Módulo 8

Sistema Operacional

Lição 7

ZFS

Versão 1.0 - Mar/2008

Autor
-

Equipe
Rommel Feria
John Paul Petines

Necessidades para os Exercícios

Sistemas Operacionais Suportados

NetBeans IDE 5.5 para os seguintes sistemas operacionais:

- Microsoft Windows XP Profissional SP2 ou superior
- Mac OS X 10.4.5 ou superior
- Red Hat Fedora Core 3
- Solaris™ 10 Operating System (SPARC® e x86/x64 Platform Edition)

NetBeans Enterprise Pack, poderá ser executado nas seguintes plataformas:

- Microsoft Windows 2000 Profissional SP4
- Solaris™ 8 OS (SPARC e x86/x64 Platform Edition) e Solaris 9 OS (SPARC e x86/x64 Platform Edition)
- Várias outras distribuições Linux

Configuração Mínima de Hardware

Nota: IDE NetBeans com resolução de tela em 1024x768 pixel

Sistema Operacional	Processador	Memória	HD Livre
Microsoft Windows	500 MHz Intel Pentium III workstation ou equivalente	512 MB	850 MB
Linux	500 MHz Intel Pentium III workstation ou equivalente	512 MB	450 MB
Solaris OS (SPARC)	UltraSPARC II 450 MHz	512 MB	450 MB
Solaris OS (x86/x64 Platform Edition)	AMD Opteron 100 Série 1.8 GHz	512 MB	450 MB
Mac OS X	PowerPC G4	512 MB	450 MB

Configuração Recomendada de Hardware

Sistema Operacional	Processador	Memória	HD Livre
Microsoft Windows	1.4 GHz Intel Pentium III workstation ou equivalente	1 GB	1 GB
Linux	1.4 GHz Intel Pentium III workstation ou equivalente	1 GB	850 MB
Solaris OS (SPARC)	UltraSPARC IIIi 1 GHz	1 GB	850 MB
Solaris OS (x86/x64 Platform Edition)	AMD Opteron 100 Series 1.8 GHz	1 GB	850 MB
Mac OS X	PowerPC G5	1 GB	850 MB

Requerimentos de Software

NetBeans Enterprise Pack 5.5 executando sobre Java 2 Platform Standard Edition Development Kit 5.0 ou superior (JDK 5.0, versão 1.5.0_01 ou superior), contemplando a Java Runtime Environment, ferramentas de desenvolvimento para compilar, depurar, e executar aplicações escritas em linguagem Java. Sun Java System Application Server Platform Edition 9.

- Para **Solaris**, **Windows**, e **Linux**, os arquivos da JDK podem ser obtidos para sua plataforma em http://java.sun.com/j2se/1.5.0/download.html
- Para **Mac OS X**, Java 2 Plataform Standard Edition (J2SE) 5.0 Release 4, pode ser obtida diretamente da Apple's Developer Connection, no endereço: http://developer.apple.com/java (é necessário registrar o download da JDK).

Para mais informações: http://www.netbeans.org/community/releases/55/relnotes.html

Colaboradores que auxiliaram no processo de tradução e revisão

Aécio Júnior
Alberto Ivo da Costa Vieira
Alexandre Mori
Alexis da Rocha Silva
Allan Wojcik da Silva
Antonio José Rodrigues Alves Ramos
Angelo de Oliveira
Bruno da Silva Bonfim
Carlos Fernandes Gonçalves
Denis Mitsuo Nakasaki
Felipe Gaúcho
Jacqueline Susann Barbosa
João Vianney Barrozo Costa
Luiz Fernandes de Oliveira Junior
Marco Aurélio Martins Bessa
Maria Carolina Ferreira da Silva
Massimiliano Giroldi
Paulo Oliveira Sampaio Reis
Ronie Dotzlaw
Seire Pareja
Thiago Magela Rodrigues Dias
Vinícius Gadis Ribeiro

Auxiliadores especiais

Revisão Geral do texto para os seguintes Países:

- **Brasil** – Tiago Flach
- **Guiné Bissau** – Alfredo Cá, Bunene Sisse e Buon Olossato Quebi – ONG Asas de Socorro

Coordenação do DFJUG

- **Daniel deOliveira** – JUGLeader responsável pelos acordos de parcerias
- **Luci Campos -** Idealizadora do DFJUG responsável pelo apoio social
- **Fernando Anselmo -** Coordenador responsável pelo processo de tradução e revisão, disponibilização dos materiais e inserção de novos módulos
- **Rodrigo Nunes -** Coordenador responsável pela parte multimídia
- **Sérgio Gomes Veloso -** Coordenador responsável pelo ambiente JEDI™ (Moodle)

Agradecimento Especial

John Paul Petines – Criador da Iniciativa JEDI™
Rommel Feria – Criador da Iniciativa JEDI™

Original desta por – McDougall e Mauro – Solaris Internals. Sun Microsystems. 2007.

1. Objetivos

ZFS foi desenvolvido pela equipe de *Jeff Bonwick* como um novo sistema de arquivo para o **Solaris**. Seu desenvolvimento foi anunciado em setembro de 2004, foi incluído como parte da distribuição oficial do **OpenSolaris** em novembro de 2005 e distribuído com a atualização 6/06 do Solaris 10 em junho de 2006.

ZFS promete para ser a "última palavra em sistemas de arquivos". Desenvolvedores de **ZFS** o descrevem na página WEB da comunidade Solaris como "um novo tipo de sistema de arquivo que provê uma administração simples, semântica transacional, integridade de dados *end-to-end* e melhor escalabilidade"[1].

Discutiremos nesta lição estas características do ZFS e mostraremos suas melhorias sobre o sistema de arquivo atualmente em uso.

Ao final desta lição, o estudante será capaz de:

- Compreender o conceito de sistema de arquivos
- Utilizar o aplicativo ZFS
- Realizar agrupamento, espelhamento, fotografias e clones no disco de sistema Solaris

1 http://www.opensolaris.org/os/community/zfs/ publicado em 21 de agosto de 2007.

2. Capacidade

A capacidade de um sistema de arquivo pode ser descrita pelo número de *bits* em uso pelo sistema que é utilizado para armazenar informações sobre de arquivos. Atualmente os sistemas de arquivo são de **64 *bits***. Isso significando que são usados 64 *bits* de dados para armazenar a informação de cada arquivo, como local para o dispositivo, permissões de arquivo, conteúdos de diretório, entre outros. Sendo que um sistema de arquivo de 64 *bits* denota um arquivo de tamanho máximo teórico de sistema com 2^{64} *bytes,* temos aproximadamente 10^{11} Gb ou 1 bilhão de discos rígidos (de 100 Gb cada).

Este é um tamanho máximo teórico. A seguir temos as limitações de sistema de arquivos para cada sistema conhecido.

Sistema de Arquivo	Sistema Operacional	Capacidade
FAT16	DOS-Win3.11	2 Gb
FAT32	Win95-WinME	8 Tb (ou 8000 Gb)
ext3	Linux	32 Tb (32000Gb)
NTFS	WinXP-Vista	2 Tb (2000 Gb)

Figura 1: Capacidade teórica dos sistemas de arquivos

Isto pode parecer bastante grande, entretanto atualmente existem algumas companhias que possuem dados na escala de *petabyte*, aproximadamente 10^6 no valor de GB de espaço armazenado. Se a tendência na Lei de *Moore* continuar, então, na metade da próxima década podemos ver que o limite de 64 *bits* será suplantado[1]. Realmente o armazenamento mundial já excedeu os 161 *exabytes* (10^{18} bytes) e alcançará 988 *exabytes* antes de 2010[2]. Nosso armazenamento mundial já foi além dos 10^{11} Gb do limite teórico de sistemas de 64 *bits*, ou seja, um bilhão de vezes acima.

ZFS é o primeiro sistema de arquivos de 128 *bits*. Sendo que um sistema de arquivos de 128 *bits* que pode armazenar teoricamente 2^{128} *bytes*. Isto corresponde a 10^{24} Gb, ou seja, um trilhão de trilhão de *gigabytes* ou 10.000 vezes maior do que o tamanho total de armazenamento mundial. Os desenvolvedores usam limites físicos de computação para armazenar 128-bits completos em um sistema de arquivo, um usuário iria necessitar de um dispositivo de armazenamento com uma massa mínima de 136 bilhões quilogramas, e isso seria energia mais do que suficiente para ferver os oceanos.

1 http://blogs.sun.com/bonwick/entry/128_bit_storage_are_you de 22 de agosto de 2007.

2 Cooporação EMC. The Expanding Digital Universe: A Forecast of Worldwide Information Growth Through 2010

3. Armazenamento Agrupado

3.1. O novo problema do disco rígido

Suponhamos a situação de um computador com um longo tempo de vida, este está conectado à *internet*, provavelmente, ocorreria o problema de falta de espaço em disco. Considerando a compra de um novo disco rígido, podemos adotar duas estratégias:

Como uma primeira estratégia configuramos o novo dispositivo como um disco rígido secundário, (como um novo *drive* para o sistema operacional **Windows** ou montado como um novo diretório em sistemas do tipo Unix, tais como **Linux** e **Solaris**). Poderíamos colocar grandes arquivos ou aplicações neste disco rígido novo. E, se este for o caso, então os usuários de computador devem lembrar do *drive* para guardar os arquivos. As aplicações começariam a manipular diferentes diretórios e seu *filesystem* ficaria completamente desorganizado.

Para evitarmos esse problema descrito, a segunda estratégia consistiria em copiar o conteúdo do antigo disco rígido para o novo (ou, pelo menos, os documentos do usuário). Porém, deste modo teríamos alguns problemas. Se o usuário ou uma aplicação não puder localizar um determinado arquivo porque foi movido para outro diretório (nenhum problema para computadores com **Linux** ou **Solaris**). Entretanto, e se alguns arquivos fossem deixados para trás?

No fim, acrescentar um novo disco rígido no seu computador causa uma grande quantidade de dores de cabeça devido a uma nova reorganização.

3.2. A resposta através de ZFS: Armazenamento agrupado

Em outros sistemas de arquivo existe um único dispositivo, e meios de dispositivos múltiplos que possuem um **Gerenciador de Volume**. Gerenciadores de volume mostram os dispositivos diferentes como "*drives*" em sistemas **Windows**. Em sistemas **Linux**, o diretório raiz atravessa todos os dispositivos, e estes dispositivos (ou diretórios nesses dispositivos) podem ser montados como diretórios no diretório de raiz.

Por outro lado, **ZFS** usa um sistema de armazenamento agrupado. Durante a organização de um agrupamento de **ZFS**, é possível nomear um, dois ou mais dispositivos a este. Toda a capacidade de todos os dispositivos é acessível pelo agrupamento.

Este agrupamento permite que seja montado como um diretório no *filesystem* regular do Solaris. Do ponto de vista do usuário, estará economizando um diretório no sistema de Solaris. Porém, internamente, podem ser economizados dados em um dispositivo e podem ser estendidos ao próximo dispositivo se houver espaço faltando. Isto pode ser refletido até mesmo para outro disco ou ser distribuído para vários discos rígidos obtendo uma maior redundância. O usuário deve apenas se preocupar onde e como o dados são armazenados.

Para colocar isto de forma simples, "ZFS faz com o armazenamento o que a memória virtual fez com a RAM"[1]. Como visto em lições anteriores, a memória virtual pode ser resumida em: detalhes de aplicações e como elas são armazenadas. Uma aplicação não sabe onde está em memória, se foi alocada em memória contínua ou não. De fato, está na memória, entretanto pode estar temporariamente armazenada no disco rígido. Ocorre o mesmo com arquivos **ZFS**. O usuário não necessita se preocupar como o arquivo é armazenado de fato, a menos que este diretório possa ser acessado.

Adicionar um novo dispositivo ao ZFS significa que estamos adicionando no agrupamento do armazenamento. O computador irá automaticamente pegar esta nova capacidade disponível e usá-lo sem a necessidade de transferir ou reorganizar o *filesystem*. É possível ter no máximo 264 dispositivos em um único agrupamento.

1 Jeff Bonwick. ZFS: The Last Word in File Systems. Sun Microsystems

4. Integridade de Dados

Armazenamento secundário está longe de ser um meio seguro de manter seus dados intactos. Problemas podem ocorrer e destruir a informação a qualquer momento.

A perda de *bits* acontece quando se desgasta o meio magnético, seu disco rígido começa a falhar devido a este desgaste. Pode ocorrer o que denominamos de **fantasma**. Um fantasma pode escrever e o disco rígido reivindicar a escrita com dados que de fato não existem. Resultado que pode acidentalmente tentar ler ou escrever dados em uma porção errada do disco. Isto acaba tornando áreas completas do disco ilegíveis.

ZFS tem diversas características que mantêm a integridade dos dados.

4.1. Checksums end-to-end[1]

Sistema de arquivo armazenam informação em blocos. Sistemas de arquivo tradicionais acrescentam blocos em *checksums* aos arquivos e estes blocos podem prover correções de erro. Isto é, podem descobrir trechos com problemas de fantasmas, como o que acontece quando os dados e o *checksum* não são compatíveis.

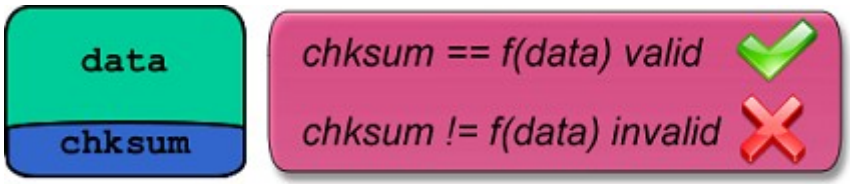

Figura 2: Dados não emparelhados com o checksum

Porém, o que acorre quando são colocados os dados corretos com o *checksum* correto na porção errada do disco rígido? Na realidade, este sistema pode descobrir um único pedaço com problema de fantasmas e nada mais.

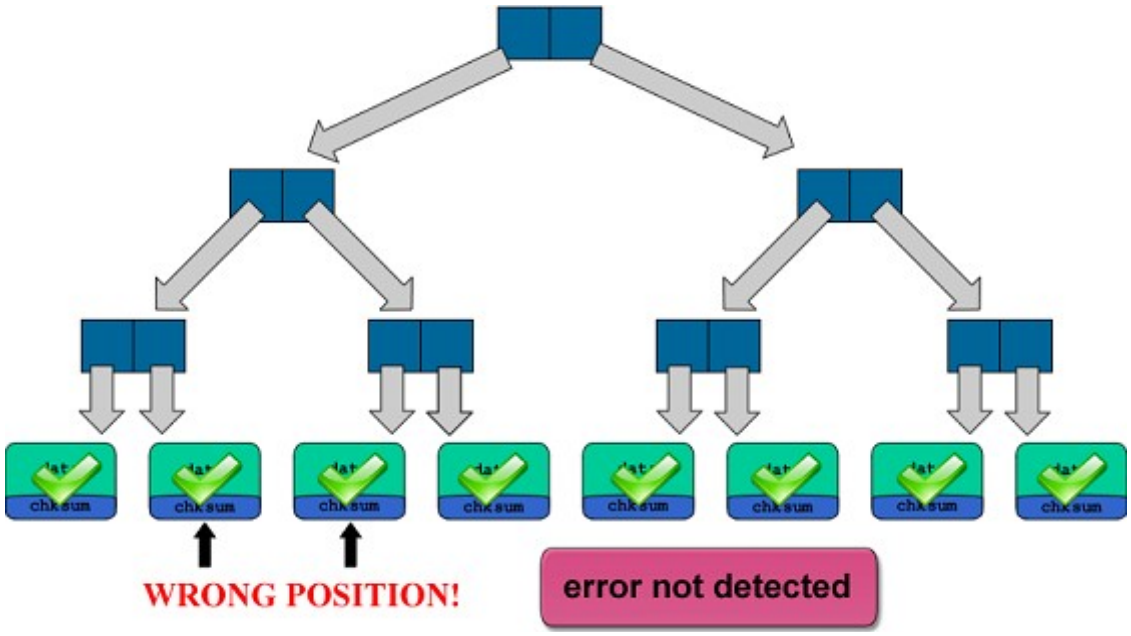

Figura 3: Pedaço errado do sistema de arquivos

ZFS vai um passo adiante e coloca um *checksum* em cada nível do bloco.

Inserindo *checksums* em todo o caminho até o bloco pai não só asseguramos que cada dado é bloqueado de forma consistente, mas também mantemos um agrupamento inteiro de forma

1 **Sum**mation **Check** - verificação do numérica dos *bits* que estão sendo transferidos para descobrir erros na transferência

consistente. Qualquer operação que resulte em qualquer lugar em um *checksum* errado ao longo dos meios da árvore indica que o agrupamento inteiro está de alguma forma incompatível e necessita de correção.

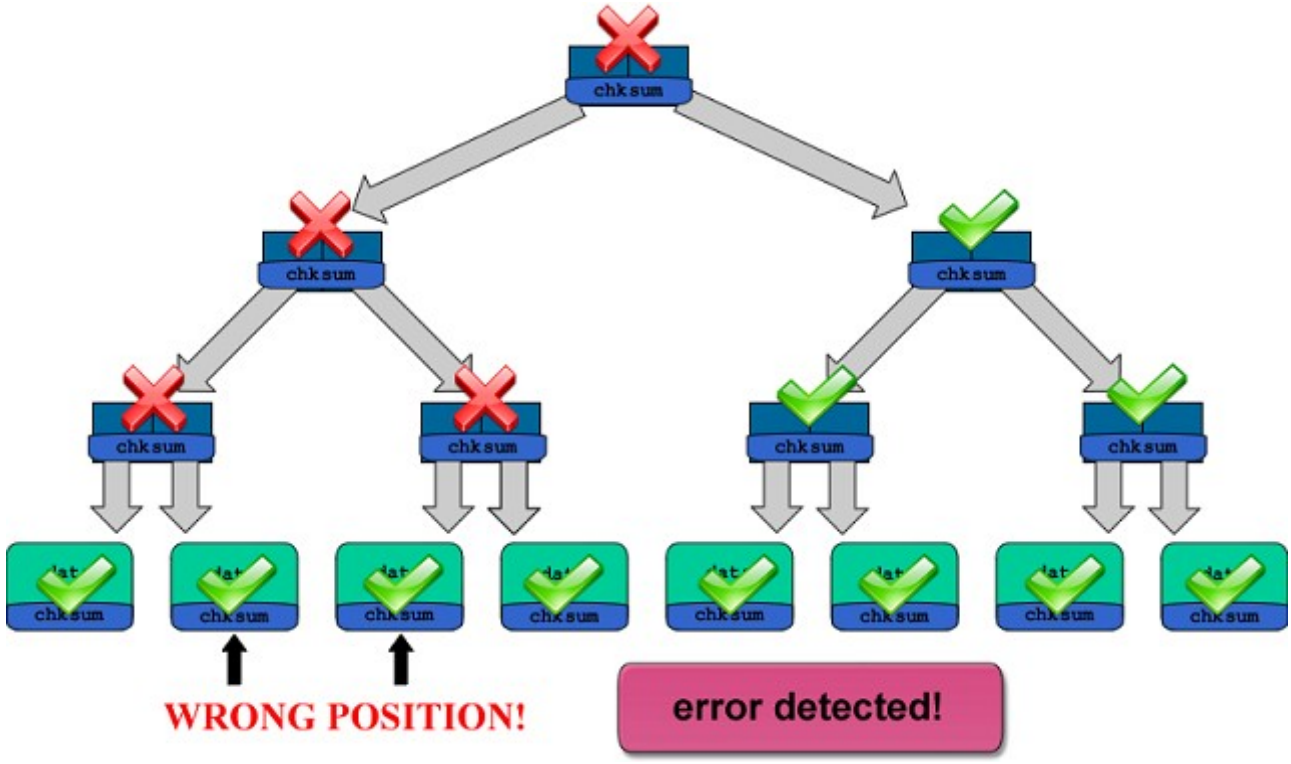

Figura 4: Agrupamento inválido

E ainda, **ZFS** separa os dados do *checksum*. Em sistemas de arquivo tradicionais, onde os dados e o *checksum* são armazenados no mesmo bloco, existe uma chance de um erro se o disco rígido modificar tanto os dados quanto o *checksum* de forma que o sistema não pode mais determinar se ocorreu algum erro. Fisicamente separados, dados e *checksum*, erros no disco rígido irão afetar somente os dados ou o *checksum*, sendo fácil de serem detectados.

4.2. Limpeza do disco

Para assegurar que os dados continuem consistentes, **ZFS** faz uma verificação continua de todos os blocos de dados em um processo conhecido como "Limpeza do Disco" (*disk scrubbing*). **ZFS** percorre o disco inteiro verificando se os dados combinam com os respectivos *checksums*. Em caso de erros, **ZFS** está apto a corrigir a informação automaticamente, derivando a partir do *checksum*, ou através de um espelho.

4.3. Transações de escrita e leitura na entrada e saída

Já aconteceu que ao salvar um documento (normalmente muito importante) ocorre uma falha de energia? Para nosso horror, descobrimos que o arquivo do documento foi corrompido e devemos refazê-lo do zero.

O arquivo foi corrompido porque o arquivo foi deixado em um estado inconsistente. O arquivo consiste de diversos blocos de dados de uma nova versão assim como os dados da antiga versão, os quais deveriam ser sobrescritos se não ocorresse a falha de energia.

Alguns sistemas tradicionais de arquivo fazem uso de um sistema de notícias (*journal*). Todas as operações de entrada e saída são escritas primeiramente neste sistema de publicação antes de serem executadas. Isso garante que em caso de falha de energia ou outro erro, o sistema simplesmente continue suas operações escrevendo no arquivo de publicação antes que se torne inconsistente novamente. As operações devem ser **atômicas**. Ou todas são executadas com sucesso, ou são novamente executadas a partir do arquivo de publicação em caso de estar

corrompido, ou não são executadas.

Fazer uso do sistema de publicação, entretanto, diminui a velocidade da execução de entrada e saída devido ao passo extra de se tomar nota de todas as instruções que serão executadas. Após um longo período de tempo, o arquivo de publicação ocupará um espaço significante no disco rígido.

ZFS dá um passo extra para implementar este tipo de transação. Nenhum dado está de fato sendo sobrescrito pelo **ZFS**, quaisquer modificações do sistema toma o lugar de uma cópia dos dados. Qualquer falha de sistema irá afetar somente a cópia dos dados. No caso de uma falha, o sistema recupera os blocos de dados originais antes de executar as operações. Apenas quando as modificações atingem o bloco raiz é que ocorre a gravação e conclusão das modificações no sistema. Ou todas as instruções de entrada e saída agrupadas são executadas ou a transação não acontece.

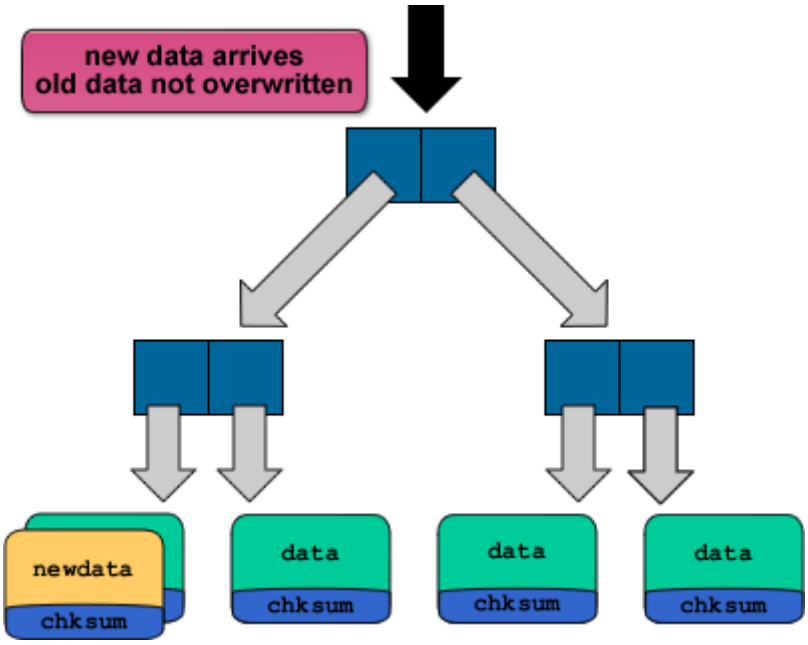

Figura 5: Novos dados chegam e os antigos não são alterados

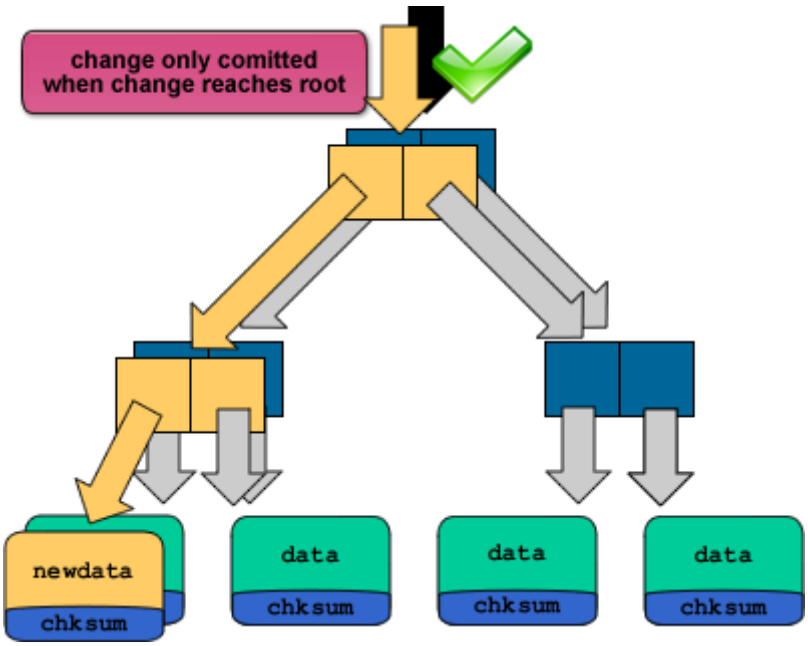

Figura 6: Modificações são realmente gravadas

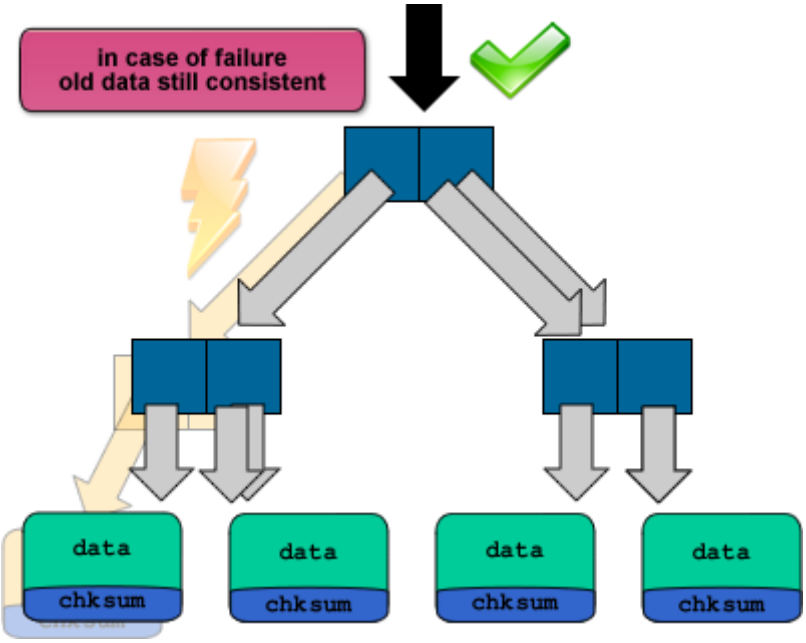

Figura 7: Em caso de falha permanece os dados consistentes

4.4. Fotografias de tempo linear

Como efeito colateral dos sistemas de escrita e leitura, fotografias de sistema de arquivo são automaticamente realizadas depois de qualquer operação do sistema. Fotografias são uma cópia do sistema em algum momento no passado as quais podem ser usadas para propósitos de *backup*. Cada operação em **ZFS** automaticamente cria uma fotografia do antigo sistema. De fato é mais rápido criar uma fotografia do que ter aquele passo extra de substituir os dados antigos.

5. Espelhamento e RAID-Z

5.1. Espelhamento

ZFS possibilita, sem muito esforço, configurar um sistema de arquivo espelhado. Um sistema de arquivo espelhado usa um segundo disco rígido para replicar completamente os dados de um outro disco rígido. Espelhos são freqüentemente usados se o primeiro disco rígido falhar, o sistema continua operando com dados contidos no segundo disco rígido. Além disso, espelhos tornam a leitura duas vezes mais rápida, os dados podem ser retornados do segundo disco rígido enquanto o primeiro estiver ocupado.

Implementações de espelhamento tradicionais não diferenciam blocos ruins. Mesmo que uma cópia *backup* exista no espelho, isso não possibilita dizer se um bloco foi, de algum modo, corrompido. Ao fazer o *checksum* de todos os blocos, o **ZFS** determina silenciosamente se um bloco foi corrompido. Se isso ocorre, o **ZFS** recupera automaticamente o bloco correto do segundo disco e repara o bloco ruim do primeiro disco. Isso é feito de modo transparente, sem informar ao usuário qualquer problema.

5.2. RAID-Z

RAID é uma sigla para *Redundant Array of Inexpensive Disks* (Conjunto Redundante de Discos Baratos). Configurar um sistema **RAID** significa adicionar novos discos rígidos, seguindo um esquema particular (chamado nível **RAID**) de como a informação será armazenada nestes discos.

Há 5 níveis tradicionais de RAID:

- **RAID 0 –** os dados são simplesmente colocados em trilhas de múltiplos discos. Uma trilha consiste de vários blocos agrupados de dados. Fornece a vantagem de melhor *performance*, pois as leituras podem ser feitas em paralelo. No entanto, não provê nenhum tipo de segurança ao dado, uma falha no disco resulta em falha de todos os dados.
- **RAID 1 –** os dados são espelhados ou completamente duplicados em um segundo disco, ou múltiplos. Leituras podem também ser feitas em paralelo, assim como prover segurança aos dados já que se um disco falhar os dados podem ainda ser lidos de um segundo disco. Entretanto, esse nível de **RAID** é caro, necessitamos ter o dobro da quantidade de disco rígido para armazenar os dados.
- **RAID 2 –** os dados são escritos um **bit** por vez em cada disco, com um disco dedicado especialmente para armazenar a informação de paridade para a recuperação desses dados. Esse nível não é usado já que é inviável armazenar **bit** a **bit**.
- **RAID 3 –** assim como o RAID 2, divide os dados entre discos com uma paridade de disco especial. Entretanto, os dados dessa vez são divididos em trilhas.
- **RAID 4** – divide os dados em blocos de dados do sistema de arquivo ao invés de colocar os blocos em trilhas de múltiplos discos. Um disco de paridade dedicado contém informação que pode ser usada para detectar e corrigir erros de dados nos discos.

O problema com o **RAID 4** é que qualquer escrita feita em um bloco de dados significa ter que novamente computar a paridade. Isso significa que qualquer escrita irá envolver o processo de duas escritas, a escrita dos dados e a da uma nova paridade no bloco. Isso causa um gargalo na paridade de disco. Um novo tipo denominado de **RAID 5** distribui a paridade de disco e permite um tempo mais rápido de escrita.

O problema com **RAID 4** e **5** é que sempre que qualquer bloco de dados for escrito, outros blocos na mesma trilha devem ser lidos para recalcular o bloco de paridade. E isso diminui a velocidade de escrita. Os dados podem se tornar corrompidos se uma falha de energia ocorrer enquanto o bloco de paridade estiver sendo computado. Como o novo bloco de paridade ainda não foi escrito corretamente, então os blocos de dados não batem com o bloco de paridade. Quando isso ocorre, **RAID** assume incorretamente que o dado está corrompido.

ZFS inclui o modelo **RAID-Z**. **RAID-Z** é uma implementação modificada do **RAID 5**. Este novo

modelo configura cada bloco de dados do sistema de arquivo para ser sua própria trilha. Dessa forma, para se recalcular a paridade, é preciso ter que carregar somente um único bloco de dados, não é mais necessário ler qualquer outro bloco. E como as operações do sistema de arquivo são agora baseadas em transações, a questão de paridade é evitada, ou o bloco inteiro (incluindo a paridade) é escrito, ou ainda, não é escrito de forma alguma.

***Striping* dinâmico** também é uma característica adicional de RAID-Z. Antigas implementações de RAID faziam que, uma vez fixado o número de discos rígidos, quando o sistema era organizado, sua faixa de tamanho também era. Quando um novo disco for adicionado, todos os novos dados são acertados para usar este novo disco. Não existe nenhuma necessidade de se migrar os antigos dados, com o passar do tempo estes dados migram para o novo formato de faixa. Esta migração é feita automaticamente.

6. Administração ZFS

6.1. Convenção de nomes do disco

Antes de podermos discutir a administração do **ZFS**, devemos nos familiarizar primeiro com a notação de disco usada no sistema Solaris.

Todos os dispositivos em Solaris são representados como arquivos. Estes arquivos são armazenados no diretório /devices.

```
# ls /devices
iscsi          pci@1f,2000:devctl    pci@1f,4000:devctl    pseudo:devctl
iscsi:devctl   pci@1f,2000:intr      pci@1f,4000:intr      scsi_vhci
options        pci@1f,2000:reg       pci@1f,4000:reg       scsi_vhci:devctl
pci@1f,2000    pci@1f,4000           pseudo
```

Estes são todos os dispositivos que estão conectados ao sistema, inclusive o teclado, monitor, dispositivos de USB e outros. Para diferenciar entre discos rígidos (inclusive *drives* de CD) e outros dispositivos, Solaris provê um diretório separado para os discos rígidos, denominado **/dev/rdsk**.

Ao listar o conteúdo deste diretório, veremos os seguintes arquivos com formatos do tipo: c#d#s# ou c#t#d#s# ou c#d#p#. Estes descrevem o endereço completo de um trecho do disco.

```
# ls /dev/dsk
c0d0p0  c0d0s7    c1t0d0s4  c1t1d0s15 c1t2d0s12 c1t3d0s1  c1t4d0p3
c0d0p1  c0d0s8    c1t0d0s5  c1t1d0s2  c1t2d0s13 c1t3d0s10 c1t4d0p4
c0d0p2  c0d0s9    c1t0d0s6  c1t1d0s3  c1t2d0s14 c1t3d0s11 c1t4d0s0
c0d0p3  c1t0d0p0  c1t0d0s7  c1t1d0s4  c1t2d0s15 c1t3d0s12 c1t4d0s1
c0d0p4  c1t0d0p1  c1t0d0s8  c1t1d0s5  c1t2d0s2  c1t3d0s13 c1t4d0s10
c0d0s0  c1t0d0p2  c1t0d0s9  c1t1d0s6  c1t2d0s3  c1t3d0s14 c1t4d0s11
c0d0s1  c1t0d0p3  c1t1d0p0  c1t1d0s7  c1t2d0s4  c1t3d0s15 c1t4d0s12
c0d0s10 c1t0d0p4  c1t1d0p1  c1t1d0s8  c1t2d0s5  c1t3d0s2  c1t4d0s13
c0d0s11 c1t0d0s0  c1t1d0p2  c1t1d0s9  c1t2d0s6  c1t3d0s3  c1t4d0s14
c0d0s12 c1t0d0s1  c1t1d0p3  c1t2d0p0  c1t2d0s7  c1t3d0s4  c1t4d0s15
c0d0s13 c1t0d0s10 c1t1d0p4  c1t2d0p1  c1t2d0s8  c1t3d0s5  c1t4d0s2
c0d0s14 c1t0d0s11 c1t1d0s0  c1t2d0p2  c1t2d0s9  c1t3d0s6  c1t4d0s3
c0d0s15 c1t0d0s12 c1t1d0s1  c1t2d0p3  c1t3d0p0  c1t3d0s7  c1t4d0s4
c0d0s2  c1t0d0s13 c1t1d0s10 c1t2d0p4  c1t3d0p1  c1t3d0s8  c1t4d0s5
c0d0s3  c1t0d0s14 c1t1d0s11 c1t2d0s0  c1t3d0p2  c1t3d0s9  c1t4d0s6
c0d0s4  c1t0d0s15 c1t1d0s12 c1t2d0s1  c1t3d0p3  c1t4d0p0  c1t4d0s7
c0d0s5  c1t0d0s2  c1t1d0s13 c1t2d0s10 c1t3d0p4  c1t4d0p1  c1t4d0s8
c0d0s6  c1t0d0s3  c1t1d0s14 c1t2d0s11 c1t3d0s0  c1t4d0p2  c1t4d0s9
```

- **c#** - representa o número de controlador. Eles são numerados c0, c1, c2 e assim sucessivamente. E conectam discos rígidos a um controlador.
- **d#** - representa número de discos para aquele controlador.
- **s#** - representa número de trechos. Estes números podem ir de 0 a 15.
- **p#** - representa número da partição, algumas vezes é usado ao invés do número de trechos. Números de partição podem ir de 0 a 4.

A seção de inicialização mostra normalmente quatro dispositivos. Estes dispositivos são: *master* primário, *slave* primário, *master* secundário e *slave* secundário. Seu disco rígido primário é freqüentemente o dispositivo de *master* primário. Um *drive* de CD ou DVD é colocado freqüentemente como o dispositivo *slave* primário. Ao possuir discos rígidos adicionais, estes normalmente são o *master* e o *slave* secundário.

Para Solaris, esta poderia ser uma anotação para estes discos:

- mestre primário: c0d0s0 (controlador 0, disco 0, parte 0)
- escravo primário: c0d1s0 (controlador 0, disco 1, parte 0)
- mestre secundário: c1d0s0 (controlador 1, disco 0, parte 0)
- escravo secundário: c1d1s0 (controlador 1, disco 1, parte 0)

Alguns computadores podem ter uma interface **SCSI**. **SCSI** permite até 16 dispositivos conectados a um único controlador. Freqüentemente, os controladores **SCSI** começam com o número **2**, já que 0 e 1 são, respectivamente, os controladores primários e secundários.

Computadores **SCSI** usam o valor t# para indicar um disco. Este pode variar de 0 a 15. Endereços **SCSI** também podem usar a notação d#, mas sempre é fixo para zero (d0). por exemplo, o arquivos de **c2t2d0s0** até **c2t2d0s15** representam todas as partes do dispositivo nomeados para o **t2** no controlador **2**.

6.2. Administração do agrupamento

Agrupamentos são mantidos pelo comando **zpool**. Comandos de **zpool** permitem criação, exclusão e adição de novos dispositivos, inscrição e modificação de agrupamentos.

Para criar um agrupamento básico em **ZFS**, executamos o comando **zpool**. A sintaxe básica é:

```
# zpool create <poolname> <vdev> <dispositivos>
```

Poolname é o nome do agrupamento a criar. O parâmetro **vdev** descreve que característica de armazenamento deve usar o agrupamento. **Dispositivos** indicam que discos rígidos devem ser usados no agrupamento. É possível criar agrupamento de pedaços de disco (ou até mesmo arquivos) entretanto, isso é feito normalmente com discos inteiros.

Observe que é possível utilizar parte de um disco para formatá-lo como agrupamento do disco. Por exemplo, o seguinte comando cria um **zpool** básico nomeado como **myfirstpool1** que usa o disco *master* secundário.

```
# zpool create myfirstpool1 disk c1d0
```

Para criar um agrupamento espelho, simplesmente substituímos a palavra-chave ***disk*** por ***mirror***. Devemos prover mais que um disco para criar um espelho.

```
# zpool create mymirroedpool mirror c1d0 c1d1
```

Uma coleção de discos pode ser organizada em uma configuração do tipo **RAID-Z** inserindo o valor **raidz** no parâmetro <vdev>. O número indicado de discos para **RAID-Z** está entre 3 a 9.

```
# zpool create myraidzpool raidz c0d1 c1d0 c1d1
```

ZFS permite criar agrupamentos de arquivos regulares. Em vez de especificar um disco, pode-se especificar um arquivo regular. Por exemplo:

```
# zpool create mypool1 file /export/home/alice/mypoolfile
# zpool create mypool2 mirror /export/home/alice/p1 /export/home/alice/p2
```

Esta característica permite testar as características do **ZFS** sem precisar ter discos adicionais. Usaremos isto em futuros exercícios. Para acrescentar um dispositivo a um agrupamento:

```
# zpool add myfirstpool disk c1d0
```

Pode-se acrescentar um espelho ou um dispositivo adicional de **RAID-Z** a um agrupamento existente simplesmente substituindo o disco por um espelho ou **raidz**. O comando a seguir lista todos os agrupamentos **ZFS** disponíveis e as informações de uso espacial e *ßstatus*:

```
 # zpool list
```

Um agrupamento pode ter 3 valores de estado: *online*, *degraded* ou *faulted*. Um **zpool** *online* tem todos seus dispositivos em bom estado de funcionamento. Um agrupamento *degraded* tem um dispositivo falho que ainda pode ser recuperado por redundância. Um dispositivo *faulted* significa que falhou e seus dados não podem mais ser recuperados.

Exportar um agrupamento significa por os dispositivos em uma área de transferência. Para exportar um agrupamento, executamos o comando:

```
# zpool export mypool
```

E devemos importar os dispositivos a um novo computador, através do comando:

```
# zpool import mypool
```

E finalmente, para eliminar um agrupamento, executamos o comando:

```
# zpool destroy mypool
```

Isto elimina um agrupamento e faz com que os dispositivos que faziam parte deste fiquem disponíveis para outros usos.

6.3. Uso Básico de Agrupamento em ZFS

Uma vez que criamos um agrupamento, podemos montá-lo para fazer parte permanente do sistema de arquivos do Solaris. Os usuários economizariam a montagem do diretório a partir do momento que não precisam conhecer que este diretório está usando o **ZFS**. Também é desnecessário saber que os diretórios são espelhados ou armazenados usando RAID-Z.

Para montar um agrupamento como parte do sistema de arquivos do Solaris, usamos o comando:

```
# zfs set mountpoint=/target/directory/in/regular/filesystem poolname
```

Por exemplo, montaremos um agrupamento *mypool* para guardar o diretórios dos usuários.

```
# zfs set mountpoint=/export/home mypool
```

Podemos criar diretórios adicionais em *mypool*. Por exemplo, criamos diretórios para **user1** e **user2**:

```
# zfs create mypool/user1
# zfs create mypool/user2
```

Como já montamos *mypool* para /export/home, **user1** está automaticamente montado como /export/home/user1 e **user2** como /export/home/user2.

Há outras opções adicionais, tais como, compressão, cotas de disco e garantias espaciais de disco que podem ser ativadas com os seguintes comandos.

```
# zfs set compression=on mypool
# zfs set quota=5g mypool/user1
# zfs set reservation=10g mypool/user2
```

6.4. Fotografias e Clones

Como foi discutido, **ZFS** permite a criação de **fotografias** (s*napshots*). Fotografias são cópias de leitura de um sistema de arquivo em um determinado ponto que pode ser usado para propósitos de *backup*.

Para criar uma fotografia, executamos o comando **zfs snapshot**. Indicamos o diretório ZFS que desejamos obter uma fotografia juntamente com um nome deste. Por exemplo, o seguinte comando cria uma fotografia do diretório de projetos de **user1** e nomeado como **ver3backup**.

```
# zfs snapshot mypool/user1/projects@ver3backup
```

Devido ao modo como **ZFS** armazena dados, são criadas fotografias imediatamente e não demandam nenhum espaço adicional. Não existe nenhum processo adicional necessário, **ZFS** preserva os dados originais e os bloqueia sempre que são feitas mudanças no diretório informado.

Todas as fotografias são armazenadas dentro do diretório **zfs/snapshot** localizado na raiz de cada *filesystem*. Isto permite que os usuários possam conferir suas fotografias sem ter que ser um administrador do sistema. Por exemplo, a fotografia que criamos está armazenada em:

```
/export/home/user1/.zfs/snapshot/ver3backup
```

Na qual o user1 pode acessar sem ter que ser um administrador de sistema.

Além disso, podemos executar um *rollback* no diretório a partir de uma fotografia. *Rollback* quer

dizer que podemos, a qualquer momento, desfazer todas as mudanças realizadas no diretório a partir da fotografia. Por exemplo, **user1** pode cometer diversos erros na versão 4 do projeto, deste modo, haveria uma necessidade de retornar à versão 3. Para reverter a situação a uma fotografia denominada **ver3backup**, executamos o comando:

```
# zfs rollback -r mypool/user1/projects@ver3backup
```

Clones em ZFS são cópias de fotografias que podem ser escritas. Para se criar um clone, indicamos a fotografia e o diretório para a cópia.

```
# zfs clone mypool/user1/project@ver3backup mypool/user1/project/ver3copy
```

Existem ainda muitos comandos para o aplicativo **ZFS**. Podemos verificar como utilizá-los acessando o manual, através do comando:

```
# man zfs
```

Parceiros que tornaram JEDI™ possível

Instituto CTS
Patrocinador do DFJUG.

Sun Microsystems
Fornecimento de servidor de dados para o armazenamento dos vídeo-aulas.

Java Research and Development Center da Universidade das Filipinas
Criador da Iniciativa JEDI™.

DFJUG
Detentor dos direitos do JEDI™ nos países de língua portuguesa.

Politec
Suporte e apoio financeiro e logístico a todo o processo.

Instituto Gaudium
Fornecimento da sua infra-estrutura de hardware de seus servidores para que os milhares de alunos possam acessar o material do curso simultaneamente.