
- Het onderzoeken wat er gebeurt als een synchronisatie plaats vind wanneer een gebruiker een bestand offline aanpast en een andere gebruiker dit online op de server doet.
- Alleen de 2 aspecten van performance; latency en bandbreedte zullen onderzocht worden. Geen andere performance aspecten zullen onderzocht worden.
- Het onderzoek beperkt zich tot de inzetbaarheid op 1 soort besturingssysteem namelijk Linux/BSD.

2.2 Aanpak

Het onderzoek is opgedeeld in een theoretisch en praktisch gedeelte.

2.2.1 Theoretische aanpak

Er is literatuur over de verschillende DFS-en geraadpleegd om te kunnen beschrijven op welke manier Coda, GFS en Lustre inzetbaar zijn voor de 2 verschillende scenario's. De scenario's worden apart bekeken.

2.2.2 Practische aanpak

De Proof of Concept wordt opgezet met Coda voor een client-server scenario. Het is interessant om te weten te komen wat de invloed is van het netwerk tussen 2 systemen. Vandaar dat er wordt gekeken naar de performance van het DFS. Er is getest met de volgende bandbreedtes:

- 128 Kbps
- 256 Kpbs
- 512 Kpbs
- 1024 Kpbs
- 2048 Kpbs
- 100 Mbps

Deze bandbreedtes zijn de meest voorkomende en verdubbelen zodat ze een goed verloop zullen weergeven.
De volgende latency's zijn per bandbreedte getest:

- Latency van het LAN (afhankelijk van de gekozen bandbreedte)
- 50 ms
- 100 ms
- 200 ms
- 400 ms

Dit alles is met een bestand van 5MB getest.

2.3 Planning

De volgende planning binnen dit project is gehanteerd:

Week 1 In deze week wordt het projectplan gevormd. Er wordt ingelezen in de literatuur over het onderwerp.

Week 2 Hierin worden de verschillende DFS-en onderzocht. Er wordt ook al beschreven op welke manier deze DFS-en ingezet kunnen worden in de 2 scenario's.

Week 3 In deze week zal de proof of concept worden geïnstalleerd en zal de performance worden getest.

Week 4 In deze week zal het verslag worden geschreven en zal als eventuele uitloopwijk gebruikt worden.

3 DFS

In dit hoofdstuk worden Coda, GFS en Lustre uitgelicht. De historie en werking wordt beschreven. Ook worden AFS en InterMezzo even besproken maar zal niet diep op de werking ingegaan worden omdat deze systemen geen onderdeel van dit onderzoek zijn wegens tekortkomingen die bij het hoofdstuk 4 worden uitgelegd. Na de beschrijving van de werking van deze file systemen zal de inzetbaarheid per file systeem behandeld worden.

Belangrijk onderdeel bij de inzetbaarheid van deze distributed filesystemen is de ondersteuning van een *disconnected operation*. Dit zorgt ervoor dat data die aangemaakt, verwijderd of aangepast wordt tijdens verlies van netwerkconnectiviteit of een uitval van een server, bij het herstel van netwerkconnectiviteit of herstel van de server weer consistent is met de data op de server. Stel we hebben dus een client die aan het netwerk hangt en aan het werk is. Als de netwerkverbinding dan weg valt moet de client nog gewoon door kunnen werken. De bestanden op het DFS dienen dan bij een terugkeer van de netwerkverbinding gesynchroniseerd te worden met de server om zo data consistent te houden.

3.1 AFS

AFS [1] staat voor Andrew File System en is ontwikkeld op de Carnegie-Mellon Universiteit in 1984. Het doel van AFS was het maken van een filesysteem dat, overal op universiteitscampus beschikbaar zou zijn, over een netwerk met geringe bandbreedte. Het wordt op dit moment als commercieel product aangeboden door IBM en als open source implementatie als openAFS of Arla. Het ondersteund vele besturingsystemen.

AFS gebruikt een lokale cache om de workload te reduceren en de performance te verbeteren. Als data opgevraagd wordt zal de data van de server eerst in de lokale cache worden gezet. De eerst volgende keer dat de data opgevraagd wordt door de client wordt de data uit de lokale cache gehaald en hoeft deze niet de data van de server te downloaden.

AFS werkt met volumes, dit is een boomstructuur van files en subdirectories. Deze volumes worden door een administrator aangemaakt en verbonden aan een padnaam in een AFS cel. Een cel is een collectie van file servers en client systemen. Deze volumes kunnen zich in meerdere getallen bevinden over de verschillende servers. Verder gebruikt AFS keberos voor authenticiteit en kent het accesslists voor beveiliging van de data.

3.2 InterMezzo

InterMezzo [7] is een DFS dat geïnspireerd is door Coda en ook gemaakt door de Carnegie-Mellon Universiteit. InterMezzo wordt gesupport door kernels hoger dan 2.4.5 en is uitgebracht onder de GNU GPL[6] licentie.

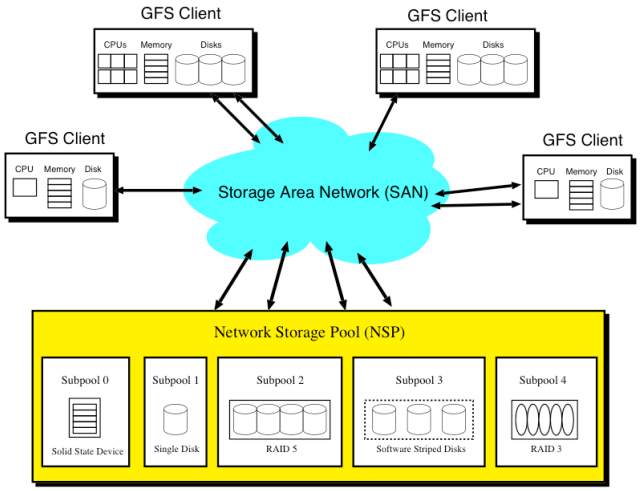
InterMezzo exporteert de data die is opgeslagen in een journaling filesysteem, dit kan bijvoorbeeld ext3, jfs of ReiserFS zijn. Het is opgebouwd uit kernel componenten en een server proces ”InterSync” dat verantwoordelijk is voor het synchroniseren van files tussen client en server. InterSync zorgt voor het consistent houden van de data op de client doormiddel van een periodieke

poll naar de server. De server houdt een log bestand bij met daarin al de veranderingen op de files die hij bezit en stuurt dit tijdens de poll naar de client. De client vergelijkt dit log bestand met zijn ge-cachte data en mocht er een verandering opgetreden zijn dan haalt hij het vernieuwde bestand met behulp van het HTTP protocol op van de server.

3.3 GFS

GFS [5] is ontworpen als onderdeel van een promotie project bij de universiteit van Minnesota. Na een verblijf van een paar jaar bij Sistina Software heeft Red Hat GFS nu onder zijn hoede en is het onder de GNU GPL licentie beschikbaar.

GFS is een opslag architectuur voor clusters die data opslag beschikbaar maakt voor de clients in een cluster. Het is schaalbaar tot honderden node's. De structuur van het cluster zorgt ervoor dat gebruikers de disk data over een Fibre Channel LAN kunnen opvragen. Alle gebruikers in het cluster kunnen dezelfde data opvragen met perfecte consistentie. Figuur 2 geeft de structuur van GFS weer.



Figuur 2: Een GFS cluster

Zoals je kan zien worden de netwerk opslag media beschikbaar gemaakt voor de clients. Elke client ziet dit data opslag medium als een lokaal bruikbaar opslag medium. Clients werken ook onafhankelijk van elkaar, er vindt geen communicatie tussen deze systemen plaats zodat andere clients geen hinder ondervinden van fouten van andere clients. Bij het aanpassen van data wordt er een *lock* gevraagd aan het data opslag device. Deze lock geeft aan dat de file in gebruik is. Na het aanpassen wordt de lock weer opgeheven. Dit zorgt voor consistentie, de meest recente data zal bij opvragen terug worden gegeven. Clients weten dus niks van elkaar en zijn geheel onafhankelijk.

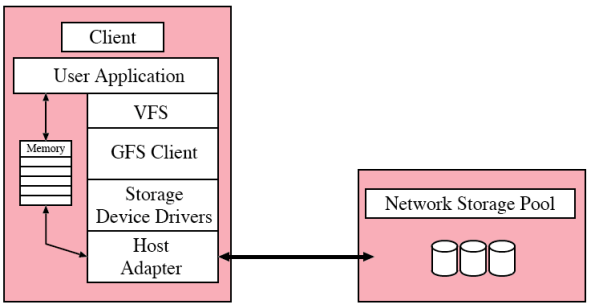
De communicatie tussen de clients en de data opslag devices gaat via channel netwerken, zogenaamde Storage Area Networks (SAN). Een SAN is een

soort netwerk dat devices aan het netwerk verbindt via channel interfaces, zoals SCSI. GFS groepeert deze devices tot een Network Storage Pool (NSP). In deze pool bevinden zich weer subpools die dezelfde devices groepeert. Het opdelen in subpools kan de performance verbeteren. Bij het implementeren van GFS kunnen grote bestanden opgeslagen worden op subpools met snelle devices. Clients die dan grote bestanden opvragen doen dit via de snelle subpools.

Grote data opslag capaciteiten beïnvloeden de structuur van het file systeem en de caching policies. GFS richt zich op applicaties die veel data opslag capaciteit en bandbreedte nodig hebben. GFS heeft geen ingebouwde security, security kun je bereiken door Network Attached Secure Disks (NASD) te gebruiken. Deze disks ondersteunen authenticatie schema's. Een GFS client kan de data die hij verkrijgt via GFS ook weer delen door bijvoorbeeld een NFS server te draaien en de data vandaar te exporteren.

GFS doet aan device caching. Meerdere clients spreken dezelfde data aan en zullen dus profijt hebben van device caching. Device caching gebeurt door specifiek de data aan te geven die gecached moet worden en niet de recent gebruikte data zoals dat normaal gebeurt. Clients geven aan, in een verzoek naar de server, welke data voor hun in aanmerking komt om gecached te worden. Veel opgevraagde data zal dus snel voorhanden zijn.

In figuur 3 is te zien welke lagen in GFS worden doorlopen. Na user appli-



Figuur 3: GFS control en data pad

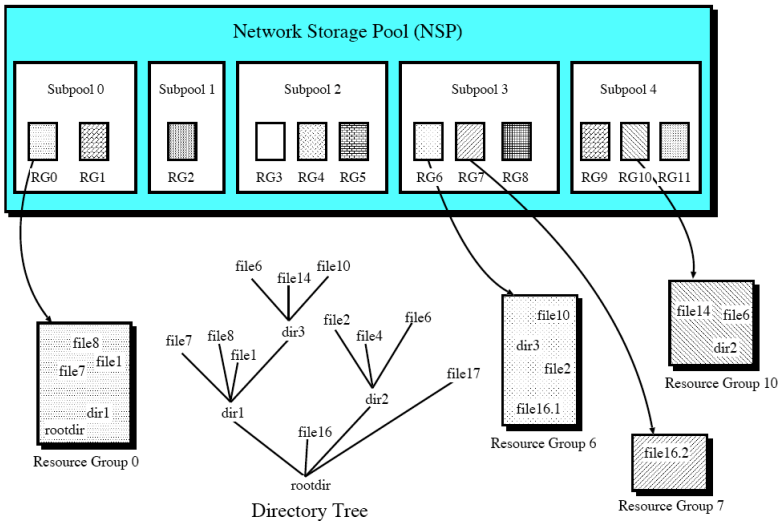
caties wordt het Virtual File Systeem (VFS) aangesproken. Hierna wordt via client GFS de storage device driver aangesproken. Dit is bij andere file systemen anders. Daar moet eerst nog tcp/ip laag worden doorlopen en dan pas wordt de device driver van het netwerk aangesproken. Op de server moeten dan ook weer vele lagen doorlopen worden. Bij GFS wordt meteen de NSP aangesproken. Dit direct aanspreken van de storage device driver betekent echter wel dat data uitwisseling niet direct via een tcp/ip netwerk kan verlopen.

De NSP beheert de *device locks*. De NSP ontvangt de lock commando's met een logical lock number die hij vertaald in een fysiek nummer van het desbetreffende device. De NSP pollt het device met een ingestelde tijd. Deze tijden dienen afgestemd te worden op de workload. De NSP handelt ook fouten in locks af en kent hiervoor herstelprocedures.

De NSP bevat Resource Groups die de file systeem resources over het hele NSP distribueren. Een RG is als het ware een mini-file systeem. Elke infor-

matie groep bezit een informatie block, data bitmaps, dinodes en data blocks. In dit block staat informatie over de hoeveelheid clients die gemount zijn met het systeem en ook en ook de Resource Group Index (RGI). Deze index geeft de locatie en attributen weer van elk RG. De bitmaps calculeren unieke identificeerders voor elke client. Een dinode is als het ware de inode voor GFS. Het verschil is echter dat er niet meerdere dinodes per block worden geplaatst, dit single block sharing is niet efficiënt bij DFS-en. Omdat er dan vaker verkeer wordt gegenereerd en dit weer naar de server moet getransporteerd worden en dus een belasting voor het netwerk en de server vormen vandaar dat gebruik wordt gemaakt van meerdere dinodes per block. De RGs zijn bij normaal gebruik transparant voor de gebruiker.

In figuur 4 is te zien hoe de directory tree verdeeld wordt over verschillende



Figuur 4: GFS Resource Groups

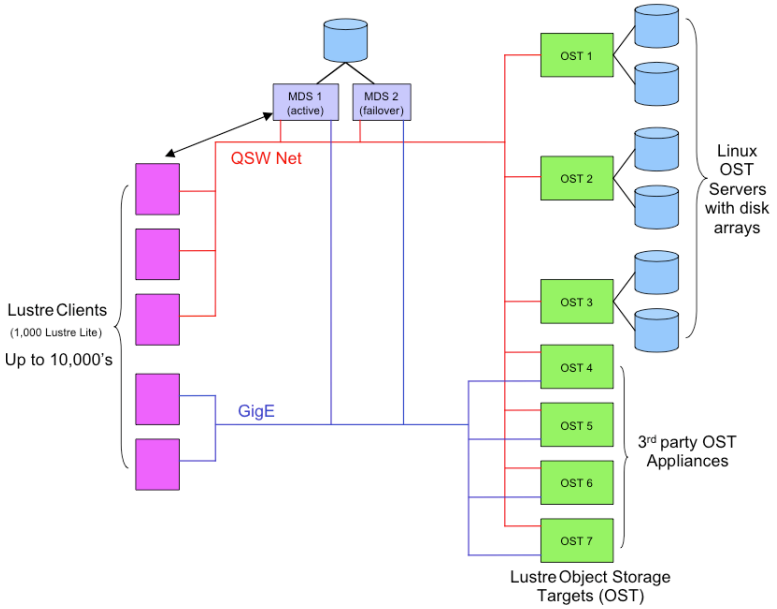
RGs in de verschillende subpools. Deze verdeling over de verschillende devices zorgt bij het openen van bestanden voor parallellisatie en dat komt de performance van het systeem ten goede.

3.4 Lustre

Het Lustre [9] project is in 1999 ontwikkeld door de Carnegie-Mellon Universiteit (Gesponsord door Seagate) om een object based file system met cluster mogelijkheden te ontwikkelen. Tegenwoordig wordt Lustre onderhouden door Cluster File Systems. In 2000 wordt de ontwikkeling beïnvloed door National Labs en Tri-Labs/NSA. Lustre is verkrijgbaar onder de GNU GPL licentie. Lustre kan data transporteren over TCP en QSWNet netwerken. Andere netwerken zijn in ontwikkeling. De naam Lustre is een samenvoeging van Linux en clusters.

In Lustre wordt een bestand op file systeem niveau als een object gezien. Deze objecten bevinden zich op Metadata Servers, zogenaamde MDSs. Met-

adata is informatie over de bestanden en directories die tezamen een file systeem maken. Deze informatie kan informatie zijn over locale bestanden of directories maar ook informatie over mount points en symbolische links. De Metadata Servers ondersteunen alle file systeem operaties zoals; file lookup, file creation, file en directory attribute manipulation, enz. Ook zorgt de MDS ervoor dat I/O requests naar de Object Storage Targets (OSTs) worden geleid. Deze OST beheert de data die op Object-Based Disks (OBDs) staan en heeft een pointer die naar de plaats van het object wijst. Als een bestand geopend wordt zorgt de MDS ervoor dat de daadwerkelijke dataoverdracht via het OST verloopt. Het MDS houdt de metadata veranderingen bij, de cluster status en heeft een failover mechanisme dat bij verlies van netwerk connectiviteit de operatie niet verstoord (wordt verderop nog behandeld). Het aanmaken en schrijven van bestanden gebeurt dus via het MDS dat verbinding maakt met de OST die het vervolgens weer wegschrijft op de OBDs. Data wordt verspreid over verschillende OST systemen voor een betere performance. In figuur 5 wordt een overzicht van Lustre weergegeven.



Figuur 5: Lustre overzicht

Lustre maakt gebruik van inodes, deze bevatten een verwijzing naar de objecten op de OSTs die de bestandsdata bevatten in plaats van de verwijzing naar het bestand zelf zoals dat bij andere file systemen is geregeld. In Lustre wordt bij het creëren van een nieuw bestand de MDS geraadpleegd die vervolgens een inode voor het bestand creëert en de OST aanspreekt om de objecten aan te maken waar de bestandsdata in opgeslagen kan worden. Zo wordt metadata gescheiden van de eigenlijke data en onderling opgeslagen op verschillende servers. De eigenlijke data kan ook nog verspreid worden over meerdere OSTs.

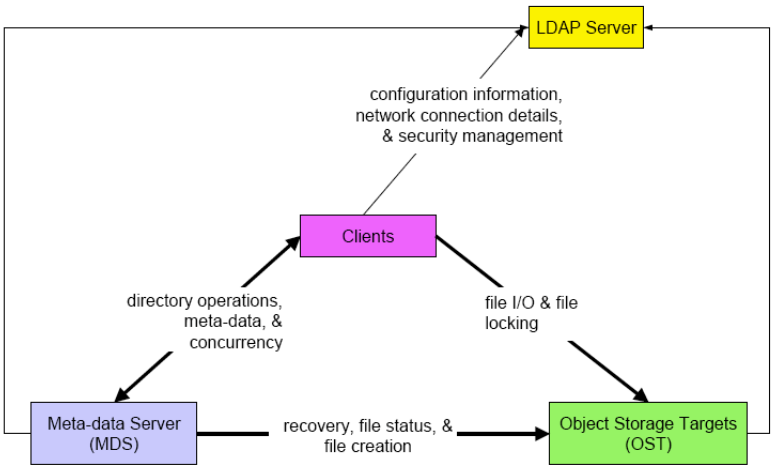
De OST handelt dus de data requests van een client af en de uitwisseling

met het opslag medium. De interactie met het opslag medium gebeurt via een device driver. Lustre kent OBD device drivers die data opslag in journaling Linux file systemen kent zoals; ext3, JFS, ReiserFS en XFS. Het maakt dan gebruik van de mechanismen van het al bestaande journaling file systeem.

OST bezit ook een model dat het makkelijk maakt om nieuwe opslag capaciteit toe te voegen. Nieuwe OSTs kunnen gemakkelijk toegevoegd worden aan het cluster zodat de MDS deze kan gebruiken. Ook de ODBs kunnen gemakkelijk toegevoegd worden aan een OST.

Lustre kent een recovery mechanisme dat gebruikt kan worden bij verlies van connectiviteit of storage failure. Als de data niet kan worden opgehaald vraagt de client de LDAP server of er een vervangende MDS is en stuurt het direct zijn requests daar naartoe. Als de failover niet lukt dan zal Lustre zich automatisch aanpassen. Bij een failure van een OST zal de OSC ervoor zorgen dat nieuwe data op andere OSTs terecht komen.

De interacties tussen de verschillende subsystemen die lustre kent staan weergegeven in figuur 6.



Figuur 6: Lustre's interacties tussen subsystemen

Lustre's configuratie is opgeslagen in een XML bestand dat conformeert met een DTD die gepubliceerd is. Dit maakt het eenvoudig aan te passen met bijvoorbeeld text editors. Het gebruik van een gepubliceerde DTD maakt het voor third-party utilities eenvoudiger te integreren. Configuratie bestanden worden gegenereerd en geupdated door de Lustre make configuration utility (lmc). LDAP wordt gebruikt voor redundancy en assisteert bij cluster recovery's. Lustre kent een eigen SNMP MIB die monitoring mogelijk maakt.

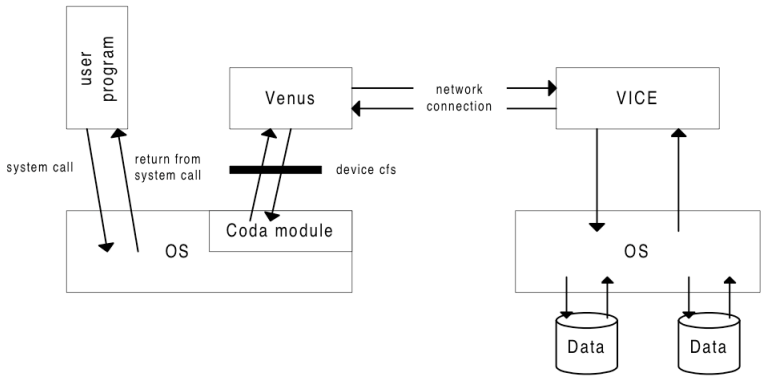
3.5 Coda

Coda [2] is ontwikkeld op de Carnergie Mellon Universiteit rond 1987, Het is gebaseerd op een oudere versie van AFS maar kent geavanceerdere aspecten.

Het is beschikbaar onder de GNU GPL licentie en de huidige versie is 6.0.11 van 7 juni 2005.

Coda clients krijgen 1 gemount coda filesysteem. Deze hoeft dus niet verbonden te zijn aan een server maar dit kunnen er meerdere zijn. De client connect met coda en niet naar individuele servers. Dit is anders dan NFS, die wel mount met een server. Coda kent dus 1 mountpoint zodat alle clients hetzelfde geconfigureerd kunnen worden en dezelfde filetree zien. Dit maakt het makkelijker bij de installatie van meerdere systemen.

Om de werking van coda beter te begrijpen kijken we naar een file operation. Stel we voeren het commando: *cat /coda/tmp/niks* uit. Om te achterhalen wat de content van de file "niks" is. Er zullen een paar system calls worden gedaan in relatie tot de file. Een system call is een operatie die aan de kernel een bepaalde actie vraagt: Bijvoorbeeld het opzoeken van de inode van de file en returnen van een file handle. De open call wordt ontvangen door de Virutal File System (VFS) in de kernel. Deze beseft dat de open call voor een file bestemd is die zich op het coda file systeem bevindt. Hij geeft dit dan door aan de coda file system module in de kernel. Deze module houdt een cache bij van de recente requests van de VFS. Mocht de request niet eerder zijn gedaan dan wordt het request naar de coda cache manager , Venus, gestuurd. Venus zal de clients disk cache bekijken of de file *tmp/niks* hierin staat. Mocht de file er niet staan dan wordt een verbinding met Vice (de servers) via het netwerk opgezet en zal deze opgehaald worden. Als hij opgehaald is zal Venus de kernel vertellen waar hij te vinden is. De kernel zal het op zijn beurt weer aan het client programma doorgeven. Deze gang van zaken is schematisch weergegeven in figuur 7.



Figuur 7: Coda's Architectuur

Als Venus de file opgehaald heeft van Vice bewaart hij het in een container file in de cache (*/usr/coda/venus.cache*). Het is nu een file op de lokale disk en zal bij een read-write operatie niet door Venus afgehandeld worden maar bijna helemaal via het lokale file systeem. Dit zal dus met ongeveer dezelfde tijd gaan als bij een lokaal file systeem. Directory files en de file attributen worden allemaal gecached door Venus. Wanneer een file aangepast, verwijderd of gecreëerd wordt zal Venus de servers updaten en de veranderingen doorgeven.

Wanneer het netwerk en coda normaal werken zullen de updates naar de servers synchroon verlopen. Mocht er echter een netwerk link wegvallen of de server uitvallen kunnen deze updates niet gedaan worden. Venus zal bij een opgetreden fout tijdens het verzenden van een update niet aan de user melden dat dit fout is gegaan. In dit geval zal Venus de update loggen op de client in de Client Modification Log (CML). Wanneer dan de servers weer toegankelijk zijn zal de CML aangesproken worden en zal de update alsnog verzonden worden. De CML is geoptimaliseerd, het zal bijvoorbeeld geen updates sturen als deze alweer ongedaan zijn gemaakt in de tussentijd.

Er zijn twee aspecten belangrijk bij het wegvallen van het netwerk of de server:

Hoarding Dit is een lijst met welke files up to date moeten blijven. Venus zal de server vragen om de laatste updates te sturen aan de hand van deze files. Dit updaten zorgt ervoor dat de belangrijkste files up to date zijn zodat bij een disconnection de nieuwste versie van de files gebruikt kan worden. Het up-to-date houden van de Documenten directory is bijvoorbeeld erg handig als de gebruiker ook af en toe thuis werkt.

Local/Global conflict Dit conflict krijg je als een file op de server aangepast is (door een andere client) tijdens disconnection. Reparatie zal via applicatie specifieke oplossers moeten gebeuren. Soms zal het conflict door de beheerder opgelost moeten worden.

Coda exporteert niet een directory van een standaard file systeem structuur maar een volume. Partities op de coda server bezitten bestanden die gegroepeerd zijn in een volume. Een volume kent eenzelfde directory structuur als een file systeem, is kleiner dan een partitie maar groter dan een directory. Een voorbeeld van zo'n volume is bijvoorbeeld een home directory van een gebruiker. Op een server bestaan ongeveer honderd volumes.

Coda bewaart de volume en directory informatie, access control lists en file attributen in raw partities. Ze worden aangesproken via het Recoverable Virtual Memory (RVM) om snelheid en consistentie te waarborgen. Alleen de file data staat op de server partities. RVM zorgt er ook voor om bij een server crash het systeem te kunnen restoren.

Elk volume heeft een ID en een naam. Volumes kunnen overal in */coda* worden gemount. Er mag geen volume op een bestaande directory gemount worden. Het mountpoint is niet te zien door de client en is gewoon een directory onder */coda*.

Coda deelt een file in door drie 32bit integers die Fid (FileID) heten. Een Fid bestaat uit VolumeID, een VnodeID en een Uniquifier. Het VolumeID identificeert het volume waar het bestand zich in bevindt. Het VnodeID is zeg maar de "inode" en de Uniquifiers zijn nodig voor de resolutie. Resolutie wordt later in dit hoofdstuk uitgelegd. Een Fid is uniek in een cluster van Coda servers.

Coda heeft read/write replicatie servers, dit is een groep van servers die voor data uitwisseling met clients dienen. Updates worden door de client geïnitieerd en aan alle servers in deze groep gedaan. Dit vergroot de beschikbaarheid van data: Als 1 server faalt kan de andere hem overnemen zonder dat de client de fout ziet. Volumes worden opgeslagen in een groep van servers die de Volume Storage Group (VSG) heten. De VSG is een lijst van servers die een kopie van het replicated volume bezitten. Het repliceren van volumes wordt

gedaan door het VolumeID te repliceren. Dit VolumeID weet waar het lokale volume zich bevindt op de VSG. Dit lokale volume definieert een partitie en lokaal VolumeID die de bestanden en meta-data die op die server bestaan beschrijft.

Als Venus nu data op een van de servers wil verkrijgen dan heeft het eerst VolumeInfo nodig om het volume te vinden waarop de data staat. In VolumInfo staat een lijst van alle servers en de bij het volume behorende lokale VolumeIDs. De communicatie tussen de servers in een VSG is read-one, write-many. Dit betekent dat het bestand van een server in de VSG wordt gelezen en dan naar alle beschikbare VSG leden, de Available Volume Storage Group (AVSG) leden, wordt gestuurd.

Server replicatie kent eigenaardigheden zoals resolutie. Sommige servers in de VSG kunnen gescheiden worden van andere door bijvoorbeeld uitval van netwerk of server. Updates kunnen dan niet bij alle servers komen maar alleen bij de leden van de AVSG. Zo ontstaan er globale conflicten. Als data wordt opgevraagd wordt als eerste de versie van alle servers opgevraagd. Als 1 server niet de laatste kopie van de bestanden heeft zal het een resolutie proces starten dat automatisch deze verschillen oplost. Als dat niet lukt moet een gebruiker het manueel doen en alhoewel het door de gebruikers geïnitieerd moet worden, wordt het verder afgehandeld door de servers.

4 Inzetbaarheid

AFS en InterMezzo zijn bij het bekijken van de inzetbaarheid niet meegenomen omdat AFS een voorloper van Coda is en Coda dus gebaseerd is op AFS en vernieuwender is. Ook zal bij uitval van de server of netwerk het AFS file systeem niet beschikbaar zijn. InterMezzo wordt niet meegenomen in dit onderzoek omdat het op basis van het http protocol en perl scripts werkt. Verder wordt er ook niet meer aan ontwikkeld en is de laatste versie van oktober 2003.

Bij de inzetbaarheid van de DFS-en Lustre, GFS en Coda in het client-server scenario's moet de disconnected operation ondersteunt worden. Bij het server-server scenario is het van belang dat de data gesynchroniseerd word over minimaal 2 servers. Waarbij de synchronisatie door de server zelf geïnitieerd dient te worden.

4.1 GFS

GFS praat direct met de opslag devices over een channel netwerk in plaats van over een tcp/ip netwerk. Hierdoor is de performance verwachting hoog maar de inzetbaarheid lager omdat tegenwoordig het merendeel van de netwerken tcp/ip netwerken zijn. Echter er is een optie om de opslag devices te exporteren over het netwerk via Global Network Block Device (GNBD) of iSCSI. GNBD is een shared netwerk block protocol dat ervoor zorgt dat een lokaal opslag medium als een block device over een netwerk beschikbaar wordt. iSCSI staat voor Internet Small Computer System Interface, het zorgt ervoor dat SCSI commando's over een IP netwerk getransporteerd kunnen worden. Bij het gebruik van deze protocollen zal de performance verwachting wel lager worden. Het opdelen van verschillende opslag medium devices in gelijksoortige subpoolen verbeterd ook de performance. Nadeel is dat redundancy door de devices zelf geregeld dient te worden. Wel zorgt het locking mechanisme voor perfecte consistency en is er device caching in plaats van client caching dat de performance ten goede komt.

GFS kent geen disconnected operation waardoor een client, bij uitval van het netwerk, het opslag medium niet kan bereiken en data niet weggeschreven kan worden. GFS kent geen server-server model en zorgt alleen voor een koppeling met het data opslag medium. Hierdoor komt GFS niet aanmerking om gebruikt te worden in de 2 scenario's.

4.2 Lustre

Lustre kent geen single point of failure omdat gebruik wordt gemaakt van een failover metadata server en van gedistribueerde OSTs. Ook de splitsing van metadata en ruwe data waarbij metadata opgeslagen wordt op de MDS verminderd de tijd die het kost om het file systeem consistent te krijgen en levert dus een performance verbetering op. Nadeel van Lustre is dat bij uitval van een OST de data op deze OST niet meer beschikbaar is. Er kan echter nog wel data worden weggeschreven naar de andere OST's.

Lustre kent nog geen disconnected operation en valt dus af in het client-server scenario. De disconnected operation staat wel in de roadmap om in versie 3 uitgebracht te worden. Voor het server-server scenario kan lustre ook

niet worden ingezet aangezien het nog geen mirroring van de OST servers ondersteund. Aan deze optie wordt wel gewerkt maar zal waarschijnlijk pas in versie 2.0 van lustre worden geïntroduceerd. Dit word echter volgens de roadmap pas na het vierde kwartaal van 2005.

4.3 Coda

Coda kent client caching, hierdoor hoeft de client minder vaak data van de server af te halen. Er wordt gedaan aan replicatie zodat data opgeslagen of aangepast wordt op meerdere servers dit zorgt voor betrouwbaarheid en schaalbaarheid.

Coda is inzetbaar in het client-server scenario. Clients kunnen bij verlies van netwerkconnectiviteit gewoon doorwerken en data blijft consistent. Caching mechanismen zorgen ervoor dat een coda client kan doorwerken. Verspreiding van data over meerdere server zorgt voor een betere beschikbaarheid en schaalbaarheid. Coda is vooral inzetbaar in een LAN met meerdere clients.

In een server-server scenario kan Coda niet gebruikt worden. Data op een server wordt door clients gerepliceerd over andere servers die in een VSG zitten. De client initieert dus de overdracht en de replicatie van de data over de servers in het VSG. Het is niet mogelijk om de Coda server met een andere Coda server te laten synchroniseren. In een server-server scenario zullen servers verbonden zijn via een WAN verbinding. Doordat de client ervoor moet zorgen dat de data op de servers consistent moet zijn moet de data over deze WAN verbinding verstuurd worden. De client zal ervoor zorgen dat de data consistent is, hij initieert de synchronisatie. Er zal echter aan de client kant veel gecached moeten worden aangezien de WAN verbinding minder snel is. Met veel clients is dit niet bevorderlijk. Is er tijdelijk geen WAN verbinding beschikbaar en wil men dat de data op alle servers toch consistent is zal er (als de WAN verbinding terug is) een client moeten zijn die alle bestanden die veranderd zijn, opent en sluit om zo opnieuw de servers consistent te maken. De applicatie *rsync* met de optie - *-dry-run* zou hiervoor ingezet kunnen worden. Rsync is een applicatie die gebruikt kan worden om data te synchroniseren over verschillende file systemen. Deze optie is verre van optimaal en is niet aan te raden.

4.4 Andere manieren server-server scenario

Zoals gebleken is zijn GFS, Lustre en Coda niet inzetbaar voor een server-server scenario. Aangezien het vinden van een oplossing voor dit probleem ons toch heeft bezig gehouden is er toch nog gekeken naar andere methoden dan het inzetten van een DFS.

DRDB [3] (Distributed Replicated Block Device) is een soortement van RAID-1 oplossing over het netwerk. Het mirrored een block device over het netwerk. DRBD zorgt ervoor dat data die op de lokale harde schijf wordt weggeschreven ook naar en een andere host wordt verstuurd die dit daar weer op de lokale harde schijf zet. Nadeel van het gebruik van deze oplossing doet zich voor bij een uitval van de primaire server tijdens een synchronisatie met de slave server. De slave server kan dan de operaties van de master server niet overnemen omdat het inconsistente data heeft. Hierdoor kan niemand meer bij de data

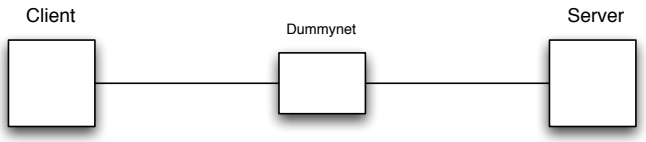
en moet er handmatig aangegeven worden welke data op welke server er het beste gebruikt kan worden.

Rsync [8] kan ook gebruikt worden om data tussen 2 servers consistent te houden. Het gebruikt het rsync remote-update protocol om data uit te wisselen. Dit protocol zorgt ervoor dat rsync incrementele synchronisaties kan uitvoeren. Het gebruikt hiervoor een efficiënt checksum-search algoritme. Deze oplossing heeft dus wel als nadeel dat de data niet automatisch wordt gesynchroniseerd. Dit kan echter wel geautomatiseerd worden.

Dit onderzoek richt zich alleen op DFS-en die voor Linux/BSD beschikbaar zijn. Er is echter voor Microsoft Windows een product die inzetbaar kan zijn in een server-server scenario en dit is Microsoft DFS in samenwerking met FRS. FRS repliceerd bestanden en folders, van door DFS gedeelde mappen, naar andere servers. Ook kent FRS een mechanisme om bestand en folder conflicten op te lossen en zo data tussen servers consistent houdt.

5 Proof of concept

In dit hoofdstuk wordt het Coda DFS systeem ingezet in een client-server scenario. Echter deze Proof of Concept is opgezet om de performance van het DFS in een client-server scenario te meten. Er kunnen dus ook andere DFS-en ingezet worden om zo een goede vergelijking te geven. Gezien het tijdsbestek is er gekozen om alleen Coda te testen. De performance wordt getest door de verbinding tussen de client en de server te variëren in bandbreedte en latency. Om het Coda DFS te testen wordt er gebruik gemaakt van drie machines.



Figuur 8: Coda testopstelling

Op twee machines zal daadwerkelijk Coda geïnstalleerd worden. Op de derde machine zal Dummynet[4] gebruikt worden om de gewenste bandbreedte en latency's in te stellen. Voor de benchmark zal er op de client bijgehouden worden hoe lang het duurt voordat een bestand verstuurd is. Op de Dummynet computer zal er ook worden bijgehouden hoe lang het duurt voordat een bestand verstuurd is. Dit wordt gedaan omdat de client bij trage verbindingen aan caching kan gaan doen. Om er voor te zorgen dat de tests niet te lang duren is er gekozen voor een bestandsgrootte van 5 megabyte. In dit hoofdstuk zal de configuratie van de systemen, de instellingen van de programmatuur en de benchmark methoden worden beschreven zodat achteraf de benchmarks herhaald kunnen worden.

5.1 Configuratie

Voor het benchmarken van Coda is er gebruik gemaakt van drie identieke machines met de volgende specificaties:

Processor Intel Pentium III 1Ghz

Geheugen 256 MB Sdram 133 mhz

Netwerkkaart 3Com 3c905C-TX Fast Etherlink XL 10/100Mbit

Hardeschijf Seagate U Series 6 20.4GB - 5400 rpm

De machines worden door middel van crosslink kabels aan elkaar verbonden. In de Dummynet computer zitten twee netwerkkaarten die in bridging mode staan om zo de bandbreedte te kunnen limiteren.

Als besturingssysteem is er voor FreeBSD 5.4 gekozen omdat op dit besturingssysteem de tests gemakkelijk herhaald kunnen worden voor een vervolg onderzoek. Op alle systemen is ports compatibility aangezet. Hieronder komt een opsomming van de gebruikte software en instellingen van de drie systemen:

5.1.1 Dummynet instellingen

De kernel van het Dummynet systeem is gecompileerd met de volgende toevoegingen:

```
options BRIDGE
options IPFIREWALL
options DUMMYNET
options HZ=10000
```

Dit zorgt voor de ondersteuning van Bridge, Firewall en Dummynet
Om de bridge te activeren moet er in /etc/sysctl.conf het volgende worden toegevoegd:

```
net.link.ether.bridge.enable=1
# Netwerkkaarten waar de bridge op actief moet zijn
net.link.ether.bridge.config=xl0,xl1
net.link.ether.bridge.ipfw=1
net.link.ether.ipfw=1
net.link.ip.fw.enable=1
```

De dummynet bridge is als volgt ingesteld:

```
# flushen van oude rules en pipes
ipfw -f
ipfw pipe f

# toevoegen van pipes
ipfw add 100 allow layer2 not mac-type ip
ipfw add 1000 pipe 1 ip from 192.168.1.1 to 192.168.1.2
ipfw add 1010 pipe 2 ip from 192.168.1.2 to 192.168.1.1
ipfw add 65000 deny ip from any to any
```

Dit zorgt voor het leggen van een pipe tussen de client met ip adres 192.168.1.2 en de server met ip adres 192.168.1.1. Op deze pipe kan dan later aanpassingen gedaan worden in bandbreedte en latency.

5.1.2 Coda Client

De kernel van de Coda client is gecompileerd met de volgende twee opties:

```
options     CODA
device      vcoda   4
```

Hierna is via de ports collectie Coda Client 6.0.7 geïnstalleerd. De coda client applicatie wordt gestart met een cache van 20Mb.

5.1.3 Coda Server

De kernel van de Coda server is met dezelfde opties gecompileerd als die van de Coda client. Verder is Coda Server 6.0.7 uit de ports collection geïnstalleerd. De coda server heeft een rvm log size van 20M, de rvm data size wordt ingesteld op 500Mb en het maximaal aantal bestanden per Vice volume wordt op 1 miljoen gezet. Deze instellingen zijn standaard instellingen die in een normaal bezet LAN van toepassing kunnen zijn. Deze instellingen kunnen aangepast worden om de performance van de server fine te tunen.

5.2 Benchmark instellingen

Voor het uitvoeren van de benchmark tests is het Dummynet systeem op de volgende manier ingesteld:

```
# instellen pipes
ipfw pipe 1 config bw <bandbreedte> delay <vertraging in ms>
ipfw pipe 2 config bw <bandbreedte> delay <vertraging in ms>
```

De eerste pipe wordt gebruikt voor het verkeer van de server naar de client. Pipe 2 wordt gebruikt voor het verkeer van de client naar de server. Om een Round Trip Time te krijgen van 200 ms moet er op beide pipes 100 ms als vertraging ingevuld worden.
Voor de test zijn de volgende instellingen gebruikt:

Dummynet Systeem Om de tijd te registreren die de client nodig heeft om de data naar de server te sturen wordt gebruik gemaakt van tcpdump. Op het Dummynet systeem wordt tcpdump gestart en zal na het einde van het dataverkeer de begintijd en de eindtijd bekend zijn. Het volgende commando is gebruikt om deze tijden vast te leggen:

```
tcpdump -l >/tcpdump.txt | tail -f /tcpdump.txt
```

Client Systeem Op de Client wordt gebruikt gemaakt van de applicaties; *dd* en *time*. Dd is een tool dat de standaard input naar de standaard output schrijft. Het wordt hier gebruikt om een file van 5 Mb te genereren. *Time* wordt gebruikt om de tijd op de client vast te leggen. Het geeft dus precies de tijd weer die de gebruiker ervaart op het Client systeem bij het wegschrijven van een bestand. *Sync* wordt ook nog gebruikt om alle pending disk writes te flushen en zo dus de tellers op nul te zetten. De volgende commando's worden uitgevoerd:

```
sync
time -h -o /ddtimes -a dd if=/dev/zero
      of=/coda/bestand bs=1M count=5
rm /coda/bestand
```

Sync flushed de cache. time -h start time met gegevens op 1 regel. De optie -o exporteerd de output naar een bestand, in dit geval /ddtimes. -a

voegt de resultaten van dd toe aan het bestand *ddtimes*. Dd wordt gestart met de opties *if* waarin aangegeven wordt dat /dev/zero moet gebruikt worden. Dit zorgt voor een input van allemaal nullen. De optie *of* geeft aan waar het bestand weggeschreven moet worden. In dit geval is dat op het coda filesysteem in een bestand met de naam "bestand". bs geeft de Block Size aan die 1 Mb is en die wordt 5 keer herhaald, count=5 geeft dit aan. Het bestand wordt na de meting ook weer verwijderd, met behulp van rm, om eventuele caching te flushen.

Server Systeem Op dit systeem dienen geen instellingen gezet te worden om benchmarking mogelijk te maken.

5.3 Resultaten

Er zijn testen gedaan met een round trip time van 400 ms. Hieruit kwamen echter resultaten uit waarbij de totale tijd voor het versturen onder de tijd lag van de 200 ms round trip time. Een verklaring was hier voor niet te vinden. Om deze reden zijn deze resultaten ook niet toegevoegd aan de resultaten.

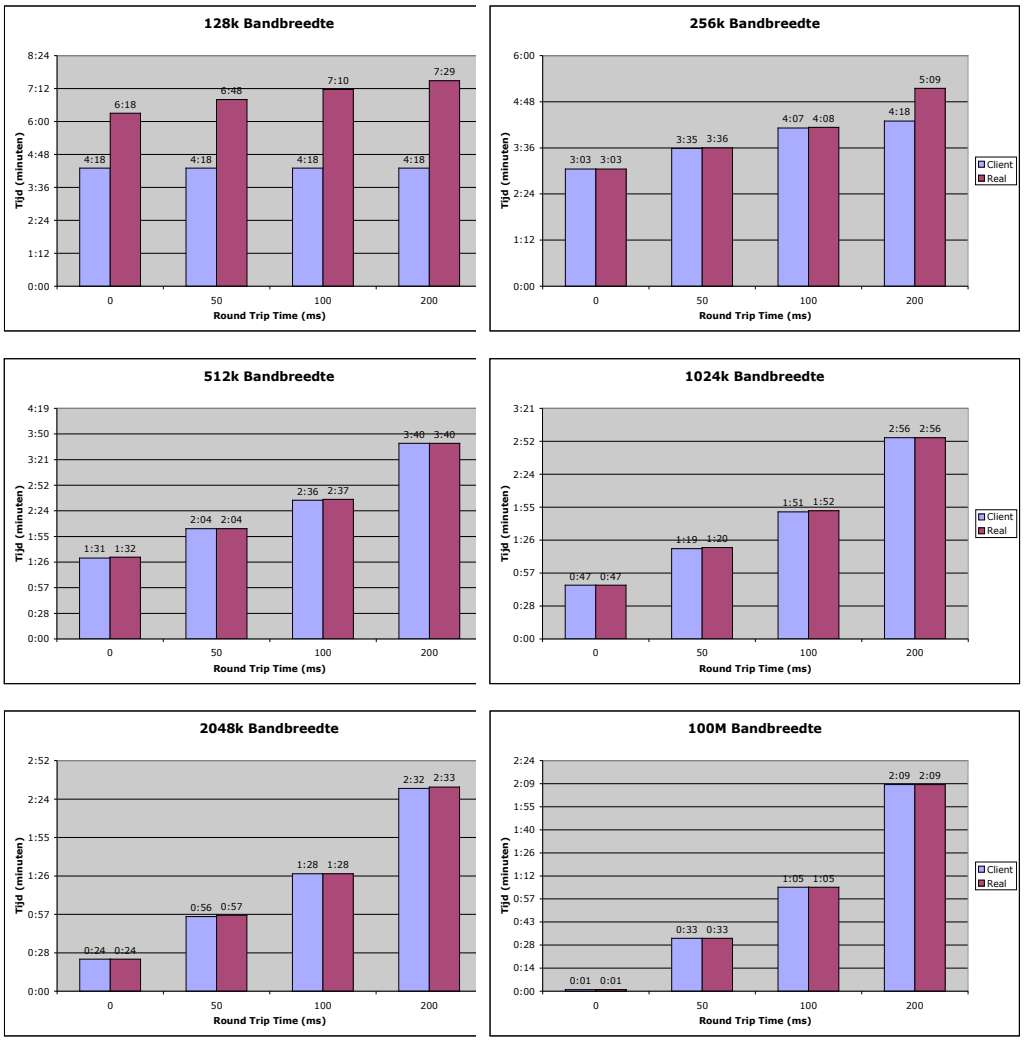
Aangezien de latency van het LAN (0 ms) afhankelijk is van de gekozen bandbreedte staat hieronder een tabel met daarin de latency's van de verbinding op de verschillende bandbreedtes.

Bandbreedte	Latency bij 0 ms instelling
128k	12.5 ms
256k	6.25 ms
512k	3.2 ms
1024	1.6 ms
2048	0.8 ms
100M	0.3 ms

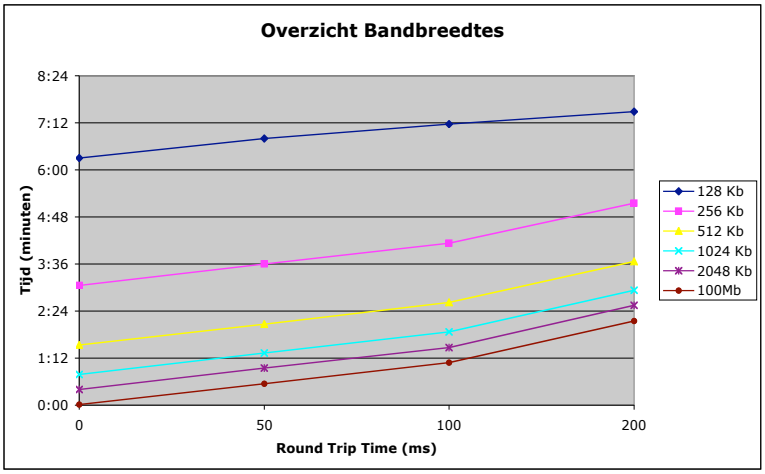
In figuur 9 is te zien dat bij een hogere latency het langer duurt om een bestand te verzenden. Hogere bandbreedtes zorgen ervoor dat bestanden sneller verstuurd worden. Bij de testen op een bandbreedte van 128kbps is duidelijk te zien dat, aan de client kant, caching wordt toegepast. De grens van het direct versturen van data ligt bij 4 minuten en 18 seconden. Lukt het de client niet om het bestand binnen deze tijd te versturen dan zal de rest van de data worden gecached en op de achtergrond worden verstuurd, dit is te zien als real tijd. Bij 256k zien we dat er alleen nog bij de 200 ms round trip time caching optreed. Bij de test hoger dan 512kbs wordt het bestand direct naar de server verstuurd, dit is te zien aan de gelijke tijden. Het totaal overzicht van alle bandbreedtes en latency's is in figuur 10 afgebeeld.

5.4 Analyse resultaten / discussie

Bij nadere bestudering van de resultaten is het duidelijk dat Coda het bestand gaat cachen als het te lang duurt om het bestand te versturen. De grens van dit alles is rond de 4 minuten en 18 seconden. In het totaal overzicht is te zien dat een hoge latency gevolgen heeft voor de gehele performance. Een verbinding met een lagere snelheid en latency kan sneller zijn dan een hogere bandbreedte



Figuur 9: Performance metingen mbt bandbreedte en latency



Figuur 10: Overzicht performance metingen

met een hogere latency. Tijdens het testen kwamen er wel een aantal tekortkomingen aan het licht. Zo gebeurde het nog wel eens dat een bestand lokaal op de client gecached werd en niet meteen werd verzonden naar de server. De client probeerde ook geen verbinding te maken met de Coda server, dit moest handmatig worden geïnitialiseerd. Coda kent verschillende connecties met de server. De client kan write-disconnected of connected zijn. In het bovenstaande geval gaat de client dus van een connected state in write-disconnected state en zal de data gechached worden in plaats van direct verstuurd. De gebruiker op de client merkt hier echter niets van maar kan er dus niet van uit gaan dat na het uitvoeren van bijvoorbeeld een kopieer actie, de data daadwerkelijk op dat moment ook op de server staat.

6 Conclusie

Na het onderzoeken van verschillende Distributed File Systems in een client-server en server-server scenario zijn er een aantal belangrijke punten naar voren gekomen. Zo is het voor het client-server scenario mogelijk om met een Coda server en client te werken om zo bestanden gemakkelijk en eenvoudig mee te nemen. Er kan worden aangegeven welke bestanden er op de client gecached moeten worden. Voor een server-server omgeving is Coda helaas niet geschikt. De client zorgt namelijk voor de consistentie van de data op de server.

Met GFS is het mogelijk om schijf arrays aan te spreken. Redundantie wordt bereikt door de schijven in raid te zetten (raid 0 uitgesloten). Nadeel van dit systeem is dat als de hoofdserver uitvalt er geen verbinding meer gemaakt kan worden met de arrays. Er is nog geen mogelijkheid om meerdere servers dezelfde data te laten opslaan.

Gezien de raodmap van Lustre beloofd dit project al dat wat verwacht wordt van een DFS. Het zou het mogelijk maken om servers redundant uit te voeren, waarbij de servers onderling de consistentie bewaken. Clients zouden data kunnen cachen als er geen verbinding is met de server. Er komt support aan voor Mac OS X en Microsoft Windows Clients zodat het op een heel groot gebied ingezet kan worden. Nadeel van dit Lustre is dat het erg langzaam wordt ontwikkeld en dat het nog een tijd kan duren voordat deze features ook daadwerkelijk zijn geïmplementeerd.

Op dit moment is er nog geen DFS systeem dat gebruikt kan worden in een server-server scenario. Een project als DRBD heeft de potentie om hier voor ingezet te worden mits de failover procedure goed gerealiseerd wordt.

Referenties

[1] Andrew File System http://www.psc.edu/general/filesys/afs/afs.html

[2] Coda File System http://www.coda.cs.cmu.edu/

[3] DRBD - Distributed Replicated Block Device http://www.drdb.org

[4] Dummynet - simulates/enforces queue and bandwidth limitations, delays, packet losses, and multipath effects. http://info.iet.unipi.it/̃luigi/ip_dummynet/

[5] Global File System - A File System for Shared Disk Storage http://www.diku.dk/undervisning/2003e/314/papers/soltis97global.pdf

[6] GNU General Public License (GPL). http://www.gnu.org/copyleft/gpl.html

[7] InterMezzo http://www.inter-mezzo.org/

[8] Rsync http://samba.anu.edu.au/rsync/

[9] Lustre http://www.clusterfs.com/

[10] Wikipedia.org http://www.wikipedia.org