ZFS-TIEDOSTOJÄRJESTELMÄ JA SEN HYÖDYNTÄMINEN TIEDOSTOJEN JAKAMISEEN WINDOWS-YMPÄRISTÖSSÄ

Anssi Marttinen

Opinnäytetyö
Huhtikuu 2012
Tietojenkäsittely
Tietoverkkopalvelut
Tampereen ammattikorkeakoulu

TAMPEREEN AMMATTIKORKEAKOULU
Tampere University of Applied Sciences

TIIVISTELMÄ

Tampereen ammattikorkeakoulu
Tietojenkäsittelyn koulutusohjelma
Tietoverkkopalvelut

MARTTINEN, ANSSI: ZFS-tiedostojärjestelmä ja sen hyödyntäminen tiedostojen jakamiseen Windows-ympäristössä

Opinnäytetyö 43 sivua, josta liitteitä 2 sivua
Huhtikuu 2012

Opinnäytetyöni käsittelee ZFS-tiedostojärjestelmää ja sen hyödyntämistä Windows-ympäristössä tiedostojen jakamiseen huomioiden nykyiset tietoturvatarpeet ja käyttöoikeudet. Opinnäytetyössä käydään läpi perustason konfiguraatiot ja hallinta tiedostojaoille. Tiedostojaot on toteutettu käyttäen Unix-käyttöjärjestelmä Solaris 10:tä ja ilmaista avoimen lähdekoodin ohjelmistokokoelmaa SAMBA:a, joka tarjoaa tiedostojako- ja tulostuspalveluita Windows-työasemille.

Opinnäytetyön tarkoituksena oli esitellä ja rakentaa pienille yrityksille sopiva tiedostojakoympäristö, jonka kautta saadaan hyödynnettyä myös ZFS:n ominaisuuksia Windows-ympäristössä. Opinnäytetyön tietojen hyödyntämisen ennakkovaatimuksena on kuitenkin jo valmis Unix/Linux-ympäristö, joka tukee ZFS-tiedostojärjestelmää. Opinnäytetyöstä on rajattu pois itse käyttöjärjestelmän konfigurointi käyttökuntoon, koska se jo yksinään vastaisi laajuudeltaan lähes yhtä opinnäytetyötä.

Opinnäytetyön tavoitteena oli tuoda esille halpa ja varma vaihtoehto muille markkinoilla oleville tiedostojärjestelmä-, palvelin- ja tiedostojakoratkaisuille, sillä jotkin Unix- ja useat Linux-distribuutiot ovat ilmaisia ja tukevat ZFS:ää.

Avainsanat: tiedontallennus, palvelimet, unix

ABSTRACT

Tampereen ammattikorkeakoulu
Tampere University of Applied Sciences
Degree Programme in Business Information Systems
Option of Network Services

MARTTINEN, ANSSI: ZFS-file system and its utilization in Windows environment for sharing files.

Bachelor's thesis 43 pages, appendices 2 pages
April 2011

This Bachelor's thesis focuses on ZFS-file system and its utilization in Windows environment for sharing files while considering the present information security requirements and user rights. Thesis covers the basic configuration of the file shares, which are implemented by using Unix operating system Solaris 10 and free Open Source/Free Software suite SAMBA. Solaris provides the operating system platform and Samba provides seamless file and print services to Windows clients.

The aim of this thesis is to introduce and build suitable file sharing environment for small companies and investigate the possibilities of ZFS file system in Windows environment. Prerequisite of efficiently using thesis's information is a pre-configured Unix/Linux environment, which supports ZFS file system. Operating system configurations and installations for making it operational are cropped from the thesis.

The goal of the thesis is to point out an option for present file system, server and file share solutions. This is relevant since some Unix and several Linux distributions are free and support ZFS.

Keywords: data storage, servers, unix

LYHENTEET JA KÄYTETTÄVÄT TERMIT

pfexec	Tämä käsky antaa root-käyttäjän oikeudet tunnuksille, jolle on määritelty sysadminin rooli Solaris-käyttöjärjestelmässä.
bit rot	Bittien ”lahoamista”, johtuen sähkövarauksen tai magneettisuuden muutoksen takia.
slice	Viipale on kovalevyn fyysinen osajoukko, joka koostuu peräkkäisistä lohkoista.
inode	Datarakenne johon tallennetaan tietoja tiedostoista tai hakemistoista kuten esimerkiksi käyttäjäoikeudet tai tiedostotyyppi.
state database	Solaris Volume Manager tallentaa kaikkien levyjen ja varalevyjen sekä levyjoukkojen konfiguraatiot ja tilan tilatietokantaan.
submirror	Varsinaisen peililevyn metalaite
meta device	Varsinaiseen levyyn tai levypakkaan osoittava virtuaalilaite
mount	”Nostaa ylös levy”
block pointer	Lohko-osoitin; Osoittaa missä tiedostoon liittyvät tiedot ovat fyysisesti levyllä.
dma parity error	DMA-pariteettitarkistuksen virhe. Pariteettitarkistuksella tarkastetaan, että muistista haettava data vastaa aiemmin sinne kirjoitettua dataa.

phantom write	Virhetilanne missä levy ilmoittaa tehneensä kirjoituksen, mutta todellisuudessa mitään ei ole kirjoitettu.
shadow copy	Varjokopio on tiedostojärjestelmän ominaisuus, joka ottaa datasta automaattisesti kopion kun vanhan datan päälle tallennetaan uusi versio. Tämä mahdollistaa vanhan datan palauttamisen nopeasti ja helposti, mikäli tällainen tarve tulee.
journalointi	Tiedostojärjestelmän ominaisuus, joka pitää kirjaa tulevista muutoksista. Tulevat muutokset siis kirjoitetaan ensin ”journaliin” ennen kirjoittamista varsinaisesti levylle. Näin äkkinäisen sähkökatkon tai muun vastaavan sattuessa, dataa ei korruptoidu tai katoa.

1. Johdanto

Opinnäytetyön tarkoituksena on ollut esitellä ja rakentaa vaihtoehtoinen tiedostojako-ympäristö pienille yrityksille, jonka kautta saadaan hyödynnettyä ZFS:n ominaisuuksia ja varmatoimisuutta.

Opinnäytetyössä käyn läpi tiedostojärjestelmien tarkoitusta ja historiaa, jolla saadaan sopivasti perspektiiviä ZFS-tiedostojärjestelmän arkkitehtuuriin ja suunnitteluun, jonka jälkeen tuon esille ZFS:n vahvuudet muihin tiedostojärjestelmiin nähden esimerkkien vahvistamana. Oleellista tässä on kiinnittää huomiota aikaisempien tiedostojärjestelmien vaiettuihin puutteisiin esimerkiksi tarkistussummien suhteen.

Käytännön toteutuksen osuudessa käytän esimerkkinä kuvitteellista muutaman hengen yritystä, jolla on potentiaalia laajentua tulevaisuudessa. Konfigurointiin sisältyy esimerkiksi storage poolien ja tiedostojärjestelmien luonti, oikeuksien määrittely ja SAMBA:n konfigurointi.

Ympäristön konfiguroinnin jälkeen pyrkimyksenä on käydä läpi muutamat ylläpidolliset seikat ja tilannevedosten hyödyntäminen tiedostojärjestelmän palauttamisessa aikaisempaan tilaan.

2. Tiedostojärjestelmien tarkoitus

Tiedostojärjestelmän tarkoitus on hallinnoida fyysistä levyä kun on tarve muuttaa, tallentaa tai hakea tietoa. Tiedostojärjestelmä toimii ”tulkkina” ja ”oppaana” ohjelmille, jotka käyttävät tallennusmediaa. Toisin sanoen ohjelman tarvitsee vain tietää mitä tallennetaan, muutetaan tai haetaan. Oleellisimpia tiedostojärjestelmän tehtäviä ovat siis mm. tiedostolistauksen ja hakemistorakenteen ylläpitäminen, tilan ja metadatan, sekä erityisesti nykyään käyttöoikeuksien hallinta.

Tiedostolistauksella tarkoitetaan tiedostojärjestelmän ominaisuutta, joka ylläpitää listaa tiedostojen sijainnista kovalevyn lohko-tasolla. Hakemistorakenne sen sijaan on ikään kuin ”sisällysluettelo”, joka kokoaa tiettyjen hakemistojen alle tietyt tiedostot. Tätä ei tehdä fyysisellä tasolla vaan ainoastaan loogisella tasolla. Hakemistorakenteita on olemassa hiearkkisia sekä lineaarisia, joista jälkimmäistä ei juurikaan enää tavata käytössä olevissa levytiedostojärjestelmissä. Ensimmäinen hierarkkinen tiedostojärjestelmä esiteltiin Multics-käyttöjärjestelmän yhteydessä.

Metadatassa säilytetään tietoa esimerkiksi tiedoston koosta, luomispäivämäärästä tai siitä milloin tiedostoa käytettiin viimeksi. Laajennettujen tiedosto-ominaisuuksien avulla, joita tukevat esim. NTFS, ext2/ext3 ja XFS, voidaan metadataan sisällyttää myös esimerkiksi dokumentin tekijä tai vaikka kuvan koko.

Tiedostojärjestelmä hallinnoi myös tiedostojen ja hakemistojen käyttöoikeuksia. Nykyisessä tietoliikenneyhteiskunnassa on oleellista hallinnoida tiedostojen käyttöoikeuksia, jottei kuka tahansa pääsisi mihin tahansa käsiksi verkossa. Tämä on ollut aina tärkeää, mutta viimeisen 10 vuoden aikana sen merkitys on korostunut entisestään. Unix tiedostojärjestelmineen oli tässä asiassa ”edelläkävijä” sillä UNIX-käyttöjärjestelmät on suunniteltu alusta lähtien siltä kantilta, että järjestelmällä on useita käyttäjiä. Windows-maailma heräsi tähän vasta vuonna 1993 Windows NT 3.1 ja NTFS-tiedostojärjestelmän myötä (Wikipedia 2011. Comparison of file systems).

Vaatimuksien kasvaessa vuosi vuodelta, tiedostojärjestelmiin on lisätty lukuisia eri ominaisuuksia aiemmin mainituiden ”ydinominaisuuksien” lisäksi. Näistä mainittakoon muun muassa unicode-tuki, kryptaus, nopeat indeksointitekniikat ja journalointi.

3. Nykyaikaiset levytiedostojärjestelmät

3.1 FAT

FAT eli File Allocation Table oli ensimmäisiä tiedostojärjestelmiä, jossa oli nykyaikaisen tiedostojärjestelmän piirteitä. FAT-12 oli ensimmäinen versio FAT:sta, joka julkaistiin 80-luvun alussa. Se oli 12-bittinen tiedostojärjestelmä, joka tämän myötä mahdollisti tiedon tallentamista 32 megatavun edestä. Tosin tuohon aikaan yleisin tallennusmedia oli 160 kilotavun yksipuolinen levyke, joten tästä maksimikoosta ei juurikaan hyödytty. Muita tiedostojärjestelmälle ominaisia piirteitä oli taulukko, joka piti kirjaa siitä missä oli vapaata tai käytettyä tilaa ja mikä osio levystä oli mahdollisesti vaurioitunut (bad sector).

Tallennusmediat kehittyivät, mikä loi uusia vaatimuksia tiedostojärjestelmille. Myös kovalevyt nykymuodossaan alkoivat saada jalansijaan mikrotietokoneiden yleistyessä ja siirtyessä suurista halleista ihmisten koteihin. Vuonna 1987 Microsoft toi MS-DOS 3.31 mukana FAT-16:sta, joka oli, kuten nimestäkin saattaa päätellä, 16-bittinen. Tämä toi teoreettiset puitteet yli kahden gigatavun tallennuskapasiteetille. Käyttäjien harmiksi Microsoft ei vielä tässäkään vaiheessa huomioinut FAT:n yhtä suurinta ongelmaa eli fragmentoitumista. FAT kirjoittaa ensimmäiseen vapaaseen paikkaan levyllä. Tiedostojen poistaminen ja lisääminen aiheutti siis selvää fragmentoitumista hyvin nopeasti, joka taas vuorostaan hidasti ja kulutti kovalevyä pakottamalla lukupään hyppimään tarpeettomasti puolelta toiselle. Microsoft jätti tämän huolen kolmannen osapuolen ohjelmien, kuten Norton Utilitiesin hoidettavaksi (Wikipedia 2011. Comparison of file systems).

Vuonna 1995 julkaistiin Windows 95 ja sen mukana FAT-32, joka edelleen vastasi kasvavaan tilantarpeeseen, mutta ei varsinaisesti korjannut tiedostojärjestelmässä havaittuja ongelmia. Käytettävän tilan teoreettinen maksimi kasvoi kahdeksaan teratavuun, mutta fragmentoitumisongelma oli edelleen läsnä, samoin kuin hauska 8+3 rajoitus tiedostonimissä. Windows 95 kuitenkin sisälsi sisäänrakennetun eheytysohjelman (defragment), jolla tiedostojen palaset saatiin kasaan. FAT32:en VFAT- ominaisuus mahdollisti näennäisesti pitkät tiedostonimet käyttöjärjestelmän graafisella puolella, vaikka todel-

lisuudessa DOS:sta tarkasteltuna ne noudattivat 8+3-sääntöä (Pitkatiedostonimi.txt siis nimettiin PITKAT~1.txt) (Wikipedia 2011. Comparison of file systems).

3.2 MFS ja HFS

Applen tallissa vaikutti MFS eli Macintosh File System samoihin aikoihin kuin FAT-12 ja FAT-16. Tiedostojärjestelmänä se oli omalla tavallaan lähes yhtä alkeellinen kuin ”kilpailijansa”. MFS oli lineaarinen tiedostojärjestelmä, mutta toisaalta se tuki Finderin rajoittamana 32 merkin tiedostonimiä. MFS julkaistiin vuonna 1984 ja vain 3 vuotta sen jälkeen HFS (Hierarchial File System) korvasi sen. HFS:n varjopuolena mainittakoon se, että sitä käyttävät MAC:it tallensivat kaiken tiedon tiedostoista ja hakemistoista luettelotiedostoon (Catalog File), jota pystyi lukemaan vain yksi ohjelma kerrallaan ja jonka korruptoituessa koko järjestelmä saattoi mennä käyttökelvottomaksi. Suurimpina eroina MFS:ään näkyi tallennuskapasiteetin kasvu kahdestakymmenestä megatavusta kahteen teratavuun ja hierarkkisuus, jonka myötä myös Macintoshille saatiin aidot hierarkkiset hakemistorakenteet.

MFS ja HFS toivat myös tiedostojärjestelmien maailmaan ”forkit” (forks). Tämä oli uudenlainen tapa käsitellä metadataa. Sen sijaan, että tiedostojen metadata tallennettaisiin yksittäiseen paikkaan, se tallennettiin MFS:ssä ja HFS:ssä varsinaisen tiedoston rinnalle. Varsinaista tiedostoa tietoineen kutsuttiin tällöin ”data forkiksi” ja näkymätöntä metadataa ”resource forkiksi” (Wikipedia 2011. Comparison of file systems).

3.3 OFS ja FFS

OFS (Old File System) ja FFS (Fast File System) olivat Amigan tiedostojärjestelmiä samalta aikakaudelta kuin FAT ja MFS/HFS. Sinänsä Amigan tiedostojärjestelmät eivät tuoneet miten uutta tiedostojärjestelmien saralta, mutta AmigaOS tuki poikkeuksellisesti kahta tiedostojärjestelmää (Wikipedia 2011. Comparison of file systems).

3.4 UFS

Unix File System oli UNIX-käyttöjärjestelmien tiedostojärjestelmä, joka julkaistiin 70-luvun alussa. Alun perin se tunnettiin yksinkertaisesti nimellä FS eli File System. Myöhemmin nimi muotoutui UFS:ksi tai sitä kutsuttiin vaihtoehtoisesti nimellä Berkeley Fast File System. UFS:ään on lisätty vuosien varrella monia ominaisuuksia ja siitä on tehty monia versioita eri UX-pohjaisille käyttöjärjestelmille. Tiedostojärjestelmää optimoitiin tarpeen mukaan. Yksi suuri tekijä oli myös kovalevyjen koon kasvu, joka pitkälti asetti vaatimukset tiedostojärjestelmille (Wikipedia 2011. Comparison of file systems).

Esimerkkejä optimoinneista:

Sylinteriryhmien (Cylinder Group) lisääminen vähensi lukupään hyppimistä inode:ien ja datalohkojen välillä ja vähentäen fragmentoitumista kun tarvittava tieto on cylinder groupin sisällä tiiviimmässä paketissa eikä ympäri kovalevyä.

Kovalevyjen ja sillä olevien tiedostojen koon kasvaessa entisestään fragmentoituneiden tietojen lukeminen aiheutti entistä enemmän hukkaa. Pienimmän tallennusyksikön (allocation unit) tilaa suurennettiin ensin 512 tavusta yhteen kilotavuun ja siitä pian kahdeksaan kilotavuun. Tästä seurasi useita tehokkuutta lisääviä seikkoja. Tiedosto sijoittui loogisesti peräkkäisille sektoreille nyt todennäköisemmin. Tiedoston ei tarvinnut levittäytyä niin monille tallennusyksiköille eikä tällöin niin useista tarvinnut pitää kirjaa. Lisäksi se mahdollisti suuremmat levyt kasvattamalla mahdollisten tallennusyksiköiden määrää. Tämä johtui siitä, että tallennusyksiköiden ”osoitetieto” on kiinteä bittijono tallennusyksikön ohessa.

Suuremmat tallennusyksiköiden koot taas aiheuttavat tiedostojen fragmentoitumista.
Tilan käyttöä parantaakseen lisättiin ”block suballocation” -ominaisuus, joka keräsi tiedostojen ”ylijäämät” samaan kasaan. Tällöin puoliksi täytettyjä tallennusyksiköitä oli levyllä huomattavasti vähemmän (Wikipedia 2011. Unix File System).

3.5 Ext2, ReiserFS ja ext3

Ext2, ReiserFS ja ext3 tunnetaan usein ”Linux-tiedostojärjestelminä”. Ext2 vuodelta 1992 oli hyvin pitkälti klooni UFS:stä ja tämän ansiosta esimerkiksi levyn eheyden tarkistusohjelma fsck (File System Consistency Check) oli helppo kopioida UFS:stä.

2000-luvun alussa ReiserFS korjasi ext2 puutteet, jonka takia tietyt Linux distribuutiot ottivatkin tämän ensisijaiseksi tiedostojärjestelmäksi. ReiserFS toi Linux-maailmaan muun muassa journaloinnin ja ”B-tree” -indeksoinnin, joka nopeutti hakuja huomattavasti. Journalointi sinänsä ei ollut uusi keksintö sillä ensimmäisen kaupallisen version journaloivan tiedostojärjestelmän VxFS:n toi markkinoille Veritas Software jo vuonna 1991.

Parin vuoden sisällä ilmestyi myös ext3, joka korjasi vanhemman ext2:n puutteet. Ext3 vahvuutena oli se, että se pohjautui pohjimmiltaan vakaaseen ja varmaan ext2-tiedostojärjestelmään ja ext2:n pystyi konvertoimaan helposti ext3:ksi formatoimatta kovalevyä (Wikipedia 2011. Comparison of file systems).

3.6 NTFS

NTFS vastasi Windows-maailmassa tiedostojärjestelmien vaatimuksiin vuonna 1993. 64-bittisenä tiedostojärjestelmänä tallennuskapasiteetti nousi 16 eksatavuun. Nykyaikaisen B+Tree-indeksoinnin, unicode-tuen ja journaloinnin lisäksi siinä oli myös ACL-tuki (Access Control List) käyttöoikeuksien hallinnoimista varten. Myöhempien versioiden myötä NTFS:ään on lisätty hallinnan ja luotettavuuden kannalta oleellisia ominaisuuksia kuten tiedostojen kryptaus, hakemistojen tilarajoitukset, ”Shadow Copy”-ominaisuus ja tiedostojen pakkaus (Wikipedia 2011. Comparison of file systems).

3.7 HFS+

HFS+ eli Hierarchial File System Plus on Applen kehittämä nykyaikainen versio HFS:stä. Nykyään se on Mac OS:n oletustiedostojärjestelmä, minkä käyttöön se alun perin valjastettiin. Päivityksien myötä HFS+ on pysynyt ajan tasalla ja täyttää nykyään tiedostojärjestelmien perusvaatimukset (Wikipedia 2011. Comparison of file systems).

4. ZFS-tiedostojärjestelmä

4.1 Johdanto

ZFS (Zettabyte File System) on uuden sukupolven tiedostojärjestelmä, joka on vikasietoinen, skaalautuva ja helposti ylläpidettävä. Se sopii niin suurien palvelinympäristöjen kuin kotikoneidenkin tiedostojärjestelmäksi. Etenkin palvelinympäristöjen tapauksessa ylimääräinen työ ja tehottomuus on tärkeää kitkeä pois, mutta tätä ei ole aiemmin voitu tehdä vaarantamatta tietojen eheyttä tai saatavuutta. Nykyään tähän tarjoaa mahdollisuuden ZFS, joka julkaistiin vuoden 2005 lopulla (Oracle Corporation 2010. ZFS Administration Guide) (Wikipedia 2011. Comparison of file systems).

Vanhasta poiketen ZFS:n kanssa on ajateltava ”tiedostojärjestelmillä”. Eli sen sijaan, että luo uutta hakemistoa uudelle käyttäjälle tai käyttäjäryhmälle, tulisi miettiä, että kannattaako itse asiassa luoda kokonaan uusi tiedostojärjestelmä. Uusien tiedostojärjestelmien luominen ZFS:ssä on hyvin helppoa, nopeaa ja lisäksi kannattavaa, sillä ne tuovat mukanaan parempaa hallittavuutta esimerkiksi varmuuskopioinnin suhteen. Tällöin voidaan lisäksi muuttaa tiedostojärjestelmien ominaisuuksia käyttäjän, yksikön tai yrityksen tarpeisiin riippuen siitä mitä kyseinen tiedostojärjestelmä kattaa.

ZFS käyttää ns. storage pooleja, joilla ylläpidosta ja tilan lisäämisestä tehdään nopeaa ja helppoa. Tiedostojärjestelmät luodaan storage poolin päälle, joka hallinnoi kaikkia siihen liitettyjä fyysisiä laitteita.

ZFS:n copy-on-write transaktionaalinen menetelmä estää kirjoitusvaiheessa olevan datan katoamisen sähkökatkon yllättäessä, sillä mitään dataa ei ylikirjoiteta tallentamatta sitä ensin johonkin. ZFS:n kehittyneemmällä tarkistussummien käytöllä pystytään estämään datan korruptoituminen laajemmassa mittakaavassa verrattuna perinteisiin tiedostojärjestelmiin, joissa tarkistussumma on sisällytetty lohkoon itseensä.

Skaalautuvuus on ollut yksi suurimpia prioriteetteja ZFS:n suunnittelussa. Ensimmäisenä 128-bittisenä tiedostojärjestelmänä tila ei tule loppumaan aivan heti kesken, sillä

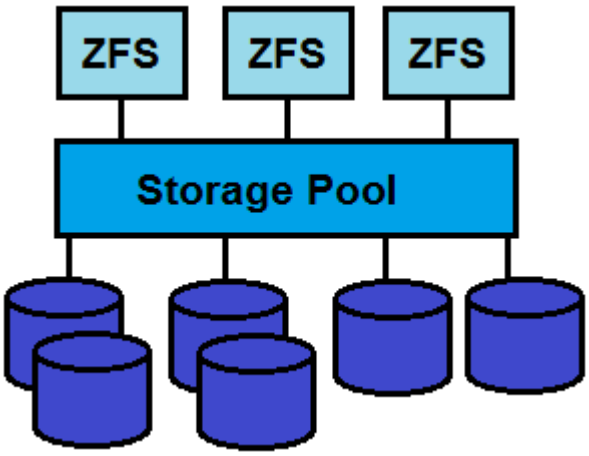
ZFS-järjestelmään pystytään tallentamaan 256 zettatavua * 10^15 edestä tietoa (1 zettatavu = 1 073 741 824 teratavua). Metadata tallennetaan dynaamisesti, jolloin inodeille ei tarvitse esivarata tilaa. Muita skaalautuvuuteen ja nopeuteen vaikuttavia tekijöitä ovat mm. datan liukuhihnoitus (data pipelining), dynaaminen lohkon koon muuttaminen (dynamic block sizing), älykäs esihaku (intelligent prefetch), dynaaminen raidoitus (dynamic striping) sekä sisäänrakennettu pakkaaminen (built-in compression) (Oracle Corporation 2010. ZFS Administration Guide).

4.2 Vahvuudet

4.2.1 Storage poolit

Storage poolit ovat yksi ZFS:n merkittävimmistä ominaisuuksista perinteisiin tiedostojärjestelmiin verrattuna. Perinteisten tiedostojärjestelmien ylläpidollinen ongelma ja rajoittunut skaalautuvuus tulee usein esiin kun järjestelmän ja sen eri osioiden tilantarvetta määritellään tai myöhemmin koetaan tarpeelliseksi lisätä levytilaa. Usein ongelmana on myös se, että vapaata levytilaa on muualla kuin siellä missä tarve on suurin, mutta tilan siirtäminen vaatii kohtuuttoman paljon manuaalista operointia levykirjainten, osioiden ja taltioiden suhteen.

ZFS:ssä tämä ongelma on ratkaistu siten, että tiedostojärjestelmät luodaan storage poolin päälle, joka hallinnoi kaikkia siihen liitettyjä fyysisiä laitteita (kuvio 1). Niiden avulla päästään tietyllä tapaa eroon fyysistä ja loogisista rajoitteista sekä levyosioiden tai –jakojen etukäteissuunnittelusta ja myöhemmin - ylläpidollisista ongelmista levytilan suhteen.



Kuvio 1: Storage pool

Esimerkki: UFS- ja ZFS konfiguroinnin vertailu

Yrityksellä on asiakkaana kolme yksityistä yrittäjää - alpha, beta ja gamma. Kaikilla heillä on huomattavia määriä tärkeää dataa, jonka täytyy olla saatavilla kokoajan. Myöhemmin saattaa esiintyä tarvetta lisätä tilaa.

UFS:n konfigurointi

Luodaan tilatietokannasta replika molemmille peiliin tuleville osioille (state database replica).
metadb –a –f disk1:slice0 disk2:slice0

Luodaan ketju (concatenation) tai raita (stripe) levyn 1 viipaleesta 1.
metainit d10 1 1 disk1:slice1

Luodaan ketju tai raita levyn 2 viipaleesta 1.
metainit d11 1 1 disk2:slice1

Seuraavilla komennoilla luodaan ketjuista alipeilit (submirror) ja asetetaan metalaite (metadevice) viittaamaan näihin kahteen alipeiliin. Muodostetaan siis yksisuuntainen peili (one-way mirror) luomalla metalaite (metadevice) d20 käyttäen alipeiliä d10.

metainit d20 –m d10

Luodaan kaksisuuntainen peili (two-way mirror) liittämällä alipeili d11 metalaitteeseen d20.
metattach d20 d11

Luodaan itse tiedostojärjestelmä viitaten metalaitteeseen d20, joka viittaa alipeileihin d10 ja d11. Käyttöjärjestelmälle metalaite d20 näyttää vain yhdeltä tavalliselta levyltä.

newfs /dev/md/rdsk/d20

Nostetaan (mount) levy ylös hakemistoon /export/home/alpha
mount /dev//md/dsk/d20 /export/home/alpha

Huom. tämä pitäisi tehdä yhteensä kolme kertaa, jotta kaikki kolme henkilöä saisivat oman peilatun tiedostojärjestelmänsä käytettäväkseen. Tilan lisääminen olisi tässä vaiheessa oma lukunsa ja tarvittavia konfigurointeja sen suhteen en tuo esille. Konfigurointeihin kuitenkin liittyisi levyviipaleiden lisäämistä yllä esitettyyn tyyliin (Hunter, Jeff 2010. Solaris Volume Manager) (Oracle Corporation 2010. Solaris Volume Manager Administration Guide).

ZFS:n konfigurointi

Luodaan peili levyistä yksi ja kaksi.
zpool create home mirror disk1 disk2

Luodaan tiedostojärjestelmä ja asetetaan se hakemistoon. Ilman /export/home/xyz määritystä ZFS asettaisi tiedostojärjestelmän sijaintiin home/xyz.
zfs create home/alpha /export/home/alpha
zfs create home/beta /export/home/beta
zfs create home/gamma /export/home/gamma

Tilan lisääminen:
zpool add home mirror disk3 disk4

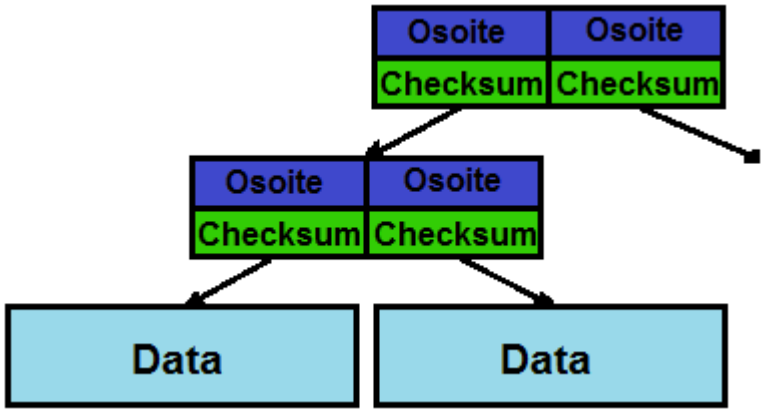
4.2.2 Transaktionaalisuus ja copy-on-write –operaatiot

Transaktionaalinen malli varmistaa sen, että levyllä oleva data on aina eheää. Kirjoitusoperaatiot ovat jakamattomia (atomic), joka tarkoittaa sitä, että levylle kirjoittaminen joko onnistuu tai epäonnistuu – myös sähkökatkon yhteydessä. Perinteisissä tiedostojärjestelmissä data kirjoitetaan ”lohko kerrallaan”. Tämä vaatii hieman enemmän käsittelyä ja lisäksi journaloinnin sähkökatkon tai muun häiriön varalta. Nämä seikat yhdessä hidastavat kirjoitusoperaatiota. ZFS:ssä näitä tekijöitä ei ole, mikä nopeuttaa kirjoitusoperaatiota. Lisäksi tarve jälkitoimille kuten fsck:lle ja mahdollisen peilatun levyn uudelleensynkronoinnille levyeheyden varmistamiseksi poistuu.

Transaktionaaliseen malliin liittyy läheisesti myös copy-on-write-operaatiot. Käsitteellä tarkoitetaan sitä, että mitään dataa ei muuteta suoraan. Jos data muuttuu, niin muutokset kirjoitetaan aluksi toiseen sijaintiin ja vasta kun kirjoitusoperaatio on valmis, lohko-osoitin (block pointer) muutetaan osoittamaan datan uuteen sijaintiin. Metadatalohkoja käsitellään myös samalla tavalla eli niiden dataa ei ylikirjoiteta missään vaiheessa. Copy-on-write operaatiolla estetään tehokkaasti tiedostojen korruptoituminen ja tietojen katoaminen virhetilanteiden sattuessa kesken kirjoitusoperaation (Oracle Corporation 2010. ZFS Administration Guide).

4.2.3 Kokonaisvaltaiset varmistussummat

ZFS:ssä käytetään kokonaisvaltaista varmistussummaa. Tämä tarkoittaa sitä, että tarkistussummat tallennetaan hierarkiassa ylempään lohko-osoittimeen (parent block pointer) (kuvio 2). Kyseessä on eräänlainen ”tarkistussummapuu”. Tämä ulkoinen tarkistus siis estää huomattavasti tehokkaammin datan korruptoitumista ja siksi apuohjelmia tietojen korjailuun ja palauttamiseen ei tarvita. Hierarkian ylimpänä istuu ”uberblock”, joka käyttää itsevalidoivaa SHA-256 tarkistussummaa. ZFS estää myös järjestelmän panikoinnin, sillä myös metadatalla on omat tarkistussummat!



Kuvio 2: Tarkistussummapuu

Muut tunnetut tiedostojärjestelmät tallentavat tarkistussumman itse datalohkoon, jolloin ne tarjoavat suojan vain bittien lahoamista (bit rot) vastaan.

Lyhyesti siis ZFS tarkistaa koko I/O-polun. Tämä käsittää seuraavat asiat:

- Bittien lahoaminen (bit rot)
- Haamukirjoitukset (phantom writes)
- Virheellisesti kohdistetut luku- ja kirjoitusoperaatiot (misdirected reads and writes)
- DMA pariteettivirheet (DMA parity errors)
- Ajurien virheet (Data tallennetaan väärään puskuriin kernelissä)
- Tahattomat ylikirjoitukset

ZFS:n tehokkaat virheentarkistusalgoritmit tuovat mukanaan myös erittäin tervetulleen ominaisuuden etenkin korkean saatavuustason järjestelmissä, joissa levyt on peilattu. Vikojen paikallistaminen mahdollistaa myös niiden korjaamisen. Perinteisissä peilatuissa järjestelmissä virheellistä dataa saatetaan käyttää ihan kuin mikään ei olisikaan vikana. Tällöin virheellinen data kopioituu myös peilatulle levylle eli ongelma ikään kuin tuplaantuu. ZFS sen sijaan löytää viallisen datan ajoissa ja pystyy korjaamaan sen käyttäen eheän peilauksen versiota datasta. Tämä vähentää ylläpitotarvetta virhetilanteissa ja mahdollistaa korkean saatavuustason ilman käyttökatkoja tai –hidasteita (Oracle Corporation 2010. ZFS Administration Guide).

4.2.4 Skaalautuvuus

ZFS on suunniteltu hyvin skaalautuvaksi ja tästä syystä ZFS:n suunnittelijat päättivät tehdä siitä 128-bittisen. ZFS on ensimmäinen 128-bittinen tiedostojärjestelmä ja mahdollistaa todella suuren tallennuskapasiteetin, joka on kooltaan 256 zettatavua * 10^15. Kaikki metadata allokoidaan dynaamisesti. Tämä helpottaa ylläpitoa, sillä inodeja ei ole pakko määritellä etukäteen. Kaikki algoritmit on kirjoitettu skaalautuvuutta silmälläpitäen. Hakemistoilla voi olla 256 biljoonaa hakemistomerkintää (directory entry), ja tiedostojärjestelmien määrää ei ole rajattu, kuten ei myöskään tiedostojen määrää yksittäisessä tiedostojärjestelmässä. Tarvitaanko sitten tällaista kapasiteettia, sillä ZFS:n tallennuskapasiteetti saattaa tuntua turhankin suurelta? Todellisuudessa se ei sitä kuitenkaan ole. Myös tallennustilan kasvuvauhti voidaan rinnastaa karkeasti Mooren lakiin ja transistorien määrän kasvuun. Tämä tarkoittaa sitä, että n. 10 vuoden kuluttua 64-bittisten tiedostojärjestelmien kapasiteetti käy liian pieneksi, joten ei liene liian aikaista siirtyä pikkuhiljaa 128-bit maailmaan. 128-bittisyys mahdollistaa huomattavan kapasiteetin. Kapasiteettia kuvaa hyvin ZFS-kehitysprojektin pääarkkitehdin Jeff Bonwickin (2004) lausahdus: *"Populating 128-bit file systems would exceed the quantum limits of earth-based storage. You couldn't fill a 128-bit storage pool without boiling the oceans."*

Muita skaalautuvuuteen ja nopeuteen vaikuttavia tekijöitä ovat mm. datan liukuhihnoitus (data pipelining), dynaaminen lohkon koon muuttaminen (dynamic block sizing), älykäs esihaku (intelligent prefetch), dynaaminen raidoitus (dynamic striping) sekä sisäänrakennettu pakkaaminen (built-in compression).

Dynaaminen lohkonhallinta parantaa myös levyn käyttöprosenttia, sillä datalohkojen loppuun jää vähemmän tyhjää tilaa kun lohkon koko vaihtelee tallennettavan tiedon luonteen mukaan (isot tiedostot vs. pienet tiedostot).

Dynaamisella raidoituksella pyritään jakamaan kirjoitus- ja lukuoperaatioiden taakkaa monelle eri fyysiselle laitteelle, jotka on lisätty zpooliin. Yksi tiedosto kirjoitetaan monelle eri levylle pienempinä osina, jolloin levyt toimivat yhtä aikaa ryhmänä. ZFS:ssa lisämääritelmä ”dynaaminen” raidoituskäsitteeseen tulee siitä, että kun zpooliin lisätään uusi fyysinen laite – myös raita ulotetaan myös lisättyyn levyyn automaattisesti. Toisin sanoen ZFS hallitsee raidoitusta dynaamisesti.

Yleisestä mielikuvasta poiketen sisäänrakennetut pakkausalgoritmit nopeuttavat tiedostojärjestelmän toimintaa. Tiedon käsittelyn pullon kaulana on nimittäin kovalevy. Kovalevy ei pysty käsittelemään tietoa yhtä nopeasti kuin prosessori. Kun pakkaus on sisäänrakennettu tiedostojärjestelmään, on se vaivatonta ottaa käyttöön ja jokaista lohkoa kohden kovalevyn lukupään täytyy tehdä vähemmän töitä, sillä muutettavia bittejä on vähemmän pakkauksen ansiosta. Osa työstä siis ”siirretään” prosessorin harteille (Oracle Corporation 2010. ZFS Administration Guide).

4.2.5 ZFS-tilannevedokset

ZFS-tiedostojärjestelmästä pystytään ottamaan nopeasti ja helposti tilannevedos (snapshot). Tilannevedoksen ottaminen on nopea ja kevyt operaatio, sillä dataa ei ottamishetkellä kopioida uuteen sijaintiin vaan siihen luodaan viittaus. Tilannevedos ei vie tilaa ennen kuin dataan tulee muutoksia. Toisin sanoen muuttumaton data jaetaan tiedostojärjestelmän ja sen tilannevedoksen kesken. Sitä voidaan siis esimerkiksi käyttää varmuuskopiointiin. Muutamia päiviä vanhan kopion palauttaminen tiedostojärjestelmästä on helppoa, sillä käytännössä vain muutetaan osoittimet osoittamaan oikeaan sijaintiin. Hidasta kopiointia esimerkiksi nauhavarmistusasemalta ei tarvitse tehdä. Tekniikka on omiaan etenkin lyhytaikaisten varmuuskopioiden ottamiseen (muutaman päivän varmuuskopiot) ja tallennuskapasiteettia säästävä, sillä muuttumattomista tiedoista ei tehdä turhaan tuplakopiota. ZFS tilannevedokset muistuttavat jossain määrin Microsoftin varjokopioita (Shadow Copy).

Samaa tekniikkaa käyttäen voidaan myös luoda kirjoitettavia tilannevedoksia (klooneja), jolloin saadaan aikaiseksi kaksi erillistä tiedostojärjestelmää, jotka jakavat osan datalohkoistaan, mutta joita voidaan käyttää erillisinä yksiköinä (Oracle Corporation 2010. ZFS Administration Guide).

5. SAMBA

5.1 SAMBA:n tarkoitus

Samba on avoimen lähdekoodin ilmainen ohjelmistokokoelma ja tarjoaa tiedostojako- ja tulostuspalveluita SMB ja CIFS asiakkaille. Toisin sanoen Unix/Linux palvelimien resursseja tuodaan Windows-pohjaisten työasemien saataville.

Samban avulla on myös mahdollista integroida Unix/Linux palvelimet Active Directoryyn, mutta sitä tässä opinnäytetyössä ei konfiguroinnin laajuuden takia lähdetty tekemään (Ts, Jay; Eckstein, Robert; Collier-Brown, David 2003: Using Samba, 2nd Edition).

5.2 SAMBAN:n asennus

Opinnäytetyössäni käyttämäni Solaris 10 Update 8:ssa SAMBA kuului käyttöjärjestelmän perusasennukseen, joten sitä ei tarvinnut asentaa erikseen. Ainoat huomioitavat seikat olivat palvelun käynnistäminen ja root-käyttäjän $PATH ympäristömuuttujaan samban binaaritiedostojen /usr/sfw/bin ja /usr/sfw/sbin sijainnin lisääminen (Thomas, Tim 2007. Blogi-kirjoitus: Enabling and configuring SAMBA as shipped with Solaris 10).

Palvelun käynnistäminen tapahtui seuraavalla komennolla:
pfexec svcadm start samba

Tämän jälkeen varmistettiin palvelun tila, kuten kuviosta 3 näkyy.

```
zech@hangar21:~$ svcs | grep samba
online         12:41:26 svc:/network/samba:default
```

Kuvio 3: Samba-palvelun tila

6. Yrityksen tiedostojakoympäristön peruskonfigurointi

6.1 Johdanto

Tässä kappaleessa käyn esimerkin kautta läpi storage-poolin ja tiedostojärjestelmien luomisen sekä itse hakemistojen jakamisen windows-verkkoon samban kautta. Oletusarvona on, että Solaris 10 palvelin on jo asennettu ja konfiguroitu. Tavoitteena on siis tuottaa kuvitteellisen yrityksen ”Omega Corporation” kolmelle eri työntekijälle kotihakemistot ja markkinointi- ja kirjanpito-osastolle omat verkkojaot. Lisäksi tarvitaan yksi verkkojako kaikkien käyttäjien käytettäväksi suurien tiedostojen siirtelyä ja hetkellistä tallentamista varten. Kaikkiin jakoihin asetetaan sopivat tilakiintiöt ja oikeudet (Solaris Internals. ZFS Best Practices Guide) (Oracle Corporation 2010. ZFS Administration Guide).

6.2 ZFS-Storage Pool:ien luominen

Esimerkkinä käytän tässä niin sanottua root poolia, joka luotiin jo Solaris 10 asennuksen yhteydessä. Valitettavasti minulla oli käytössäni vain yksi kovalevy testikokoonpanossa, joten en pystynyt käytännössä testaamaan levyn peilausta tai toisen levyn liittämistä storage pooliin. Alla esimerkissä luodaan ensiksi rpool niminen storage pool ja tämän jälkeen verifoidaan luomisen onnistuminen ja poolin tila.

Luonti tehdään käskyllä:
zpool create rpool c1t0d0

Tarkistetaan tila *zpool list* ja *zpool* status –komennoilla (kuviot 4 ja ja 5).

```
root@hangar21:/rpool/data/omega-corp$ zpool list
NAME    SIZE   USED  AVAIL    CAP  HEALTH  ALTROOT
rpool   149G  6,30G   143G     4%  ONLINE  -
```

Kuvio 4: Zpoolien listaus.

```
root@hangar21:/rpool/data/omega-corp$ zpool status
  pool: rpool
 state: ONLINE
 scrub: none requested
config:

        NAME        STATE     READ WRITE CKSUM
        rpool       ONLINE       0     0     0
          c1t0d0s0  ONLINE       0     0     0

errors: No known data errors
```

Kuvio 5: Zpoolin tilanne.

Mikäli käytettävissä olisi toinenkin kovalevy, niin pooli voitaisiin luoda seuraavalla tavalla, jolloin saataisiin mukaan peilaus aikaiseksi vikasietoisuutta varten:
zpool create rpool c1t0d0 c1t1d0

Tarvittaessa levy voidaan myös lisätä myöhemmin käskyllä:
zpool add rpool mirror c1t0d0 c1t1d0

Toisaalta kovalevy voidaan taas poistaa hetkeksi poolista esimerkiksi vaihtoa varten seuraavalla käskyllä:
zpool detach rpool c1t1d0

6.3 Tiedostojärjestelmien ja oikeuksien konfigurointi

Varataan aluksi kotihakemistojen rpool/export/home isäntätiedostojärjestelmälle 10Gt. Tämä siis varaa heti storage poolista kyseisen määrän tilaa, jota ei voida poolista jakaa muille tiedostojärjestelmille, mutta rpool/export/home voi käyttää enemmänkin tilaa, mikäli uusille käyttäjille luodaan kotikansioita tai vanhojen koko kasvatetaan. Tällöin tulee miettiä tulisiko myös varausta kasvattaa, jotta käyttäjien kotilevylle tallentaminen ei häiriinny, kun kotilevyn kiintiö ei ole vielä täynnä.

zfs set reservation=10G rpool/export/home

Luodaan ensin tiedostojärjestelmä jokaiselle käyttäjälle ja samalla asetetaan pakkaus päälle. Tiedostojärjestelmä mounttaantuu automaattisesti sijaintiin /export/home/<user>.

Asetetaan tiedostojärjestelmän kiintiöksi yksi gigatavu. Kiintiöt ovat muutettavissa samalla käskyllä kuin ne alun perin luodaankin.

pfexec zfs create -o compression=on rpool/export/home/alpha
pfexec zfs set quota=1G rpool/export/home/alpha

Edellisessä kohdassa luodut tiedostojärjestelmät ja niiden kiintiöt (kuvio 6).

```
zech@hangar21:/export/home$ zfs list
NAME                        USED  AVAIL  REFER  MOUNTPOINT
rpool                      6,68G   140G    35K  /rpool
rpool/ROOT                 4,38G   140G    21K  legacy
rpool/ROOT/s10x_u8wos_08a  4,38G   140G  4,38G  /
rpool/data                   21K   140G    21K  /rpool/data
rpool/dump                 1,00G   140G  1,00G  -
rpool/export                820M   140G    23K  /export
rpool/export/home           820M   140G   820M  /export/home
rpool/export/home/alpha    22,5K  1024M  22,5K  /export/home/alpha
rpool/export/home/beta       21K  1024M    21K  /export/home/beta
rpool/export/home/gamma      21K  1024M    21K  /export/home/gamma
rpool/swap                 512M   140G   114M  -
```

Kuvio 6: zfs list-komennon tuloste.

Nyt kun tiedostojärjestelmä on luotu käyttäjää varten, voimme luoda itse käyttäjän ja asettaa kotihakemiston osoittamaan juuri luotuun tiedostojärjestelmämounttiin. Lisäksi asetamme kunkin käyttäjän kotihakemistonsa omistajaksi ja poistamme ylimääräiset oikeudet uteliailta silmiltä. Käyttäjän asettamisen hakemiston (tai oikeastaan tiedostojärjestelmän mount pointin) omistajaksi joutuu vielä toistaiseksi tekemään käsin. Zfs create ja useradd-komennot eivät vielä osaa toimia “yhdessä”, mikä poistaisi turhan hakemiston omistajan määrittelyn vaivan. Normaalia kotihakemistoa luodessa useradd-komento osaa tehdä sen automaattisesti.

pfexec useradd -d /export/home/alpha -s /usr/bin/bash alpha
pfexec chown alpha /export/home/alpha
pfexec chmod 770 /export/home/alpha

Komentojen lopputulos (kuvio 7).

```
zech@hangar21:/export/home$ ls -lg
total 7
drwxrwx---   2 alpha    root            3 Mar  5 15:42 alpha
drwxrwx---   2 beta     root            2 Mar  5 15:20 beta
drwxrwx---   2 gamma    root            2 Mar  5 15:20 gamma
drwx------  32 zech     sysadmin       51 Mar  5 13:17 zech
```

Kuvio 7: Hakemisto-oikeuksien listaus.

Varmistetaan, että oikeudet ovat muuttuneet. Yritetään siis siirtyä alphan kotihakemistoon tai hakea sieltä tiedostolistaus ja todetaan, että se ei onnistu (kuvio 8).

```
zech@hangar21:/export/home$ cd alpha
-bash: cd: alpha: Permission denied
zech@hangar21:/export/home$ ls -lg alpha
alpha unreadable
```

Kuvio 8: Alphan kotihakemiston lukuyrityksien lopputulos.

Käyttäjien kotilevyt ovat nyt luotuna. Seuraavaksi tarvitaan kirjanpidolle ja markkinoille omat tiedostojakonsa. Noudatetaan hierarkkista ja skaalautuvaa ajattelua tässäkin kohtaa mikäli yritys vielä laajenee tai mahdollisesti yhdistää jonkin toisen yrityksen kanssa. Luodaan siis aluksi pohjalle rpool/data-tiedostojärjestelmä, jonka päälle rakennamme hierarkian seuraavilla käskyillä:

pfexec zfs create -o compression=on rpool/data
pfexec zfs create -o compression=on rpool/data/omega-corp
pfexec zfs create -o compression=on rpool/data/omega-corp/kirjanpito
pfexec zfs create -o compression=on rpool/data/omega-corp/markkinointi

Asetetaan kirjanpidon kiintiöksi 10Gt ja markkinoinnin 20Gt.

pfexec zfs set quota=10G rpool/data/omega-corp/kirjanpito
pfexec zfs set quota=20G rpool/data/omega-corp/markkinointi

Käskyjen lopputuloksena näemme neljä uutta tiedostojärjestelmää, joista kahden kiintiöt ovat 10Gt ja 20Gt (kuvio 9).

```
rpool/data                              89K   140G    23K  /rpool/data
rpool/data/omega-corp                   66K   140G    24K  /rpool/data/omega-corp
rpool/data/omega-corp/kirjanpito        21K  10,0G    21K  /rpool/data/omega-corp/
kirjanpito
rpool/data/omega-corp/markkinointi      21K  20,0G    21K  /rpool/data/omega-corp/
markkinointi
```

Kuvio 9: Kirjanpidon ja markkinoinnin tiedostojärjestelmät.

Yrityksellä on myös tarvetta siirrellä tiedostoja sisäverkossa. Luodaan siis vielä yksi tiedostojärjestelmä tätä tarkoitusta varten, mutta kun kyseessä on vain väliaikaiset siirrot, ei levykiintiötä toistaiseksi aseteta. Varsinaisessa tuotantoympäristössä tällainen kannattaisi asettaa, mikäli levytila täyttyy kohtuuttoman nopeasti. Lisäksi Siirto-tiedostojärjestelmän tyhjennys kannattaisi automatisoida tapahtumaan esimerkiksi viikon välein.

pfexec zfs create -o compression=on rpool/data/omega-corp/siirto

Jotta oikeudet saadaan delegoitua oikein, luodaan kolme ryhmää. Markkinointijakoa varten markg, kirjanpidon jakoa varten kirjag ja siirtojakoa varten siirtog.

pfexec groupadd markg
pfexec groupadd kirjag
pfexec groupadd siirtog

Asetetaan hakemisto-oikeudet kuntoon seuraavilla käskyillä:

pfexec chgrp markg /rpool/data/omega-corp/markkinointi
pfexec chgrp kirjag /rpool/data/omega-corp/kirjanpito
pfexec chgrp siirtog /rpool/data/omega-corp/siirto

pfexec chmod 770 /rpool/data/omega-corp/markkinointi
pfexec chmod 770 /rpool/data/omega-corp/kirjanpito
pfexec chmod 770 /rpool/data/omega-corp/siirto

Alpha ja beta ovat markkinoinnin ja gamma on kirjanpidon työntekijöitä. Lisätään käyttäjät asianmukaisiin ryhmiin ja lisäksi jokaiselle käyttäjälle sekundaariseksi ryhmäksi siirtog-ryhmä, jolla saadaan delegoitua oikeudet siirto-jakoon.

pfexec usermod -g markg alpha
pfexec usermod -g markg beta
pfexec usermod -g kirjag gamma

pfexec usermod -G siirtog alpha
pfexec usermod -G siirtog beta
pfexec usermod -G siirtog gamma

6.4 Samban konfigurointi

Kun käyttö- ja tiedostojärjestelmän puolesta konfiguroinnit on tehty, niin saattaaksemme tiedostojaot Windows-koneiden saataville, konfiguroidaan SAMBA toimimaan ”tulkkina” Unix- ja windows-maailman välillä.

Aluksi luodaan halutut käyttäjät samban käyttäjätietokantaan. Jokaisen uuden tilin luonnin yhteydessä syötetään myös salasana. Kirjautuminen Sambajakoon tapahtuu myöhemmin käyttäen alla luotua käyttäjätunnusta ja määriteltyä salasanaa. Tämä on oleellista, sillä myöhemmin määrittelemme sambajaon autentikointimetodiksi käyttäjätunnukset (Ts, Jay; Eckstein, Robert; Collier-Brown, David 2003: Using Samba, 2nd Edition).

pfexec smbpasswd -a alpha
pfexec smbpasswd -a beta
pfexec smbpasswd -a gamma

6.4.1 Globaalit asetukset

Samban konfigurointi tapahtuu pitkälti käyttäen smb.conf tiedostoa, joka sijaitsee /etc/sfw hakemistossa. Seuraavaksi käyn läpi asetukset, jotka määrittelin kotihakemistoille. Aluksi määriteltiin globaalit asetukset. Samban konfiguraatiot ovat opinnäytetyön liitteenä kokonaisuudessaan.

Työryhmän määrittely. Tällä perusteella sambajako liitettiin haluttuun työryhmään domainin puuttuessa:

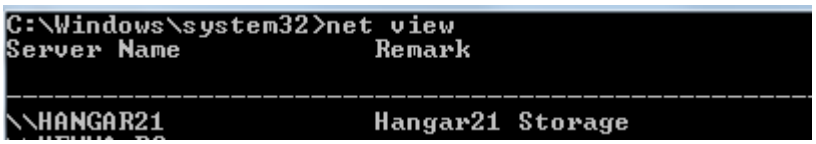
workgroup = WORKGROUP

Salasanat määriteltiin kryptattavaksi:

encypt passwords = yes

Serverin kuvauksen määrittely. Kuviosta 10 ilmenee miten palvelin näkyy kun windows-koneelta ajetaan net view-komento:

server string = Hangar21 Storage



Kuvio 10: Net view-komento

Lokitiedoston sijainti:

log file = /var/adm/samba_log.%m

Turva- ja autentikointi taso/tapa:

security = USER

Tiedosto johon on määritelty mahdolliset käyttäjänimien mappaukset:

username map = /etc/username.map

Käyttöoikeuslistojen tyyppi:

acl compatibility = auto

Serverin netbios nimi:

netbios name = HANGAR21

Vierailijatili Unix puolella mikäli tarvetta yleisille jaoille tulee.

guest account = smbguest

6.4.2 Käyttäjähakemistojen asetukset

Kaikkien kolmen käyttäjän hakemistot on konfiguroitu alla esitetyllä tavalla, jonka jälkeen selvitän konfiguraatioiden merkityksen.

Kommentiksi yleistä muuttujaa käyttäen asetin <nimi> home directory.

comment = %U home directory

Jaon sijainti määriteltiin myös muuttujaa käyttäen. Kun käyttäjä kirjautuu verkkojakoon, haetaan hänen kotihakemistonsa sijainti automaattisesti unix-asetuksista.

path = %H

Kirjoitusoikeudet asetetaan päälle verkkojakoon.

writable = yes

Määritellään validit käyttäjät.

valid users = alpha

Määritellään verkkojakoon luotavan tiedoston oletusmaski.

create mode = 700

Määritellään verkkojakoon luotavan hakemiston oletusmaski.

directory mode = 700

Nämä ”oikeusmaskit” määrittelin arvoon 700 siitä syystä, että kun kotihakemistoon luodaan uusi hakemisto tai tiedosto windowsin tiedostohallinnan kautta, se on todennäköisesti käyttäjän tekemä jolloin käyttäjälle määritellään ks. tiedostoon täydet oikeudet, mutta millekään ryhmälle tai ulkopuoliselle oikeuksia ei myönnetä oletusarvoisesti.

6.4.3 Markkinointi, kirjanpito ja siirto verkkojakojen asetukset

Oleellisena erona käyttäjän kotihakemistoihin, määriteltiin näihin verkkojakoihin valideiksi käyttäjiksi ryhmät ja käyttäjämaskit hieman eri tavalla.

Markkinointi-jako:

valid users = @markg

Kirjanpito-jako:

valid users = @kirjag

Luotavan hakemiston ja tiedoston oletusmaskeiksi määriteltiin 770. Tämä siis tarkoittaa sitä, että itse käyttäjä ja ryhmän jäsenet pääsevät käsiksi uusiin tiedostoihin ja hakemistoihin, mutta muut käyttäjät eivät.

Lisäksi Siirto-jakoon määriteltiin oikeudet kummallekin näistä ryhmistä.
valid users = @markg @kirjag

Luotavan hakemiston ja tiedoston oletusmaskeiksi määriteltiin 775. Tämä siitä syystä, että siirtohakemiston sisältöä saattaa olla tarve tutkia myös muilla autentikoituneilla käyttäjillä. Kyseisellä käyttäjämaskilla annettiin siis luku- ja suoritusoikeudet ”muille”.

7. ZFS:n hallinta ja ylläpito

Tässä kappaleessa käyn läpi oleellisimmat asiat, mitä tulisi tiedostaa ZFS:n hallinnasta ja ylläpidosta (Oracle Corporation 2010. ZFS Administration Guide).

7.1 Luominen, tuhoaminen ja uudelleennimeäminen

Komento tiedostojärjestelmän luomiselle:
zfs create rpool/data/demo

Vaihtoehtoisesti –o parametrilla voidaan määritellä eri ZFS-ominaisuuksia kuten compression=on tai mountpoint=/home/tahan/haluttu/polku.

Poistaminen tapahtuu yksinkertaisesti seuraavalla käskyllä:
zfs destroy rpool/data/demo

Mikäli tiedostojärjestelmällä on “lapsitiedostojärjestelmiä” niin täytyy tuhoaminen tehdä rekursiivisesti käyttäen parametria –r lisänä.

Uudelleennimeäminen tapahtuu seuraavalla komennolla:
zfs rename rpool/data/demo rpool/data/demo-uusi

7.2 ZFS:n ominaisuudet

Jokaisella luodulla tiedostojärjestelmällä on ominaisuudet (property). Näitä käsittelemällä saadaan muutettua tiedostojärjestelmän toimintaa (pakkaus, pakkaussuhde, mount point) tai niiden kautta saadaan haettua tietoja esimerkiksi koosta tai luomispäivästä. Listaus ominaisuuksista saadaan seuraavalla tavalla :

zfs get all rpool/data/demo

Tai vaihtoehtoisesti voidaan hakea jokin haluttu ominaisuus. Kuviosta 11 näkyy miten esimerkiksi pakkauksen ja luomispäivän tila on haettu:

```
zech@hangar21:~$ zfs get compression rpool/data/demo
NAME             PROPERTY     VALUE     SOURCE
rpool/data/demo  compression  off       default
zech@hangar21:~$ zfs get creation rpool/data/demo
NAME             PROPERTY  VALUE                  SOURCE
rpool/data/demo  creation  ti maalis  6 17:11 2012  -
```

Kuvio 11: Tiedostojärjestelmän pakkauksen tila ja luomispäivämäärä

Oracle Solaris ZFS Administration Guide:sta kappaleesta 6, kohdasta: Introducing ZFS Properties – Voidaan nähdä kattava listaus ominaisuuksista.

7.3 Tiedostojärjestelmän jakaminen Unix/Linux järjestelmille

Tiedostojärjestelmät voidaan jakaa helposti muiden Unix/Linux järjestelmien kanssa kontrolloimalla sharenfs-ominaisuutta. Jako on käytössä heti eikä mountd-daemonia tarvitse käynnistää uudelleen.

zfs set sharenfs=on rpool/data/demo

Tämän jälkeen mounttaaminen toisesta Solaris-järjestelmästä onnistuu esimerkiksi käskyllä:

mount -F nfs hangar21.tontut.fi:/rpool/data/demo /mnt

Jaon poisto:

zfs set sharenfs=off rpool/data/demo

7.4 Tilannevedosten käyttäminen

Tilannevedokset eli snapshotit ovat nopea ja helppo tapa palauttaa tiedostojärjestelmä aiempaan tilanteeseen. Seuraavaksi käyn esimerkkitapauksen kautta muutamat oleelliset perusseikat läpi tilannevedosten ottamisesta, palauttamisesta ja poistamisesta. Samalla havainnollistan tilannevedosten toimintaa ja sitä miksi ne ovat kevyitä ja nopeita luoda.

Aluksi luodaan tiedosto rpool/data/demo sijaintiin, jonka jälkeen siitä otetaan tilannevedos.
touch pieni-tiedosto
zfs snapshot rpool/data/demo@test1

Kuviosta 12 näemme, että tiedosto on luotu ja tilannevedos tallennettu.

```
zech@hangar21:/rpool/data/demo$ ls -lg
total 0
-rw-r--r--   1 root     root           0 Mar  6 20:33 pieni-tiedosto
zech@hangar21:/rpool/data/demo$ zfs list -t snapshot
NAME                    USED  AVAIL  REFER  MOUNTPOINT
rpool/data/demo@test1      0      -    22K  -
zech@hangar21:/rpool/data/demo$
```

Kuvio 12: Hakemiston ja tilannevedoksen tilanne

Luodaan suurempi tiedosto, jonka jälkeen otetaan toinen tilannevedos. Tällä hetkellä tilannevedokset eivät vielä vie varsinaisesti tilaa. Demo@test2-tilannevedos osoittaa tällä hetkellä vielä muuttumattoman tiedoston sijaintiin. Kuvio 13 osoittaa tilanteen näiden käskyjen jälkeen.

mkfile 10m iso-tiedosto
zfs snapshot rpool/data/demo@test2

```
zech@hangar21:/rpool/data/demo$ ls -lg
total 10243
-rw------T   1 root     root     10485760 Mar  6 20:37 iso-tiedosto
-rw-r--r--   1 root     root            0 Mar  6 20:33 pieni-tiedosto
zech@hangar21:/rpool/data/demo$ zfs list -t snapshot
NAME                    USED  AVAIL  REFER  MOUNTPOINT
rpool/data/demo@test1    19K      -    22K  -
rpool/data/demo@test2      0      -  10,0M  -
zech@hangar21:/rpool/data/demo$
```

Kuvio 13: Hakemiston ja tilannevedosten tilanne

Nyt testiksi poistetaan iso-tiedosto. Tarkastellaan tämän jälkeen tilannevedosten listaa ja huomataan, että demo@test2 vie nyt 10Mt:a tilaa kovalevyltä kuten kuviosta 14 näkyy.

rm iso-tiedosto

```
zech@hangar21:/rpool/data/demo$ zfs list -t snapshot
NAME                    USED  AVAIL  REFER  MOUNTPOINT
rpool/data/demo@test1    19K      -    22K  -
rpool/data/demo@test2  10,0M      -  10,0M  -
zech@hangar21:/rpool/data/demo$
```

Kuvio 14: Tilannevedosten levytilan käyttö

Palautetaan tilannevedos demo@test2. Kuviosta 15 nähdään kuinka tiedosto on palautunut ja demo@test2 ei vie enää tilaa kovalevyltä.

zfs rollback rpool/data/demo@test2

```
zech@hangar21:/rpool/data/demo$ ls -lg
total 10243
-rw------T   1 root     root     10485760 Mar  6 20:37 iso-tiedosto
-rw-r--r--   1 root     root            0 Mar  6 20:33 pieni-tiedosto
zech@hangar21:/rpool/data/demo$ zfs list -t snapshot
NAME                    USED  AVAIL  REFER  MOUNTPOINT
rpool/data/demo@test1    19K      -    22K  -
rpool/data/demo@test2      0      -  10,0M  -
zech@hangar21:/rpool/data/demo$
```

Kuvio 15: Hakemiston ja tilannevedoksen tilanne palautuksen jälkeen

Poistetaan ylimääräiset tilannevedokset:

zfs destroy rpool/data/demo@test1
zfs destroy –r rpool/data/demo@test1

Tilannevedosten hyödyntämisen varmuuskopiointiin olen rajannut opinnäytetyöstä pois. Tämä vaatisi kloonien luomisen tilannevedoksista, automaattisten tilannevedosten ajoittamisen ja niiden lähettämisen haluttuun varmuuskopiointisijaintiin. Nämä seikat lisäisivät huomattavasti perehtymisen tarvetta ja konfiguroinnin määrää. Lisäksi, jotta asia saataisiin hieman suuremman yrityksen tarpeisiin räätälöityä, se vaatisi perehtymistä kolmannen osapuolen varmuuskopiointijärjestelmiin.

8. Yhteenveto

Opinnäytetyö osoittautui loppujen lopuksi hyvinkin laajaksi, joten rajauksia täytyi tarkentaa useampaan otteeseen opinnäytetyön varrella niin, että ydinasiat olisivat päärooolissa. Esimerkiksi oikeusmäärittelyt tehtiin POSIX-standardin mukaisilla pääsylistoilla laajemman NFSv4-pääsylistojen sijaan.

Lisäksi opinnäytetyöhön liittyen piti tehdä yllättävän paljon ”näkymätöntä työtä”, jotta ympäristöä päästiin ylipäätänsä rakentamaan. Solaris 10:n asentaminen, verkkoyhteysongelmien ratkominen, muuttujien ja shellin/profiilien konfigurointi, etäyhteyksien avaaminen ja muut käyttöjärjestelmään liittyvät toimenpiteet veivät huomattavasti aikaa itse aiheelta.

Solaris 10 erosi monella tapaa Linux-distribuutioista, joita allekirjoittanut oli aiemmin käyttänyt. Suurimpina eroina esimerkiksi ohjelmien asennukset, konfiguraatiotiedostojen sijainnit ja palveluiden hallinta, jotka olivat oleellisia asioita myös ZFS:n ja SAMBAn kannalta. Käytännössä siis puolet opinnäytetyöhän käytetystä ajasta kului uuden käyttöjärjestelmän opetteluun. Tähän perustuen en siis ensitöikseni lähtisi suosittelemaan opinnäytetyössä esitettyä ratkaisua yritykseen, jossa ei ole Unix (tai Solaris) osaamista tai jo valmista Unix-ympäristöä. Mikäli ”ennakkovaatimukset” täyttyvät, niin on erittäin suositeltavaa siirtyä käyttämään ZFS:ää ainakin osassa tiedostojakoja. Riippumatta siitä onko niitä tarkoitus käyttää Unix- tai Windows-ympäristössä. Pelkästään jo se että volume managereita ei enää tarvita niiden aiemmassa merkityksessä, on todella suuri helpotus ylläpidolle. Tämä ilmenee hyvin aiemmin opinnäytetyössä esitetyssä esimerkissä jossa konfiguroidaan uudet levyt ja tiedostojärjestelmät uusille käyttäjille ensin Solaris Volume Manageria käyttäen (UFS) ja sitten ZFS:n ”volume manager”-työkaluja käyttäen.

Opinnäytetyö oli onnistunut siinä mielessä, että se tuo esille ZFS:n vahvuudet ja muistuttaa siitä, että vaikka tiedostojärjestelmä onkin Unix/Linux käyttöjärjestelmille, niin sitä voidaan hyödyntää myös Windows-ympäristöissä. Lisäksi opinnäytetyö antaa hyvin osviittaa siihen millaista osaamista vaaditaan, jotta ZFS ja SAMBA olisivat käypä ratkaisu tiedostojen jakamiseen.

LÄHTEET

Bonwick, Jeff 2004. Blogi-kirjoitus: 128-bit storage: Are you high? [online][viitattu 4.3.2012]. https://blogs.oracle.com/bonwick/entry/128_bit_storage_are_you

Hunter, Jeff 2010. Solaris Volume Manager - (Solaris 9): Creating Volumes http://www.idevelopment.info/data/Unix/Solaris/SOLARIS_CreateVolumes_VolumeManager9Commands.shtml

Oracle Corporation 2010. ZFS Administration Guide: http://docs.oracle.com/cd/E19253-01/819-5461/index.html

Oracle Corporation 2010. Solaris Volume Manager Administration Guide. http://docs.oracle.com/cd/E19683-01/817-2530/6mi6gg86c/index.html

Ts, Jay; Eckstein, Robert; Collier-Brown, David 2003: Using Samba, 2nd Edition http://www.samba.org/samba/docs/using_samba/toc.html

Solaris Internals. ZFS Best Practices Guide. http://www.solarisinternals.com/wiki/index.php/ZFS_Best_Practices_Guide

Thomas, Tim 2007. Blogi-kirjoitus: Enabling and configuring SAMBA as shipped with Solaris 10. [online][viitattu 6.3.2012]. https://blogs.oracle.com/timthomas/entry/enabling_and_configuring_samba_as

Wikipedia 2011. Unix File System http://en.wikipedia.org/wiki/Unix_File_System

Wikipedia 2011. Comparison of file systems. http://en.wikipedia.org/wiki/Comparison_of_file_systems

LIITTEET

/etc/swf/smb.conf

```
[global]

workgroup = WORKGROUP
encypt passwords = yes
server string = Hangar21 Storage
log file = /var/adm/samba_log.%m
security = USER
username map = /etc/username.map
acl compatibility = auto
netbios name = HANGAR21
guest account = smbguest

[alpha]
comment = %U home directory
path = %H
writable = yes
valid users = alpha
create mode = 700
directory mode = 700

[beta]
comment = %U home directory
path = %H
writable = yes
valid users = beta
create mode = 700
directory mode = 700

[gamma]
comment = %U home directory
path = %H
writable = yes
valid users = gamma
create mode = 700
directory mode = 700
```

```
[markkinointi]
comment = Markkinoinnin verkkojako
path = /rpool/data/omega-corp/markkinointi
writable = yes
valid users = @markg
create mode = 770
directory mode = 770

[kirjanpito]
comment = Kirjanpidon verkkojako
path = /rpool/data/omega-corp/kirjanpito
writable = yes
valid users = @kirjag
create mode = 770
directory mode = 770

[siirto]
comment = Tiedostojen siirto
path = /rpool/data/omega-corp/siirto
writable = yes
valid users = @markg @kirjag
create mode = 775
directory mode = 775
```