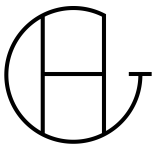


HoGent

Faculteit Bedrijf en Organisatie

ZFS met RAID-Z als alternatief voor klassieke RAID-oplossingen

Jonas De Moor

Scriptie voorgedragen tot het bekomen van de graad van
professionele bachelor in de toegepaste informatica

Promotor:
Antonia Pierreux
Co-promotor:
Karine Van Driessche

Instelling: —

Academiejaar: 2016-2017

Tweede examenperiode

Faculteit Bedrijf en Organisatie

ZFS met RAID-Z als alternatief voor klassieke RAID-oplossingen

Jonas De Moor

Scriptie voorgedragen tot het bekomen van de graad van
professionele bachelor in de toegepaste informatica

Promotor:
Antonia Pierreux
Co-promotor:
Karine Van Driessche

Instelling: —

Academiejaar: 2016-2017

Tweede examenperiode

Samenvatting

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetuer id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.

Nam dui ligula, fringilla a, euismod sodales, sollicitudin vel, wisi. Morbi auctor lorem non justo. Nam lacus libero, pretium at, lobortis vitae, ultricies et, tellus. Donec aliquet, tortor sed accumsan bibendum, erat ligula aliquet magna, vitae ornare odio metus a mi. Morbi ac orci et nisl hendrerit mollis. Suspendisse ut massa. Cras nec ante. Pellentesque a nulla. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Aliquam tincidunt urna. Nulla ullamcorper vestibulum turpis. Pellentesque cursus luctus mauris.

Nulla malesuada porttitor diam. Donec felis erat, congue non, volutpat at, tincidunt tristique, libero. Vivamus viverra fermentum felis. Donec nonummy pellentesque ante. Phasellus adipiscing semper elit. Proin fermentum massa ac quam. Sed diam turpis, molestie vitae, placerat a, molestie nec, leo. Maecenas lacinia. Nam ipsum ligula, eleifend at, accumsan nec, suscipit a, ipsum. Morbi blandit ligula feugiat magna. Nunc eleifend consequat lorem. Sed lacinia nulla vitae enim. Pellentesque tincidunt purus vel magna. Integer non enim. Praesent euismod nunc eu purus. Donec bibendum quam in tellus.

Nullam cursus pulvinar lectus. Donec et mi. Nam vulputate metus eu enim. Vestibulum pellentesque felis eu massa.

Quisque ullamcorper placerat ipsum. Cras nibh. Morbi vel justo vitae lacus tincidunt ultrices. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. In hac habitasse platea dictumst. Integer tempus convallis augue. Etiam facilisis. Nunc elementum fermentum wisi. Aenean placerat. Ut imperdiet, enim sed gravida sollicitudin, felis odio placerat quam, ac pulvinar elit purus eget enim. Nunc vitae tortor. Proin tempus nibh sit amet nisl. Vivamus quis tortor vitae risus porta vehicula.

Voorwoord

Deze bachelorproef duidt het einde aan van mijn opleiding Toegepaste Informatica. Met deze bachelorproef wil ik bewijzen dat ik op een zelfstandige en objectieve manier onderzoek kan voeren over een (nieuwe) technologie in de IT-wereld. Omdat onze branche vrijwel continu onderhevig is aan verandering, vind ik dit een niet-onbelangrijke competentie.

Als onderwerp van deze bachelorproef heb ik besloten om een al wat oudere (maar zeker niet oninteressante) technologie te bespreken: het ZFS bestandssysteem. De redenen waarom ik net dit onderwerp zou willen bespreken, zijn nogal uiteenlopend. Ik ben zelf een enorme Linux- en UNIX-fan en ik hou ervan om mezelf nieuwe dingen aan te leren. Met ZFS was ik nog niet vertrouwd, en daar ik toch van plan ben om zelf een homeserver samen te stellen en als bestandssysteem ZFS te gebruiken, leek deze bachelorproef mij een uitstekende opportuniteit om mij wat meer te verdiepen in de werking en implementatie van deze technologie. Ook hoorde ik hier en daar geruchten vallen over de mogelijke onbetrouwbaarheid van RAID (het zgn.RAID 5 "write hole") en dit was dan ook een reden om onderzoek te voeren naar een mogelijk alternatief.

Deze bachelorproef is het resultaat van vele uren noeste arbeid; het is een werk waar ik enorm trots op ben. Desalniettemin zou dit werk niet mogelijk zijn geweest zonder de hulp van een aantal mensen. Graag neem ik daarom even de tijd om enkele personen te bedanken voor hun steun en toeverlaat gedurende deze periode.

Eerst en vooral zou ik mijn promotor, mevr. Antonia Pierreux, en mijn co-promotor, mevr. Karine Van Driessche, willen bedanken voor het goed laten verlopen van deze periode. Het schrijven van deze scriptie en het voeren van een onderzoek waren geen makkelijke karweien; zonder hun inhoudelijke en technische steun zou deze bachelorproef

nooit mogelijk geweest zijn. Tevens zou ik ook nog graag mijn familie en vrienden willen bedanken voor de onophoudelijke steun en begrip. Ook wil ik graag de mensen van DViT bedanken voor het aanreiken van technische kennis en materiaal voor mijn onderzoek. En *last but not least* wil ik mijn ouders enorm bedanken om mij gedurende deze drie jaar ten volle te steunen: dankzij hen heb ik telkens opnieuw de moed teruggevonden om er met volle teugen tegenaan te gaan in perioden dat het wat moeilijker ging.

Ik hoop dat u evenveel plezier beleeft met het lezen van mijn scriptie als ik had met het schrijven ervan.

Jonas De Moor Academiejaar 2016-2017

Inhoudsopgave

1. Inleiding

1.1 Probleemstelling en Onderzoeksvragen

Aangezien (een deel van) de RAID-functionaliteit van BTRFS nog niet als *production-ready* wordt beschouwd (The BTRFS Project, 2014, maart 4), zal ZFS worden beschouwd en geëvalueerd als volwaardig alternatief voor klassieke RAID-oplossingen op Linux-systemen.

Deze bachelorproef zal een antwoord vinden op volgende vragen:

- Wat zijn de grootste verschillen tussen een klassieke RAID-oplossing en ZFS RAID-Z?
- Hoe is de architectuur van ZFS opgebouwd en op welke manieren tracht het oplossingen te vinden voor de problemen die zich voordoen bij andere bestandssystemen en RAID-opstellingen?
- Hoe staat het met data-integriteit en performantie[1] bij ZFS onder verschillende workloads en toepassingen?

1.2 Opzet van deze bachelorproef

De rest van deze bachelorproef is als volgt opgebouwd:

In Hoofdstuk 2 wordt de methodologie toegelicht en worden de gebruikte onderzoekstechnieken besproken om een antwoord te formuleren op de onderzoeksvragen.

[1] Met 'performantie' wordt het aantal I/O's per seconde en de globale schijf- en CPU-belasting bedoeld.

In Hoofdstuk 3 wordt er een korte inleiding gegeven op de geschiedenis en de algemene werking van RAID-systemen. Ook ZFS en RAID-Z worden reeds kort toegelicht.

In Hoofdstuk 4 wordt er een globaal overzicht gegeven van de architectuur en ontwerpprincipes van ZFS. In de daaropvolgende hoofdstukken worden de belangrijkste onderdelen en functionaliteiten wat meer uitgediept.

In Hoofdstuk 5 wordt het opslagmodel van het ZFS-bestandssysteem besproken. Onder andere de datastructuur en het transactiemodel van ZFS komen aan bod.

In Hoofdstuk 6, tenslotte, wordt de conclusie gegeven en een antwoord geformuleerd op de onderzoeksvragen. Daarbij wordt ook een aanzet gegeven voor toekomstig onderzoek binnen dit domein.

2. Methodologie

Etiam pede massa, dapibus vitae, rhoncus in, placerat posuere, odio. Vestibulum luctus commodo lacus. Morbi lacus dui, tempor sed, euismod eget, condimentum at, tortor. Phasellus aliquet odio ac lacus tempor faucibus. Praesent sed sem. Praesent iaculis. Cras rhoncus tellus sed justo ullamcorper sagittis. Donec quis orci. Sed ut tortor quis tellus euismod tincidunt. Suspendisse congue nisl eu elit. Aliquam tortor diam, tempus id, tristique eget, sodales vel, nulla. Praesent tellus mi, condimentum sed, viverra at, consectetuer quis, lectus. In auctor vehicula orci. Sed pede sapien, euismod in, suscipit in, pharetra placerat, metus. Vivamus commodo dui non odio. Donec et felis.

Etiam suscipit aliquam arcu. Aliquam sit amet est ac purus bibendum congue. Sed in eros. Morbi non orci. Pellentesque mattis lacinia elit. Fusce molestie velit in ligula. Nullam et orci vitae nibh vulputate auctor. Aliquam eget purus. Nulla auctor wisi sed ipsum. Morbi porttitor tellus ac enim. Fusce ornare. Proin ipsum enim, tincidunt in, ornare venenatis, molestie a, augue. Donec vel pede in lacus sagittis porta. Sed hendrerit ipsum quis nisl. Suspendisse quis massa ac nibh pretium cursus. Sed sodales. Nam eu neque quis pede dignissim ornare. Maecenas eu purus ac urna tincidunt congue.

Donec et nisl id sapien blandit mattis. Aenean dictum odio sit amet risus. Morbi purus. Nulla a est sit amet purus venenatis iaculis. Vivamus viverra purus vel magna. Donec in justo sed odio malesuada dapibus. Nunc ultrices aliquam nunc. Vivamus facilisis pellentesque velit. Nulla nunc velit, vulputate dapibus, vulputate id, mattis ac, justo. Nam mattis elit dapibus purus. Quisque enim risus, congue non, elementum ut, mattis quis, sem. Quisque elit.

Maecenas non massa. Vestibulum pharetra nulla at lorem. Duis quis quam id lacus dapibus interdum. Nulla lorem. Donec ut ante quis dolor bibendum condimentum. Etiam egestas

tortor vitae lacus. Praesent cursus. Mauris bibendum pede at elit. Morbi et felis a lectus interdum facilisis. Sed suscipit gravida turpis. Nulla at lectus. Vestibulum ante ipsum primis in faucibus orci luctus et ultrices posuere cubilia Curae; Praesent nonummy luctus nibh. Proin turpis nunc, congue eu, egestas ut, fringilla at, tellus. In hac habitasse platea dictumst.

Vivamus eu tellus sed tellus consequat suscipit. Nam orci orci, malesuada id, gravida nec, ultricies vitae, erat. Donec risus turpis, luctus sit amet, interdum quis, porta sed, ipsum. Suspendisse condimentum, tortor at egestas posuere, neque metus tempor orci, et tincidunt urna nunc a purus. Sed facilisis blandit tellus. Nunc risus sem, suscipit nec, eleifend quis, cursus quis, libero. Curabitur et dolor. Sed vitae sem. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Maecenas ante. Duis ullamcorper enim. Donec tristique enim eu leo. Nullam molestie elit eu dolor. Nullam bibendum, turpis vitae tristique gravida, quam sapien tempor lectus, quis pretium tellus purus ac quam. Nulla facilisi.

3. Inleiding tot RAID & ZFS

In dit hoofdstuk wordt er een korte inleiding gegeven over de basisprincipes van RAID, aangezien RAID-Z een softwarematige vorm van RAID is. Daarnaast wordt de geschiedenis en globale werking van ZFS reeds kort besproken.

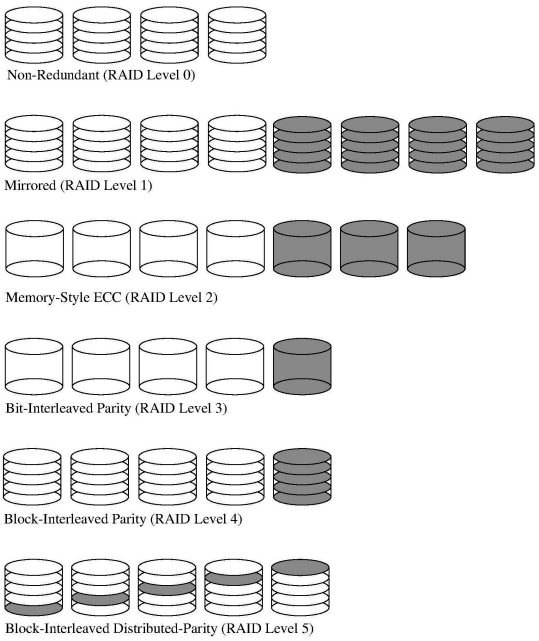
3.1 Basisbeginselen van RAID

3.1.1 Geschiedenis

Al van oudscher worden magnetische harde schijven gebruikt als opslagmedium voor data (Goda & Kitsuregawa, 2012). Maar reeds in de jaren 80 zagen onderzoekers in dat I/O-performance een bottleneck zou vormen voor computersystemen in de toekomst: terwijl geheugenchips en processoren steeds sneller werden, bevonden opslagmedia zich in een impasse (A. Paterson, 1987).

In de paper *"A Case for Redundant Arrays of Inexpensive Disks"* formuleerden A. Paterson (1987) en zijn collega's voor het eerst de term 'RAID', wat een acroniem is voor 'Redundant Arrays of Inexpensive Disks'. De oorspronkelijke idee achter RAID was dat een verzameling van goedkopere schijven performanter zou zijn dan grotere en duurdere mainframeschijven van die tijd.

Naast performantie en kost was betrouwbaarheid (reliability) ook een belangrijke factor. Als bv. één of meerdere schijven van de array falen, dan mag dit geen invloed hebben op de werking van de rest van de verzameling schijven. Daarom introduceerden de onderzoekers de zgn. "RAID levels"(A. Paterson, 1987), die vandaag de dag nog steeds in gebruik zijn. Er bestaan een aantal RAID-niveaus, waarvan de voornaamste zullen besproken worden.



Figuur 3.1: Illustratie van verschillende RAID-niveaus (M. Chen e.a., 1994)

3.1.2 Eigenschappen van RAID-systemen

Bij het bouwen van RAID-systemen worden er gebruikelijk drie aspecten in beschouwing genomen: **performantie** (performance), **betrouwbaarheid** (reliability) en **capaciteit** (capacity). Een RAID-niveau is in principe niets anders dan een balans tussen deze verschillende eigenschappen; meestal zullen er dus één of meerdere trade-offs moeten gemaakt worden (M. Chen e.a., 1994).

Begrippen die centraal staan bij RAID zijn ***striping*** en ***parity***. Striping heeft betrekking op de manier waarop een RAID-controller (hetzij hardwarematig, hetzij softwarematig) de blokken data verdeelt over de array van schijven. Bij RAID 0 bijvoorbeeld wordt de data gelijkmatig gedistribueerd volgens het *round-robin*-algoritme (Arpaci-Dusseau & Arpaci-Dusseau, 2015). Datablokken die verdeeld zijn over meerdere schijven en samen één geheel vormen worden een *stripe* genoemd.

3.1.3 RAID-niveaus: een overzicht

RAID 0

Een voordeel bij RAID 0 is dat de gehele capaciteit van de schijven kan gebruikt worden; er gaat geen ruimte verloren, aangezien de data gelijkmatig verdeeld wordt over de array. Een bijkomend voordeel van striping is dat performantie in het algemeen goed is (Arpaci-Dusseau & Arpaci-Dusseau, 2015) : de meeste en reads en writes kunnen parallel worden afgehandeld. Een voorwaarde voor goede performantie is echter wel dat de *chunk size* (de

grootte van de blokken data die worden weggeschreven en/of uitgelezen) ook optimaal wordt gekozen, i.e. afhankelijk van de workload op het systeem (Arpaci-Dusseau & Arpaci-Dusseau, 2015). Toch kent RAID 0 een groot nadeel: betrouwbaarheid is nagenoeg onbestaande. Aangezien data nergens wordt gedupliceerd, betekent dat het falen van eender welke disk leidt tot verlies van data (Arpaci-Dusseau & Arpaci-Dusseau, 2015).

RAID 4

Parity is een mechanisme dat werd geïntroduceerd bij RAID-niveau 4 om betrouwbaarheid af te dwingen. Bij RAID 4 wordt er metadata over de opgeslagen data bijgehouden in parity blocks op een aparte schijf. Deze metadata wordt verkregen d.m.v. het uitvoeren van een mathematische functie op de opgeslagen data. Meestal is dit een XOR-functie (exclusieve OF) (M. Chen e.a., 1994). Aan de hand van deze parity kan bij het verlies van één of meerdere schijven de originele data worden gereconstrueerd door XOR'ing toe te passen op de parity bits en de data bits. Bij een XOR-operatie geven een even aantal enen (1) steeds als resultaat nul (0); omgekeerd geldt ook dat een oneven aantal enen (1) steeds een één (1) als resultaat zullen opleveren. Stel dat één schijf van een array van vier schijven faalt, dan kan nog steeds de originele data worden verkregen. Echter, als er meer dan één schijf verloren gaat, dan is het bij RAID 4 onmogelijk om de originele data te herstellen. Het grote voordeel van RAID 4 is dan weer echter dat er minder wordt ingeboet op capaciteit dan bij bv. RAID 1 en RAID 5 (Arpaci-Dusseau & Arpaci-Dusseau, 2015).

RAID 1

Naast RAID 0 en RAID 4 zijn er nog andere RAID-levels, zoals RAID 1 en RAID 5. RAID 1 staat ook bekend als *mirroring*, omdat het kopieën maakt van de datablokken naar één of meerdere disks afhankelijk van het aantal schijven. Op gebied van capaciteit is RAID 1 niet echt gunstig, aangezien maar de helft van de totale schijfruimte bruikbaar is. Stel dat er vier schijven in een array aanwezig zijn, dan is slechts de opslagcapaciteit van twee schijven bruikbaar. Daarentegen is de betrouwbaarheid van RAID 1 wel vele malen beter dan die van RAID 0: in theorie mogen er bij een reeks van n schijven $\frac{n}{2}$ schijven falen. Maar dan mogen de schijven die elkaars mirror zijn niet falen, want dan is de data op deze disks verloren. Daarom houdt men in de praktijk meestal de maatstaf van één schijf aan (Arpaci-Dusseau & Arpaci-Dusseau, 2015).

RAID 5

Als laatste wordt RAID 5 besproken. RAID 5 is in principe niets anders dan RAID-niveau 4, maar dan uitgebreid met functionaliteit dat de parity blocks roteert over de verschillende schijven. Dit is een groot verschil t.o.v. RAID 4, waarbij de parity blocks zich op één disk bevinden. Read-performantie is nagenoeg gelijk aan RAID 4, maar write-performantie is stukken beter. Dit komt omdat bij RAID 5 de schrijfoperaties parallel kunnen worden afgehandeld; bij RAID 4 vormt de parity-schijf een *bottleneck* bij het wegschrijven van data (M. Chen e.a., 1994). De reden hiervoor is dat bij het updaten van data ook de parity blocks moeten worden geüpdatet; alle operaties worden dus m.a.w. serieel uitgevoerd

(Arpaci-Dusseau & Arpaci-Dusseau, 2015).

Er bestaan nog andere, niet-standaard RAID levels, zoals RAID 6 en RAID 10, maar deze worden hier niet besproken.

3.2 Inleiding tot ZFS & RAID-Z

3.2.1 Geschiedenis

ZFS is een bestandssysteem dat ontwikkeld is door het toenmalige Sun Microsystems, nu onderdeel van Oracle Corporation. Voormalig Sun-werknemer Jeff Bonwick was de oorspronkelijke hoofdontwikkelaar van het bestandssysteem. De ontwikkeling van ZFS startte aan het begin van de jaren 2000; Sun had reeds met de ontwikkeling van bestandssystemen geëxperimenteerd, maar deze pogingen mislukten telkens (Bonwick e.a., 2015). ZFS is een *copy-on-write* (COW) bestandssysteem (Hickmann & Shook, 2007): bij elke aanpassing van een datablok, wordt het datablok in kwestie niet aangepast, maar wordt het gekopieerd naar een nieuwe locatie en aangepast (Lucas & Jude, 2015).

In die tijd gebruikte het bedrijf haar UNIX-besturingssysteem Solaris intern voor verschillende soorten toepassingen, waaronder file servers (Bonwick e.a., 2015). De schijven van deze servers waren opgedeeld in volumes, beheerd door de Solaris Volume Manager (SVM) en geformatteerd in UFS (UNIX Filesystem) (Bonwick e.a., 2015). Toen een verkeerd ingevoerd SVM-commando erin slaagde om het systeem te doen crashen en voor een enorme *downtime* zorgde bij Sun, was dit voor Bonwick een aanleiding om volledig *from scratch* een bestandssysteem op te bouwen dat makkelijk in gebruik én beheer was.

De hoofdreden om een volledig nieuw bestandssysteem te ontwikkelen was volgens Bonwick en Moore (Unknown) dat de toenmalige oplossingen voor bestandssystemen totaal achterhaald waren. Bestandssystemen waren nog ontwikkeld voor opslagnoden uit de jaren ’80 en ’90, welke niet te vergelijken zijn met de huidige noden van zowel bedrijven als particulieren. Naarmate de nood aan meer opslagruimte steeg, moesten er oplossingen worden bedacht, en dit m.b.v. bestandssystemen die hier helemaal niet op voorzien waren. Een ’tussenoplossing’, aldus volgens Bonwick en Moore (Unknown), die ook vandaag de dag nog gebruikt wordt, is de combinatie van volumes en bestandssystemen. Volumes zijn abstracties voor (delen van) fysieke schijven.

3.2.2 RAID-Z

Het ZFS-bestandssysteem omvat ook een softwarematige RAID, RAID-Z genaamd. Volgens Bonwick (2005) lost RAID-Z in combinatie met ZFS een aantal problemen op die inherent aanwezig zijn bij andere RAID-implementaties. Zo claimen de ontwikkelaars dat RAID-Z het zogenaamde RAID5 *write hole* probleem volledig oplost. Bij klassieke RAID-oplossingen worden stripes weggeschreven op een niet-atomaire manier, i.e. de bewerkingen worden niet als één geheel uitgevoerd. Een bijkomend probleem hierbij

is dat bij RAID-levels met *parity* (zoals RAID5) ook nog eens de parity-blocks moeten herberekend worden. Indien het systeem tussen deze bewerkingen crasht of uitvalt door bv. elektriciteitsproblemen, dan bevindt de data op de array van schijven zich in een inconsistente toestand: stripes en parity blocks kunnen corrupt raken (Bonwick, 2005).

3.2.3 Toekomst van ZFS

Ondertussen werd Sun Microsystems overgenomen door Oracle in 2010, na het faillissement van deze eerstegenoemde (Oracle Corporation, Onbekend). In 2010 richtten enkele ex-ontwikkelaars van Solaris het illumos-project op met de bedoeling om een volledig open source variant van OpenSolaris te ontwikkelen; ook ZFS werd verder ontwikkeld als onderdeel van illumos (illumos, 2012, augustus 20). De reden hiervoor was dat Oracle geen nieuwe uitgaven meer uitbracht voor OpenSolaris. In 2013 werd het OpenZFS-project opgericht, dat zichzelf als de *"ware en open opvolger van het oorspronkelijke ZFS-project"* ziet (The OpenZFS Project, 2014).

Mede dankzij dit project is ZFS ondertussen beschikbaar op verschillende platformen, waaronder FreeBSD, Linux en macOS.

4. Ontwerpprincipes & architectuur van ZFS

In dit hoofdstuk worden enkele principes besproken waarop de ontwikkelaars zich hebben gebaseerd bij het ontwerp en de ontwikkeling van ZFS. Tevens wordt de architectuur van ZFS globaal beschreven, om zo de ontwerpbeslissingen van de ontwikkelaars wat meer toe te lichten.

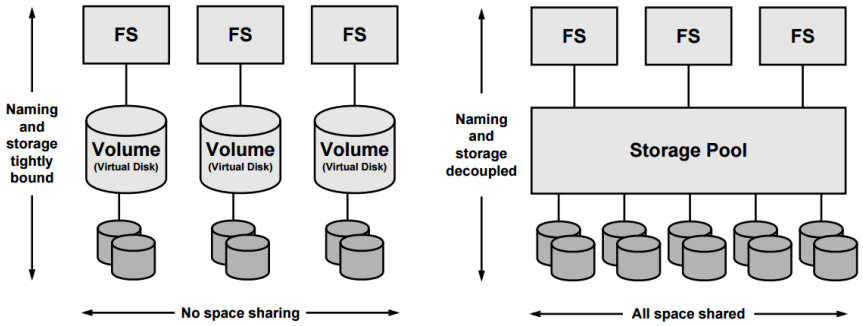
4.1 Ontwerpprincipes

De principes die aan de basis van ZFS liggen vloeiden meestal voort uit de problemen die de ontwikkelaars zelf ervaarden bij het gebruik van andere bestandssystemen.

4.1.1 Eenvoud van beheer & Storage Pools

Volgens Bonwick e.a. (2002) kan en moet het aanmaken en beheren van bestandssystemen een stuk makkelijker gemaakt worden. Hierbij speelt automatisatie van verschillende taken een belangrijke rol. Ook moet het mogelijk zijn om beheerderstaken (zoals bestandssystemen aanmaken en verwijderen) uit te voeren zonder de werking van het gehele systeem te ondermijnen (Bonwick e.a., 2002).

Een niet onbelangrijke feature hierbij zijn storage pools. Storage pools hebben als doel om opslagruimte zoveel mogelijk los te koppelen van de fysieke schijven: alle schijven bevinden zich in een pool van disks. Uit deze pool kunnen bestandssystemen worden aangemaakt, zonder rekening te moeten houden met de limitaties van bv. partities. Tevens kunnen bestandssystemen op een flexibele manier gebruik maken van deze pooled storage door automatisch in te krimpen en uit te breiden wanneer nodig (Bonwick e.a., 2002).



Figuur 4.1: Illustratie van ZFS pooled storage (rechts) t.o.v.volume-based storage (links) (Bonwick e.a., 2002)

4.1.2 Consistentie & Integriteit

Eén van de taken van een bestandssysteem is om ervoor te zorgen dat het in een consistente toestand blijft, i.e. het moet mogelijk zijn om van inconsistente toestanden te herstellen d.m.v. bijvoorbeeld een journal of door een filesystem check te draaien (Arpaci-Dusseau & Arpaci-Dusseau, 2015). Een nadeel aan deze technieken volgens Bonwick e.a. (2002) is dat deze niet makkelijk zijn om te implementeren, omdat bijvoorbeeld in het geval van journaling filesystems een roll back of roll forward moet worden uitgevoerd. Hierbij moet ook nog de volgorde van de bewerkingen in acht worden genomen (Arpaci-Dusseau & Arpaci-Dusseau, 2015).

De oplossing volgens Bonwick e.a. (2002) is om het bestandssysteem te allen tijde consistent te houden; er mag m.a.w. geen enkel moment zijn waarop het systeem in een inconsistente toestand kan terecht komen. Dit wordt verwezenlijkt door de Copy-On-Write eigenschappen van ZFS, waardoor bewerkingen atomair gebeuren (Li, 2009).

Een ander aspect waar ZFS een antwoord op tracht te vinden, is het voorkomen en minimaliseren van datacorruptie. Volgens Bonwick e.a. (2002) is het niet verstandig om volledig te vertrouwen op hardware, omdat er steeds een kans is dat deze bugs bevat. Dit kan leiden tot *silent data corruption*: dit is een fenomeen waarbij de schijf corrupte datablokken niet kan detecteren (Arpaci-Dusseau & Arpaci-Dusseau, 2015). Om deze reden bevat ZFS een uitgebreid checksumming-mechanisme dat werkt op blokniveau. In Hoofdstuk 5 wordt er dieper ingegaan op deze technieken.

4.1.3 Ingebouwde volumebeheerder

Indien bijvoorbeeld LVM (Logical Volume Manager) wordt gebruikt op een Linux-systeem voor het aanmaken van volumes, dan staan de volumes relatief "los"van de bestandssystemen die worden aangemaakt op deze volumes: eerst maakt men volumes aan, nadien pas bestandssystemen (Lewis, 2006).

Bonwick e.a. (2002) is van mening dat een volledige integratie van de volumebeheerder

in het bestandssysteem een aantal interessante voordelen met zich meebrengt, zoals betere optimalisaties en betere consistentie, omdat de volume manager nu "weet"hoe de bestandssystemen in de storage pool(s) zijn opgebouwd.

4.2 De architectuur van ZFS: een overzicht

De architectuur van ZFS is opgebouwd uit een zestal componenten (Li, 2009): de Storage Pool Allocator (SPA), de Data Management Unit (DMU), een ZFS POSIX Layer (ZPL), de ZFS Attribute Processor (ZAP), de ZFS Intent Log (ZIL) en ZFS Volume (ZVOL). Elk onderdeel biedt een bepaalde functionaliteit aan en levert diensten aan boven- en onderliggende lagen.

4.2.1 Storage Pool Allocator (SPA)

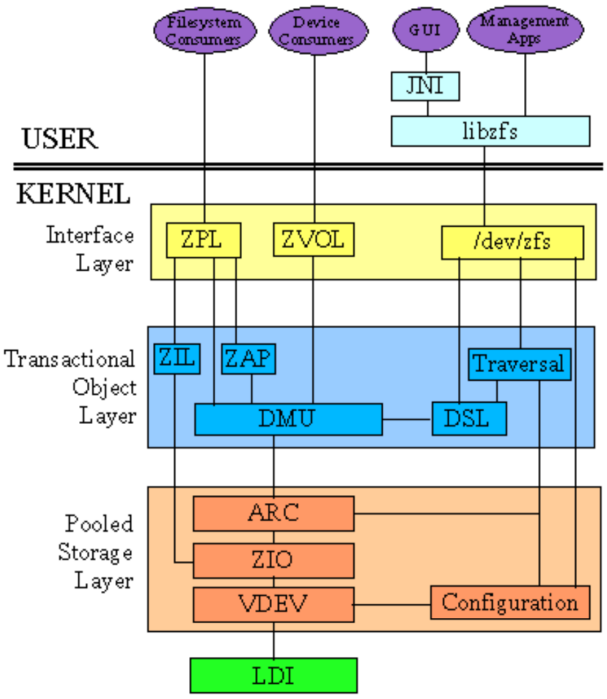
De taak van de SPA is hoofdzakelijk om datablokken van verschillende apparaten in één pool te verzamelen; de functionaliteit van de SPA komt m.a.w. in grote mate overeen met die van een volume manager. Het grote verschil tussen een volume manager en de SPA is dat de SPA louter een *interface* is voor virtuele datablokken; dit in tegenstelling tot een volume manager, dat volumes doorgeeft aan het besturingssysteem (Bonwick e.a., 2002).

De Storage Pool Allocator biedt een interface aan tot **data virtual addresses (DVA's)**: dit zijn de adressen van de virtuele datablocks die zich in een storage pool bevinden. De datablokken die zich fysiek op een schijf bevinden worden dus niet rechtstreeks aangesproken; alle communicatie van bovenliggende lagen gericht tot de fysieke disks moet eerst via de SPA. Hierdoor worden de ontwerpprincipes van *'Eenvoud van beheer'* en *'Storage Pools'* gerealiseerd: schijven kunnen dynamisch worden toegevoegd zonder de werking van de rest van het systeem te ondermijnen. Ook kan de systeembeheerder op een flexibele manier bestandssystemen aanmaken, zonder rekenschap te moeten geven aan de onderliggende fysieke structuur (Bonwick e.a., 2002).

Deze eigenschappen zjin te danken aan VDEV's (Virtual Devices). Dit zijn a.h.w. de bouwstenen van storage pools (Lucas & Jude, 2015). Deze worden in Hoofdstuk **??** wat meer toegelicht.

4.2.2 Data Management Unit (DMU)

De Data Management Unit is de laag boven de SPA en is a.h.w. de 'lijm' tussen de nogal *low-level* (virtuele) datablokken en de bovenliggende lagen. Deze component vertaalt de blokken naar ZFS objecten; dit is nodig omdat ZFS alles voorstelt als objecten. Zo stelt een ZFS-object van het type `DMU_OT_ACL` een access control list voor (Sun Microsystems, 2006). Objecten behoren steeds tot een bepaalde naamruimte (*namespace*) of dataset; dit verhoogt de flexibiliteit m.b.t. het beheer van bestandssystemen aanzienlijk. Zo wordt een bestandssysteem binnen ZFS voorgesteld door objecten uit een bepaalde



Figuur 4.2: Een overzicht van de verschillende componenten van ZFS (Kendi, Onbekend)

dataset. Aangezien objecten van elkaar gescheiden zijn door private namespaces, leven bestandssystemen (die uit een reeks van ZFS-objecten bestaan) ook naast en onafhankelijk van elkaar. Dit maakt taken zoals aanmaken en verwijderen van filesystems een stuk makkelijker voor zowel de gebruiker als de DMU (Bonwick e.a., 2002).

Daarnaast staat de DMU ook garant voor de consistentie van alle data. Om deze reden worden transacties binnen ZFS geïmplementeerd door de Data Management Unit als COW-transacties (Copy-On-Write). Hierbij wordt gebruik gemaakt van een 'alles of niets'-aanpak: ofwel is de transactie gelukt, ofwel is de transactie mislukt en moeten er back-upblokken worden aangewend (Bonwick e.a., 2002).

4.2.3 ZFS POSIX Layer (ZPL)

Deze laag vertaalt objecten komende van de Data Management Unit naar met POSIX compatibele bestandssystemen (Bonwick e.a., 2002). POSIX (acroniem voor Portable Operating System Interface) is een reeks van standaarden waaraan ontwikkelaars zich moeten houden indien ze willen dat hun programma's *portable* (overdraagbaar) zijn. Dit betekent dat het weinig tot geen moeite zou moeten kosten voor een ontwikkelaar om een programma draaiende te krijgen op bv. verschillende UNIX-systemen (IEEE and The Open Group, 2016). Solaris en macOS zijn voorbeelden van 'POSIX-compliant' besturingssystemen (The Open Group, Onbekend).

Voor het uitvoeren van bewerkingen maakt de ZPL gebruik van de onderliggende Data

Management Unit. Het aanmaken van een bestandssysteem gebeurt bijvoorbeeld door de ZFS POSIX Layer: de ZPL maakt enkele DMU-objecten met metadata e.d. aan en maakt gebruik van het transactiemodel van de DMU om deze objecten weg te schrijven. Dankzij deze aanpak verlopen deze bewerkingen ook atomair (Bonwick e.a., 2002).

4.2.4 ZFS Attribute Processor (ZAP)

De ZFS Attribute Processor is een component die samenwerkt met de DMU-laag. De ZAP manipuleert ZAP-objecten: dit zijn speciale DMU-objecten die metadata over allerhande dingen bijhouden in de vorm van key-value pairs. Deze informatie handelt over verschillende andere objecten, zoals datasets en filesystem objects. Voor een grote hoeveelheid attributen (en dus informatie) wordt er gebruikgemaakt van zgn. fatzaps; indien de hoeveelheid informatie nogal gering is, dan kiest ZFS voor microzaps (Sun Microsystems, 2006).

4.2.5 ZFS Intent Log (ZIL)

Deze component houdt logbestanden bij van alle transacties voor het geval dat het bestandssysteem toch in een inconsistente toestand zou terecht komen, bijvoorbeeld bij een stroompanne. Op deze manier kunnen transacties worden gereplayed indien consistentie van uiterst belang is. Deze records worden bijgehouden in het RAM-geheugen tot ze worden gepersisteerd door een DMU-transactie of door een `fsync` (Bonwick e.a., 2002).

4.2.6 ZFS Volume (ZVOL)

ZFS Volumes of ZVOL's zijn objecten die binnen ZFS worden voorgesteld als een koppel van een property object en een dataobject. Deze logische volumes worden dan aan het OS doorgegeven als ordinaire UNIX block devices en kunnen ook op deze manier benaderd worden (Sun Microsystems, 2006).

5. Het opslagmodel van ZFS

In dit hoofdstuk worden de interne bestandssysteemoperaties van ZFS wat meer toegelicht. Hierbij wordt vooral de werking van de Storage Pool Allocator en de Data Management Unit dieper uitgespit, aangezien deze componenten het beheer van data voor zich nemen.

5.1 Structuur van het bestandssysteem

Datablokken worden in ZFS voorgesteld als een boomstructuur. Vele andere bestandssystemen, zoals BTRFS op Linux, gebruiken ook boomstructuren voor het bijhouden van data (The BTRFS Project, 2017). De algemene werking van ZFS en andere, boomgebaseerde bestandssystemen verschillen niet zo heel veel. Echter zijn er ook eigenschappen die uniek zijn aan de manier waarop ZFS de data opslaat.

De belangrijkste elementen van een boom in de informatica zijn de volgende: de wortel (Eng.: *root*), de knopen (Eng.: *nodes*) en de bladeren (Eng.: *leaves*) (Cohen, g.d.). Indien men bij de wortel van een ZFS-boom start, dan komt men als eerste de *überblock* tegen: dit is simpelweg een andere benaming voor de wortel van de boom. Dit datablok bevat een checksum van zichzelf en verwijst naar de andere datablokken in de boom. Afhankelijk van de sectorgrootte van een schijf, bevat een ZFS pool een zeker aantal überblocks (een schijf waarvan de sectoren 512 bytes groot zijn geeft 128 überblocks als resultaat) (Lucas & Jude, 2015).

Zoals reeds gezegd in Hoofdstuk 4, worden alle blokken gechecksummed om corruptie van data tegen te gaan. Deze checksums worden bewaard in het datablok zelf en in de ouder van dit blok: op deze manier is de hele ketting van datablokken gechecksummed en kan er makkelijk schade worden vastgesteld en kan deze schade eventueel worden

gerepareerd. De *überblock* is het enige datablok die geen ouder heeft, en dus bewaart deze zijn checksum bij zichzelf (Bonwick e.a., 2002).

5.2 Checksumming & Redundantie op blokniveau

Checksumming is een goede oplossing om corruptie te detecteren. Echter moeten er nog steeds back-upblokken aanwezig zijn om van datacorruptie te herstellen. Indien men bijvoorbeeld mirroring toepast binnen een pool, dan kan ZFS de corrupte datablokken zelf herstellen door een goede kopie van de andere schijf van de mirror te halen. Als een applicatie een aanvraag doet voor een reeks datablokken waarvan er één of meerdere beschadigd zijn, dan repareert ZFS deze datablokken automatisch; nadien worden de herstelde blokken doorgegeven aan de applicatie, zonder dat deze laatstgenoemde hier iets van merkt (Bonwick e.a., 2002).

Maar zelfs zonder gebruik te maken van mirror VDEV's kan ZFS in sommige gevallen van corruptie herstellen. Hiervoor maakt ZFS gebruik van zogenaamde *ditto blocks*: dit zijn simpelweg kopieeën van belangrijke datablokken, zoals van metadata of pool data. Deze blokken worden zo ver mogelijk uit elkaar op de schijf geplaatst, om zo de impact van datacorruptie te minimaliseren (Lucas & Jude, 2015).

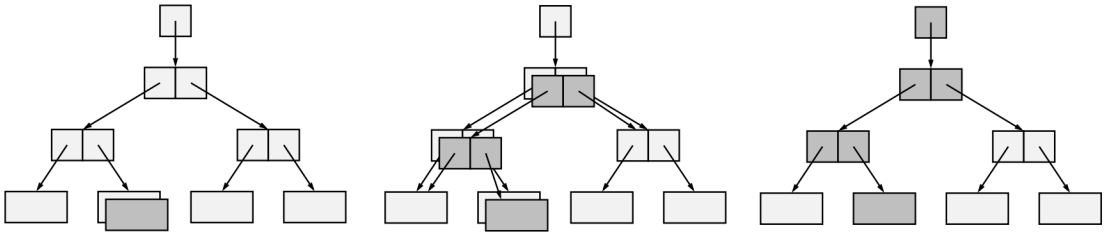
5.3 Transacties binnen ZFS

ZFS is een transactioneel bestandssysteem (Bonwick e.a., 2002). Transacties worden binnen ZFS afgehandeld door de DMU[1] en verlopen volgens een Copy-On-Write (COW) manier. Bij Copy-On-Write worden blokken nooit direct aangepast of overschreven; in plaats daarvan kopieert ZFS deze blokken eerst naar een andere locatie, om deze dan vervolgens aan te passen. Deze manier van van werken garandeert dat data op de schijf of schijven steeds consistent blijft: als er zich bijvoorbeeld een stroompanne voordoet midden in een transactie, dan kan ZFS makkelijk terugvallen op de oude boomstructuur, aangezien deze niet wordt overschreven (Lucas & Jude, 2015).

Uit de paper van Bonwick e.a. (2002) kan men de volgende stappen opmaken die worden ondernomen bij het uitvoeren van een transactie:

1. De ZFS POSIX Layer (ZPL) krijgt de opdracht om een reeks van blokken aan te passen, te creeëren of te verwijderen. Stel dat er in dit geval één enkel datablok moet worden aangepast en dat dit blok zich in een blad van de boom bevindt. De ZPL groepeert alle wijzigingen in een transaction group (txg); hierbij moet de ZPL de wijzigingen goed groeperen opdat het bestandssysteem aan het einde van de transactie in een consistente toestand wordt achtergelaten. Voor het uitvoeren van de transactie doet de ZPL beroep op de DMU.

[1] De DMU behandelt eigenlijk enkel objecten, maar voor de eenvoud wordt er in dit hoofdstuk meestal gesproken over blokken.



Figuur 5.1: Boomstructuur bij aanpassing van één datablok binnen een ZFS-transactie. Van links naar rechts: (1) Aanpassen van het datablok; (2) Aanpassen van de indirecte blokken; (3) Overschrijven van de überblock. Alle bewerkingen gebeuren op een COW-manier. (Bonwick e.a., 2002)

2. De DMU start de transactie en doorzoekt de boom. De DMU heeft het blok in kwestie gevonden, kopieert het naar een nieuwe locatie en past het vervolgens aan (Copy-On-Write). Na deze aanpassing moet ook de checkum van het datablok worden herberekend.
3. Aangezien andere blokken in de boom direct of indirect gekoppeld zijn aan het aangepaste datablok, moeten deze ook worden aangepast, en dit op dezelfde manier als het reeds aangepaste datablok. Ook moeten de checksums van deze blokken herberekend worden. Dit is een recursief gebeuren, en stopt bij de wortel van de boom, nl. de überblock.
4. Om de COW-eigenschappen van ZFS te garanderen, wordt het überblock niet direct aangepast, maar wordt er geroteerd naar de volgende beschikbare überblock. De oude en nieuwe boomstructuren worden dus a.h.w. met elkaar omgewisseld, en dit op een atomaire, transactionele manier.

Indien het systeem zou crashen tijdens deze transactie, dan kan de überblock corrupt raken. Aangezien corruptie kan worden gedetecteerd met behulp van checksums, kan er eenvoudig worden teruggevallen op een oudere versie van een überblock (Bonwick e.a., 2002).

6. Conclusie

Curabitur nunc magna, posuere eget, venenatis eu, vehicula ac, velit. Aenean ornare, massa a accumsan pulvinar, quam lorem laoreet purus, eu sodales magna risus molestie lorem. Nunc erat velit, hendrerit quis, malesuada ut, aliquam vitae, wisi. Sed posuere. Suspendisse ipsum arcu, scelerisque nec, aliquam eu, molestie tincidunt, justo. Phasellus iaculis. Sed posuere lorem non ipsum. Pellentesque dapibus. Suspendisse quam libero, laoreet a, tincidunt eget, consequat at, est. Nullam ut lectus non enim consequat facilisis. Mauris leo. Quisque pede ligula, auctor vel, pellentesque vel, posuere id, turpis. Cras ipsum sem, cursus et, facilisis ut, tempus euismod, quam. Suspendisse tristique dolor eu orci. Mauris mattis. Aenean semper. Vivamus tortor magna, facilisis id, varius mattis, hendrerit in, justo. Integer purus.

Vivamus adipiscing. Curabitur imperdiet tempus turpis. Vivamus sapien dolor, congue venenatis, euismod eget, porta rhoncus, magna. Proin condimentum pretium enim. Fusce fringilla, libero et venenatis facilisis, eros enim cursus arcu, vitae facilisis odio augue vitae orci. Aliquam varius nibh ut odio. Sed condimentum condimentum nunc. Pellentesque eget massa. Pellentesque quis mauris. Donec ut ligula ac pede pulvinar lobortis. Pellentesque euismod. Class aptent taciti sociosqu ad litora torquent per conubia nostra, per inceptos hymenaeos. Praesent elit. Ut laoreet ornare est. Phasellus gravida vulputate nulla. Donec sit amet arcu ut sem tempor malesuada. Praesent hendrerit augue in urna. Proin enim ante, ornare vel, consequat ut, blandit in, justo. Donec felis elit, dignissim sed, sagittis ut, ullamcorper a, nulla. Aenean pharetra vulputate odio.

Quisque enim. Proin velit neque, tristique eu, eleifend eget, vestibulum nec, lacus. Vivamus odio. Duis odio urna, vehicula in, elementum aliquam, aliquet laoreet, tellus. Sed velit. Sed vel mi ac elit aliquet interdum. Etiam sapien neque, convallis et, aliquet vel, auctor non, arcu. Aliquam suscipit aliquam lectus. Proin tincidunt magna sed wisi. Integer blandit

lacus ut lorem. Sed luctus justo sed enim.

Morbi malesuada hendrerit dui. Nunc mauris leo, dapibus sit amet, vestibulum et, commodo id, est. Pellentesque purus. Pellentesque tristique, nunc ac pulvinar adipiscing, justo eros consequat lectus, sit amet posuere lectus neque vel augue. Cras consectetuer libero ac eros. Ut eget massa. Fusce sit amet enim eleifend sem dictum auctor. In eget risus luctus wisi convallis pulvinar. Vivamus sapien risus, tempor in, viverra in, aliquet pellentesque, eros. Aliquam euismod libero a sem.

Nunc velit augue, scelerisque dignissim, lobortis et, aliquam in, risus. In eu eros. Vestibulum ante ipsum primis in faucibus orci luctus et ultrices posuere cubilia Curae; Curabitur vulputate elit viverra augue. Mauris fringilla, tortor sit amet malesuada mollis, sapien mi dapibus odio, ac imperdiet ligula enim eget nisl. Quisque vitae pede a pede aliquet suscipit. Phasellus tellus pede, viverra vestibulum, gravida id, laoreet in, justo. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Integer commodo luctus lectus. Mauris justo. Duis varius eros. Sed quam. Cras lacus eros, rutrum eget, varius quis, convallis iaculis, velit. Mauris imperdiet, metus at tristique venenatis, purus neque pellentesque mauris, a ultrices elit lacus nec tortor. Class aptent taciti sociosqu ad litora torquent per conubia nostra, per inceptos hymenaeos. Praesent malesuada. Nam lacus lectus, auctor sit amet, malesuada vel, elementum eget, metus. Duis neque pede, facilisis eget, egestas elementum, nonummy id, neque.

Bibliografie

A. Paterson, D. e. a. (1987). *A Case for Redundant Arrays of Inexpensive Disks*. University of California, Berkeley. Verkregen van https://www2.eecs.berkeley.edu/Pubs/TechRpts/1987/CSD-87-391.pdf

Arpaci-Dusseau, R. H. & Arpaci-Dusseau, A. C. (2015). *Operating Systems: Three Easy Pieces* (0.91). Arpaci-Dusseau Books.

Bonwick, J. e.a. (2002). *The Zettabyte Filesystem*. Verkregen van http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.184.3704&rep=rep1&type=pdf

Bonwick, J. (2005). RAID-Z. Verkregen 2 april 2017, van https://blogs.oracle.com/bonwick/entry/raid_z

Bonwick, J. e.a. (2015). The Birth of ZFS by Jeff Bonwick. OpenZFS Project. Verkregen 30 maart 2017, van https://www.youtube.com/watch?v=dcV2PaMTAJ4&feature=youtu.be

Bonwick, J. & Moore, B. (Unknown). ZFS The Last Word In File Systems. Sun Microsystems. Verkregen van https://wiki.illumos.org/download/attachments/1146951/zfs_last.pdf

Cohen, J. (g.d.). Trees. Verkregen van https://www.cs.jhu.edu/~cohen/CS226/Lectures/Trees.pdf

Goda, K. & Kitsuregawa, M. (2012). *The History of Storage Systems*. Institute of Electrical en Electronics Engineers (IEEE). Verkregen van http://ieeexplore.ieee.org/document/6182574/

Hickmann, B. & Shook, K. (2007). *ZFS and RAID-Z: The Uber-FS?* Verkregen van http://pages.cs.wisc.edu/~remzi/Classes/736/Fall2007/Projects/BrianKynan/paper.pdf

IEEE and The Open Group. (2016). The Open Group Base Specifications Issue 7. Verkregen van http://pubs.opengroup.org/onlinepubs/9699919799/

illumos. (2012, augustus 20). illumos Project Announcement - August 3, 2010. Verkregen van https://wiki.illumos.org/display/illumos/illumos+Project+Announcement+-+August+3%2C+2010

Kendi, C. (Onbekend). ZFS: Enhancing the Open Source Storage System (and the Kernel). Verkregen van https://www.blackhat.com/presentations/bh-dc-10/Kendi_Christian/Blackhat-DC-2010-Kendi-Enhancing-ZFS-slides.pdf

Lewis, A. (2006). LVM HOWTO: Setting up LVM on three SCSI disks. Verkregen van http://www.tldp.org/HOWTO/LVM-HOWTO/recipethreescsi.html#AEN1078

Li, A. (2009). *Digital Crime Scene Investigation for the Zettabyte File System* (proefschrift, Macquarie University). Verkregen van http://web.science.mq.edu.au/~rdale/teaching/itec810/2009H1/FinalReports/Li_Andrew_FinalReport.pdf

Lucas, M. W. & Jude, A. (2015). *FreeBSD Mastery: ZFS*.

M. Chen, P. e.a. (1994). *RAID: High-Performance, Reliable Secondary Storage*. University of Michigan , Carnegie Mellon University , University of California (Berkeley). Verkregen van http://citeseerx.ist.psu.edu/viewdoc/download;jsessionid=59E48A55C27748208046D52C5730C98F?doi=10.1.1.41.3889&rep=rep1&type=pdf

Oracle Corporation. (Onbekend). Oracle and Sun Microsystems. Verkregen van https://www.oracle.com/sun/index.html

Sun Microsystems. (2006). *ZFS on-disk specification*. Verkregen van http://www.giis.co.in/Zfs_ondiskformat.pdf

The BTRFS Project. (2014, maart 4). Status. Verkregen van https://btrfs.wiki.kernel.org/index.php/Status

The BTRFS Project. (2017). On-disk Format. Verkregen van https://btrfs.wiki.kernel.org/index.php/On-disk_Format#Basic_Structures

The Open Group. (Onbekend). Open Brand. Verkregen van https://www.opengroup.org/openbrand/register/xy.htm

The OpenZFS Project. (2014). History - OpenZFS. Verkregen van http://open-zfs.org/wiki/History

Woordenlijst

atomair In de informatica betekent atomiciteit (Engels: atomicity) hetzelfde als ondeelbaarheid; vaak wordt dit begrip ook omschreven als een "alles-of-niets" aanpak. In de context van bijvoorbeeld databanksystemen heeft atomiciteit betrekking op o.a. transacties: ofwel verloopt een transactie succesvol, ofwel mislukt deze (door bv. een onderbreking) en moet de databank kunnen teruggebracht worden naar een consistente toestand. Meestal betekent dit dan dat er een rollback moet worden uitgevoerd.. 20, 22

betrouwbaarheid Betrouwbaarheid (Engels: reliability) heeft betrekking op de betrouwbaarheid van een computersysteem, i.e. of deze al dan niet zijn taken op een juiste manier uitvoert. Men kan bijvoorbeeld stellen dat er zich bij een computersysteem een X aantal fouten mogen voordoen; als deze grens overschreden wordt, dan wordt het systeem als onbetrouwbaar verklaard. Daarnaast heeft betrouwbaarheid ook te maken met de handelswijze van een systeem indien er zich een fout voordoet: men kan zich bijvoorbeeld afvragen hoe er zou moeten gereageerd worden op een kritieke fout en hiervoor test cases opstellen om de betrouwbaarheid van het systeem te testen. 13, 14, 15

capaciteit In de context van schijven of RAID-systemen kan capaciteit (Engels: capacity) een aantal betekenissen hebben. In deze scriptie wordt er vooral gesproken over de bruikbare capaciteit van een schijf of RAID-array: dit is de hoeveelheid schijfruimte die kan gebruikt worden door gebruikers om data op te slaan. Daarnaast kan men ook spreken over de totale capaciteit: dit is de som van de gebruikte schijfruimte en de vrije schijfruimte. In theorie is de totale capaciteit van een schijf gelijk aan de werkelijke grootte van een schijf. . 13, 14, 15

performantie Performantie (Engels: performance) is een vrij uitgebreid begrip in de

informatica. In het algemeen bedoelt men met performantie meestal hoe goed een computersysteem vooropgestelde taken kan uitvoeren in een bepaalde situatie. Bij het meten van performantie van een systeem kan men één of meerdere aspecten beschouwen: bij schijven kan bijvoorbeeld het aantal I/O's (Invoer- en uitvoerbewerkingen) worden genomen als maatstaf; bij CPU's kan een maatstaf bijvoorbeeld het aantal instructies per seconde zijn dat deze kan verwerken. 9, 13, 14

Lijst van figuren

Lijst van tabellen