Benchmark Ativo com dtrace e IOzone - Relatório Final TPC4 -

Carlos Sá - A59905
carlos.sa01@gmail.com

April 26, 2016

Abstract

*Este documento constitui o relatório final de um estudo feito sobre Benchmark Ativo utilizando as ferramentas **IOZone**, **truss**, e **dtrace**. A ferramenta IOZone será utilizada como ferramenta principal de análise. Sem verificar o resultado desse benchmark, pretende-se tentar replicá-lo utilizando outras ferramentas tais como: **dtrace**, **truss**. Estas são ferramentas de análise e monitorização, através das quais poderemos realizar o traçado de um programa, perceber as invocações de system calls que são realizadas na execução de um determinado programa, sinais do sistema operativo, informação sobre processos etc.*

Sendo este trabalho um trabalho de Benchmark Ativo, pretende-se no final do mesmo, confrontar o resultado do benchmark feito pelo IOzone com a tentativa de replicação desse benchmark recorrendo às restantes ferramentas anteriormente descritas e atestar a veracidade dos resultados devolvidos pelo IOzone.

Numa fase inicial do trabalho a ferramenta IOZone foi explorada quanto à sua diversidade de testes disponiveis por forma a perceber que tipo de profilling de input/output poderá ser realizado. Também as ferramentas dtrace e truss foram estudadas quanto às flags disponíveis por forma a perceber de que forma poderiam ser utilizadas para produzir a replicação dos testes com IOZone.

*Na fase avançada do trabalho a ferramenta truss foi executada juntamente com a execução dos do IOzone para os testes de leitura e escrita e foram criadas duas scripts em dtrace para replicar esses testes. A nível experimental foi feito o teste em 2 tipos de disco diferentes com sistemas de ficheiros diferentes: **SSD** com sistema de ficheiros **ZFS** (minha pasta HOME) e disco **HDD (magnético)** com sistema de ficheiros **UFS**, montado na directoria /diskHitachi.*

1. Introdução ao Benchmark Ativo

Uma das prática de avaliação de performance em sistemas de computação consiste na utilização de ferramentas de benchmark. Na àrea do benchmarking e da análise de performance o autor *Brendan Gregg* distingue no seu blog [1] dois tipos de benchmark:

- Benchmarking Ativo;
- Benchmarking Passivo.

No que toca ao benchmark passivo, um determinado teste é deixado livremente a correr e a análise dos resultados é feita no final, sem que exista uma contra-análise feita por outra ferramenta que replique ativamente os testes realizados pelo benchmark original e permita atestar a veracidade dos resultados. Como discutido por Brendan Gregg em [2] os resultados obtidos por benchmark passivo podem fornecer resultados inválidos que podem conduzir a falsas conclusões devido a:

- Erros no software de benchmarking;
- Testes do software de benchmark limitados;
- Limitações de configuração do benchmark: funcionalidades de desempenho que podem não estar ativadas durante o teste p.e;
- Testes sujeitos a perturbações que ocorrem de forma transparente ao utilizador.

Entre outros problemas e erros que possam decorrer do software de benchmarking na análise e que possa conduzir a uma aceitação passiva dos resultados.

A ideia do benchmark ativo, surge numa tentativa de realizar um determinado benchmark e ativamente fazer uma análise do que está acontecer enquanto o benchmark está a correr. Percebendo o conjunto das acções que o benchmark está a realizar no sistema, é possível utilizar ferramentas auxiliares que permitam replicar o mesmo teste. Desta forma é possivel verificar se os valores obtidos pelo benchmark original estão corretos, e perceber se o benchmark está a medir corretamente aquilo que afirma estar a medir. Esta análise pode ser vista comercialmente como uma tentativa de perceber se uma unidade de armazenamento (por exemplo um

[1] `http://www.brendangregg.com/activebenchmarking.html`

disco magnético) tem a performance que o fabricante anuncia ter.

Neste estudo utilizarei a ferramenta IOZone como ferramenta principal de benchmark, e tentarei fazer benchmark ativo utilizando ferramentas auxiliares como **dtrace** cujas scripts serão criadas com a ajuda da ferramenta **truss**.

2. A ferramenta IOzone

A ferramenta IOzone é uma ferramenta que permite fazer benchmarking dos filesystems de um determinado sistema de computação. Corre na maioria dos sistemas operativos e para este trabalho será utilizado o IOZone numa máquina com Solaris 11. A versão do iozone disponível nesta máquina é a versão **3.434** :

```
a59905@solaris11:~$ /opt/csw/bin/iozone -v
    Iozone  Filesystem Benchmark Program
       Version $Revision: 3.434 $
       Compiled for 64 bit mode.
       (...)
```

E através do comando:

```
a59905@solaris11:~$ df -h
```

Podemos consultar a informação relativa aos Filesystems montados no sistema sobre os quais realizarei este estudo.

```

a59905@solaris11:~$ df -h
Filesystem             Size   Used  Available Capacity  Mounted on
rpool/ROOT/solaris     219G   7.1G       169G     5%    /
/devices                 0K     0K         0K     0%    /devices
/dev                     0K     0K         0K     0%    /dev
ctfs                     0K     0K         0K     0%    /system/contract
proc                     0K     0K         0K     0%    /proc
mnttab                   0K     0K         0K     0%    /etc/mnttab
swap                   5.5G   1.7M       5.5G     1%    /system/volatile
objfs                    0K     0K         0K     0%    /system/object
sharefs                  0K     0K         0K     0%    /etc/dfs/sharetab
/usr/lib/libc/libc_hwcap1.so.1
                       176G   7.1G       169G     5%    /lib/libc.so.1
fd                       0K     0K         0K     0%    /dev/fd
rpool/ROOT/solaris/var
                       219G   388M       169G     1%    /var
swap                   5.7G   159M       5.5G     3%    /tmp
rpool/VARSHARE         219G   847K       169G     1%    /var/share
rpool/export           219G    32K       169G     1%    /export
rpool/export/home      219G    48K       169G     1%    /export/home
rpool/export/home/admin
                       219G   852M       169G     1%    /export/home/admin
rpool/export/home/amp
                       219G   2.5G       169G     2%    /export/home/amp
rpool                  219G   4.8M       169G     1%    /rpool
rpool/VARSHARE/zones   219G    31K       169G     1%    /system/zones
rpool/VARSHARE/pkg     219G    32K       169G     1%    /var/share/pkg
rpool/VARSHARE/pkg/repositories
                       219G    31K       169G     1%    /var/share/pkg/repositories

(....)

rpool/export/home/a59905
                     219G   7.9M       169G     1%    /export/home/a59905
```

A ferramenta IOZone será utilizada como ferramenta principal de benchmarking e disponibiliza ao utilizador um vasto conjunto de testes que podem ser passados ao programa de teste através de flags de acordo com os testes que se pretendem estudar. Todos os testes estão relacionados sobretudo com testes de performance de operações de input/output tais como leituras e escritas de ficheiros de/em disco. Assim, a primeira coisa que fiz foi estudar o conjunto de testes disponíveis pelo IOZone e escolher aqueles que irei utilizar no meu estudo. Através da consulta da *man page* do IOzone de entre o vasto conjunto de testes disponíveis, irei concentrar o meu estudo da ferramenta nos seguintes:

Flags/Testes	Descrição do teste
-a	Utiliza um modo automático e produz resultados que cobre todos os testes para tamanhos de record de 4K a 16M para ficheiros de 64K a 512M
-i0	Para realizar testes sobre a performance de escrita (write) em ficheiro e de re-escrita em ficheiros que já existam. Nota para o facto que uma escrita em ficheiro não contempla apenas a escrita dos dados do ficheiro em si mas também do seu meta-dados (metadata) para ser possível localizar o ficheiro no disco.
-i1	Para realizar testes sobre a performance de leitura (read) de ficheiros já existentes e de ficheiros lidos recentemente (re-read).
-i2	Semelhante aos testes read/write, mas agora com as leituras e escritas dos dados do ficheiro a feitas de/para localizações aleatórias do disco. Neste teste vários factores são tidos em conta e que influenciam os resultados finais: número de discos, latências de acesso a disco, tamanho da cache etc. Ao mesmo tempo permite atenuar os resultados de leituras e escritas de dados de ficheiros que estejam em posições consecutivas no disco.
-i3	Utilizada para medir a performance das leituras feitas em sentido inverso (backwards read).
-I	Garante que são utilizadas apenas operações de I/O diretas no disco (sem utilizar operações de I/O de buffers ou cache).
-r #	Utilizarei para especificar o tamanho do record. O simbolo # significa que a flag -r será utilizada em conjunto com o valor e a ordem de grandeza k (KByte) m (MByte) ou g (GByte) consoante a grandeza do record.
-s #	Usado para especificar o tamanho de ficheiro. O simbolo # significa que a flag -s pode ser usada com conjunto com o valor do tamanho do ficheiro e a sua respectiva ordem de grandeza: k (Kbytes) m (Mbytes) ou g (Gbytes)

Do conjunto de testes do IOzone disponíveis, tentarei neste estudo replicar os testes baseados em operações de reads e writes no disco. Nas secções seguintes sobre experimentação (secção 4) abordarei uma parte da experimentação que realizei que se foca essencialmente nesse tipo de operações.

3. Ferramentas utilizadas na replicação: **dtrace**, **truss**

Para realizar a tarefa de replicação será necessário utilizar ferramentas auxiliares: **DTrace**, **truss**.

A ferramenta DTrace é uma ferramenta que implementa uma interface simples de chamadas da linguagem **D**, capaz de fornecer informação sobre o traçado de aplicações utilizando o kernel. A linguagem **D** é a linguagem fundamental da utilização desta ferramenta e as informações obtidas sobre o programa "a traçar" são conseguidas com a utilização e interpretação das scripts em linguagem **D**. A ferramenta *dtrace* será utilizada neste estudo com o objectivo de tornar possível a tarefa de duplicação dos resultados obtidos pelo IOzone. Para tal criarei um conjunto de scripts em linguagem **D** que tentará realizar os mesmos testes que executarei com o IOzone, e no final comparar os resultados obtidos por ambos. O comando truss é um comando que permite fazer o traçado de syscalls e sinais. Neste trabalho este comando têm igualmente um papel fundamental para a construção de scripts em **dtrace** pois permitirá perceber as system calls que são invocadas pelo IOzone quando este se encontra a executar os testes. Tal significa que mesmo sem o código fonte de um programa, se este for passado na shell ao comando truss como entrada, este é capaz de fornecer informação das chamadas ao sistema que são realizadas na sua execução. Daí a sua importância nesta tarefa de replicação. Existem outros comandos semelhantes como o *strace* que permitem fazer o mesmo tipo de traçado e que será igualmente explorado.

Existem também outro tipo de ferramentas como o **ltrace** que além de permitir o traçado de syscalls, permite saber as bibliotecas invocadas no sistema durante a execução do programa e será utilizado apenas na fase de testes e experimentação.

4. Experimentação com ferramenta IOzone

Nesta secção irei documentar uma pequena parte dos testes que fiz até ao momento com IOzone. Este conjunto de testes foram essenciais para perceber melhor os testes que o IOzone realiza e perceber melhor qual a informação relevante que pode fornecer.

4.1. Experimentação com iozone automatico

O primeiro comando explorado foi utilizando a flag **-a** para realizar os testes em modo automático. Desta forma percebi que a utilização desta flag cobre todos os testes do IOzone e que embora se possa especificar o tamanho de record, este modo automaticamente cria records de tamanho 4k, 16M, 64k e 512M:

```
$ /opt/csw/bin/iozone -a
```

De entre o output imprimido está o resultado dos testes obtidos pelo iozone: read, reread, write, rewrite, random read, random write, etc. Todos os resultados obtidos estão em KB/s. Algumas informações sobre a cache do processador é igualmente imprida. Na máquina Solaris, o IOzone diz que o tamanho da cache é de 1024kbytes com linhas de cache de 32kbytes.

4.2. Testes de Escrita: write e re-write

Depois de perceber os diferentes testes no modo automatico e perceber os valores obtidos para todos os testes do IOzone, fui restrigindo o dominio de testes e tirando algumas conclusões sobre os valores obtidos pelos diferentes testes. Para já não será dada grande importância aos tamanhos dos ficheiros nem aos tamanhos dos *records*.

Para realizar o teste de escrita executei:

```
$ /opt/csw/bin/iozone -a -i0
```

De entre a totalidade do output relevante obtive os seguintes resultados:

```
kB  reclen    write  rewrite    (...)
64       8   453587  1210227
64      16   336889   679917
64      32   357529   831569
64      64   333540   686877
128      4   339403   640042
128      8   542400  1105114
128     16   362798   710329
128     32   357721   680612
128     64   373136   699227
128    128   558771   976473
256      4   331626   759469
256      8   377572   725106
256     16   351722   701422
256     32   572721   941213
256     64   392620   260376
256    128   372076   705570
256    256   394496   701422
512      4   579804  1084694
512      8   386112   741840^C
```

E uma conclusão simples de obter é que a performance de escrita num ficheiro já existente (rewrite) é maior do que quando o ficheiro aínda não existe.

4.3. Testes de Leitura: Read e re-read

```
$ /opt/csw/bin/iozone -a -i0 -i1
```

Ao realizar os testes de leitura podemos verificar a rapidez de leitura de um ficheiro já existente (read) e a rapidez de leitura de um ficheiro que já tenha sido lido anteriormente. Uma vez que o sistema operativo faz caching de blocos de ficheiros lidos, percebemos que ler um ficheiro que já tenha sido lido anteriormente (reread) é mais rápido do que um ficheiro que aínda não tenha sido lido. Então os valores dessa rapidez obtidos na coluna reread são respectivamente maiores do que os obtidos na coluna de read.

```
kB  reclen    read    reread   (....)
64       4  1734015   1828508
64       8  2467108   2561267
64      16  2662899   2772930
64      32  2801873   2923952
64      64  3363612   3363612
128      4  3359523   4135958
128      8  2058509   2608629
128     16  4717434   6406138
128     32  4889281   6727225
128     64  2843510   3297617
128    128  3199360   3199360
256      4  3236056   4181144
256      8  4054829   5810112
256     16  2266207   2812272
256     32  2533571   3123106
256     64  2613746   3326279
256    128  2754556   3236056
256    256  1445521   1479379
```

```
512          4     1667552   1933266
512          8     4228947   5509113
512         16     2178404   2782414
512         32     4871723   6736026^C
```

5. Testes com truss

Como o truss tem uma funcionalidade poderosa em realizar o traçado de todas as system calls invocadas na execução de um determinado programa, poderemos utilizá-lo para traçar as chamadas ao sistema que são invocadas quando realizamos um determinado teste com iozone.

A titulo de exemplo, ao executar na shell:

```
$ truss /opt/csw/bin/iozone -a -i0 -s1m -r32k
```

Estamos a executar um teste IOzone de escrita e re-escrita com um ficheiro de 1MB e records de 32KB tamanho, fazendo simultaneamente truss desse teste.

Assim, podemos destacar algum output relevante tal como:

```
execve("/opt/csw/bin/iozone", 0xFE76D974, 0xFE76D98C)  argc = 5
sysinfo(SI_MACHINE, "i86pc", 257)               = 6
mmap(0x00000000, 32768, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANON, -1, 0) = 0xF87E0000

(.....)

stat("/opt/csw/lib/amd64/libthread.so.1", 0xFFFF80FFBFFFEB80) Err#2 ENOENT
stat("/opt/csw/lib/pentium_pro+mmx/libthread.so.1", 0xFFFF80FFBFFFEB80) Err#2 ENOENT
stat("/opt/csw/lib/pentium_pro/libthread.so.1", 0xFFFF80FFBFFFEB80) Err#2 ENOENT
stat("/opt/csw/lib/pentium+mmx/libthread.so.1", 0xFFFF80FFBFFFEB80) Err#2 ENOENT
stat("/opt/csw/lib/pentium/libthread.so.1", 0xFFFF80FFBFFFEB80) Err#2 ENOENT
stat("/opt/csw/lib/i486/libthread.so.1", 0xFFFF80FFBFFFEB80) Err#2 ENOENT
stat("/opt/csw/lib/i386/libthread.so.1", 0xFFFF80FFBFFFEB80) Err#2 ENOENT
stat("/opt/csw/lib/i86/libthread.so.1", 0xFFFF80FFBFFFEB80) Err#2 ENOENT
stat("/opt/csw/lib/64/libthread.so.1", 0xFFFF80FFBFFFEB80) Err#2 ENOENT
stat("/lib/64/libthread.so.1", 0xFFFF80FFBFFFEB80) = 0
resolvepath("/lib/64/libthread.so.1", "/lib/amd64/libthread.so.1", 1023) = 25
open("/lib/64/libthread.so.1", O_RDONLY)        = 3

(...)

open("iozone.tmp", O_RDWR)                      = 3
fdsync(3, FSYNC)                                = 0
write(3, " y y y y y y y y\0\0\0\0".., 32768)    = 32768
write(3, " y y y y y y y y\0\0\0\0".., 32768)    = 32768
write(3, " y y y y y y y y\0\0\0\0".., 32768)    = 32768
write(3, " y y y y y y y y\0\0\0\0".., 32768)    = 32768
write(3, " y y y y y y y y\0\0\0\0".., 32768)    = 32768
write(3, " y y y y y y y y\0\0\0\0".., 32768)    = 32768
write(3, " y y y y y y y y\0\0\0\0".., 32768)    = 32768
write(3, " y y y y y y y y\0\0\0\0".., 32768)    = 32768
write(3, " y y y y y y y y\0\0\0\0".., 32768)    = 32768
write(3, " y y y y y y y y\0\0\0\0".., 32768)    = 32768
write(3, " y y y y y y y y\0\0\0\0".., 32768)    = 32768
write(3, " y y y y y y y y\0\0\0\0".., 32768)    = 32768
write(3, " y y y y y y y y\0\0\0\0".., 32768)    = 32768
write(3, " y y y y y y y y\0\0\0\0".., 32768)    = 32768
write(3, " y y y y y y y y\0\0\0\0".., 32768)    = 32768
write(3, " y y y y y y y y\0\0\0\0".., 32768)    = 32768
write(3, " y y y y y y y y\0\0\0\0".., 32768)    = 32768
write(3, " y y y y y y y y\0\0\0\0".., 32768)    = 32768
write(3, " y y y y y y y y\0\0\0\0".., 32768)    = 32768
```

```
write(3, " y y y y y y y y\0\0\0\0".., 32768)    = 32768
write(3, " y y y y y y y y\0\0\0\0".., 32768)    = 32768
write(3, " y y y y y y y y\0\0\0\0".., 32768)    = 32768
write(3, " y y y y y y y y\0\0\0\0".., 32768)    = 32768
write(3, " y y y y y y y y\0\0\0\0".., 32768)    = 32768
write(3, " y y y y y y y y\0\0\0\0".., 32768)    = 32768
write(3, " y y y y y y y y\0\0\0\0".., 32768)    = 32768
write(3, " y y y y y y y y\0\0\0\0".., 32768)    = 32768
write(3, " y y y y y y y y\0\0\0\0".., 32768)    = 32768
write(3, " y y y y y y y y\0\0\0\0".., 32768)    = 32768
write(3, " y y y y y y y y\0\0\0\0".., 32768)    = 32768
write(3, " y y y y y y y y\0\0\0\0".., 32768)    = 32768
write(3, " y y y y y y y y\0\0\0\0".., 32768)    = 32768
fdsync(3, FSYNC)                                 = 0
close(3)                                         = 0
(...)
_exit(0)
```

No inicio percebemos que é feita a chamada ao sistema execve para executar o comando do iozone. Entretanto algumas bibliotecas são invocadas e o ficheiro "iozone.tmp" é aberto pela syscall open e começam a ser feitas as escritas no ficheiro. É possível observar todo o tipo de interação feito pelos testes do IOzone com o sistema operativo. Como executamos o teste um ficheiro de **1MB** e records de **32KB**, é possivel verificar que existe um total de 32 *writes* para escrever no ficheiro **iozone.tmp**.

Assim podemos perceber que mesmo sem possuír o código fonte, o truss é capaz de dar pistas sobre que tipo de acções é que um teste de IOzone realiza no sistema operativo. Por essa razão o comando truss revela-se um comando auxiliar que poderá ser utilizado para ajudar replicar o teste utilizando dtrace.

6. Scripts dtrace criadas

Para replicar os testes realizados pelo IOZone foram criadas duas scripts diferentes seguindo duas lógicas diferentes: uma script dtrace para replicar os testes IOZone sobre operações de escrita **(write e rewrite)** e outra para operações de leitura **read e reread**.

- Script Dtrace para operações **write e rewrite (write_tp4_script.d)**

```
#!/usr/sbin/dtrace -s

#pragma D option quiet

syscall::open*:entry
/execname=="iozone" & uid==29214/
{
 self->path = copyinstr(arg1);
}

syscall::open*:return
/self->path=="iozone.tmp"/
{
 flag = 1;
 size = 0;
 total=0;
}

syscall::write:entry
/flag == 1/
{
 self->start_clock = timestamp;
 self->write_size = (arg2/1024);
 size = size + self->write_size;
}

syscall::write:return
/self->start_clock > 0 & self->↩
    write_size > 0/
{
 self->stop_time = timestamp;
 self->elapsed = self->stop_time − ↩
     self->start_clock;
 total = total + self->elapsed;
}


syscall::close*:entry
/self->path=="iozone.tmp"/
{
 printf("%-12s %s\n","TAMANHO","↩
     TEMPO TOTAL");
 printf("%-12d %d\n",size,total);
 flag = 0;
}
```

- Script Dtrace para operações **read e reread (read_tp4_script.d)**

```
#!/usr/sbin/dtrace -s

#pragma D option quiet

syscall::open*:entry
```

```
/execname=="iozone" & uid==29214/
{
 self->path = copyinstr(arg1);
 self->flag = arg2;
}

syscall::open*:return
/self->path=="iozone.tmp" /
{
 self->start_clock = timestamp;
 total=0;
 flag = 1;
 size = 0;
}

syscall::read:return
/self->start_clock > 0/
{
 self->read_size = (arg0/1024);
 size = size + self->read_size;
}

syscall::close*:entry
/self->path=="iozone.tmp"/
{
 self->stop_clock = timestamp;
 self->elapsed = self->stop_clock - ↩
     self->start_clock;
 total = total + self->elapsed;
 printf("%-12s %s\n","TAMANHO","↩
     TEMPO TOTAL");
 printf("%-12d %d\n",size,total);
 flag = 0;
}
```

Em ambas as scripts foi feita a garantia (através de predicados) que os testes realizados só recolhem informação respetiva aos ficheiros iozone.tmp que digam respeito à execução do iozone através da minha conta de utilizador a59905. Daí a necessidade de utilizar o comando **id** para saber o meu user id (uid) e utilizá-lo no predicado do provider syscall::open*:entry de ambas as scripts.

Como poderemos verificar pelas scripts construídas a ideia de ambas é: Contabilizar o número de KBytes lidos/escritos contabilizar o tempo de cpu com timestamp (contabilizado em nanosegundos). Os resultados e o cálculo do valor em KB/s é feito posteriormente de acordo com:

$$\#KB/s = \frac{TamanhoTotal\_KBytes}{TempoTotal(ns) \times 10^{-9}} \qquad (1)$$

e verifica-se se os valores obtidos pela script dtrace são ou não aceitáveis de acordo com os resultados devolvidos pelo IOzone.

7. Testes em disco SSD com FS ZFS (HOME)

O primeiro grupo de testes foi realizado no disco SSD e as scripts executadas na minha home.

- **Teste para write e rewrite**

Este teste foi feito de acordo com a execução do IOzone para escrita de um ficheiro com 15M da seguinte forma:

```
$ /opt/csw/bin/iozone -i0 -s15m
```

```
File size set to 15360 kB
Command line used: /opt/csw/bin/iozone -i0 -s15m
Output is in kBytes/sec
Time Resolution = 0.000001 seconds.
Processor cache size set to 1024 kBytes.
Processor cache line size set to 32 bytes.
File stride size set to 17 * record size.
kB  reclen    write  rewrite
15360       4   204502   400301
```

Com a script **write_tp4_script.d** obtive os seguintes resultados:

```
TAMANHO      TEMPO TOTAL
0            0
TAMANHO      TEMPO TOTAL
0            0
TAMANHO      TEMPO TOTAL
15360        65909668
TAMANHO      TEMPO TOTAL
15360        65909668
```

De acordo com a formula de cálculo 1:

$$\#KB/s = \frac{15360}{65909668(ns) \times 10^{-9}} \approx 233046KB/s \qquad (2)$$

Podemos então verificar que existe alguma divergência entre o resultado fornecido pelo IOzone e aquele que foi conseguido com a script dtrace. No entanto creio que o resultado seja aceitável pelo facto deste se encontrar no intervalo do resultado fornecido pelo IOzone para write e rewrite. Ou seja, no intervalo [204502,400301]. Pelo resultado percebi que o benchmark realizado pelo IOZone contabiliza o tempo desde que é feita uma chamada da syscall open até atingir o close. O numero de bytes é contabilizado com a invocação da syscall write.

Ao longo do estudo fui testando a script para execuções de diferentes tamanhos de ficheiro com o IOZone. Uma facto constatado foi que à medida que se aumenta o tamanho do ficheiro a divergência entre o resultado obtido com IOzone e a script dtrace é maior. Por essa razão abandonei a estratégia inicial apresentada no pré-relatório deste trabalho que incluia testes para ficheiros de tamanhos substancialmente maiores.

- **Teste para read e reread**

O teste de leitura com o IOzone na home foi com a execução do seguinte comando:

```
$ /opt/csw/bin/iozone -i0 -i1 -s15m
```

```
File size set to 15360 kB
Command line used: /opt/csw/bin/iozone -i0 -i1 -s15m
Output is in kBytes/sec
Time Resolution = 0.000001 seconds.
Processor cache size set to 1024 kBytes.
Processor cache line size set to 32 bytes.
File stride size set to 17 * record size.
kB  reclen    write  rewrite    read    reread
15360       4   247299   521842  1143884  1292605
```

Com a script dtrace utilizada (**read_tp4_script.d**) foi obtido o resultado:

```
TAMANHO      TEMPO TOTAL
0            61924
TAMANHO      TEMPO TOTAL
0            4160
TAMANHO      TEMPO TOTAL
0            113490233
TAMANHO      TEMPO TOTAL
0            226991634
TAMANHO      TEMPO TOTAL
0            70838909
TAMANHO      TEMPO TOTAL
15364        13460772
TAMANHO      TEMPO TOTAL
15364        11897026
```

Mais uma vez, pelo cálculo 1:

$$\#KB/s = \frac{15364}{13460772(ns) \times 10^{-9}} \approx 1141390KB/s \quad (3)$$

Para este caso é possivel verificar que o valor obtido pela script dtrace já se aproxima mais do valor obtido pelo IOzone. Alguma experimentação foi feita e conclui que os resultados obtidos para os reads neste sistema de ficheiros são um pouco mais realistas do que a contra-análise feita para os writes. No que toca à forma como a contabilização de número de bytes contagem de tempo feito pelo IOzone faz, as conclusões são análogas às obtidas para a experimentação realizada com a script dtrace para write.

8. Testes em disco HDD com FS UFS (/diskHitachi)

Nesta secção irei abordar o estudo dos resultados obtidos com HDD montado em **/diskHitachi**. De sublinhar que para executar os testes neste disco foi necessário alterar a localização onde o ficheiro temporário é criado, para que os resultados sejam coerentes. Assim, foi necessário fazer algumas alterações minimas nos predicados das scripts **write_tp4_script.d** e **read_tp4_script.d**.

Concretamente, foi necessário alterar o **path** no provider *syscall* (open* e no close*) em ambas as scripts da seguinte forma :

```
(...)
syscall::open*:return
    /self->path=="/diskHitachi/a59905/↩
        iozone.tmp"/
(...)
syscall::close*:entry
    /self->path=="/diskHitachi/a59905/↩
        iozone.tmp"/
```

- **Teste para write e rewrite**

Á semelhança dos testes anteriores, neste disco os testes foram igualmente feitos com a escrita de um ficheiro com 15M. Como agora estou a realizar os testes num disco diferente, necessitei de utilizar a flag **-f** por forma a que mudar a localização da criação do ficheiro temporário. Abaixo segue a execução do iozone para os testes write e rewrite no disco HDD.

```
$ /opt/csw/bin/iozone -f /diskHitachi/↩
    a59905/iozone.tmp -i0 -s15m
```

```
File size set to 15360 kB
Command line used: /opt/csw/bin/iozone -f /diskHitachi/a59905/iozone.tmp -↩
      i0 -s15m
Output is in kBytes/sec
Time Resolution = 0.000001 seconds.
Processor cache size set to 1024 kBytes.
Processor cache line size set to 32 bytes.
File stride size set to 17 * record size.
kB  reclen    write  rewrite
15360       4   275238   517379
```

Com a script **write_tp4_script.d** obtive os seguintes resultados:

```
TAMANHO      TEMPO TOTAL
0            0
TAMANHO      TEMPO TOTAL
0            0
TAMANHO      TEMPO TOTAL
15360        51210211
TAMANHO      TEMPO TOTAL
15360        51210211
TAMANHO      TEMPO TOTAL
15360        25509974
```

De acordo com a formula de cálculo 1:

$$\#KB/s = \frac{15360}{51210211(ns) \times 10^{-9}} \approx 299940KB/s \quad (4)$$

Mais uma vez, verificamos alguma divergência entre o resultado fornecido pelo IOzone e aquele que foi conseguido com a script dtrace para as operações de escrita e reescrita. No entanto creio que o resultado seja aceitável pelo facto deste se encontrar no intervalo do resultado fornecido pelo IOzone para write e rewrite. Ou seja, no intervalo [275238,517379]. Pelo resultado percebi que o benchmark realizado pelo IOZone contabiliza o tempo desde que é feita uma chamada da syscall open até atingir o close. O numero de bytes é contabilizado com a invocação da syscall write.

- **Teste para read e reread**

O teste de leitura com o IOzone na home foi com a execução do seguinte comando:

```
$ /opt/csw/bin/iozone -f /diskHitachi/a59905/iozone.tmp -i0 -i1 -s15m
```

```
File size set to 15360 kB
Command line used: /opt/csw/bin/iozone -f /diskHitachi/a59905/iozone.tmp -i0 -i1 -s15m
Output is in kBytes/sec
Time Resolution = 0.000001 seconds.
Processor cache size set to 1024 kBytes.
Processor cache line size set to 32 bytes.
File stride size set to 17 * record size.
kB  reclen    write  rewrite    read    reread
15360       4   168921   395815  1243217  1505287
```

Com a script dtrace utilizada (**read_tp4_script.d**) foi obtido o resultado:

```
TAMANHO      TEMPO TOTAL
0            37654
TAMANHO      TEMPO TOTAL
0            3002
TAMANHO      TEMPO TOTAL
0            360536573
TAMANHO      TEMPO TOTAL
0            721095217
TAMANHO      TEMPO TOTAL
0            237120966
TAMANHO      TEMPO TOTAL
15364        12516933
TAMANHO      TEMPO TOTAL
15364        10340238
```

Pela fórmula de cálculo 1 para o número de KB/s:

$$\#KB/s = \frac{15364}{12516933(ns) \times 10^{-9}} \approx 1227457KB/s \tag{5}$$

À semelhança dos resultados para os testes de leitura feitos com o disco SSD verificamos que o valor obtido pela script dtrace para os reads já se aproxima mais do valor obtido pelo IOzone (no intervalo [1243217,1505287]. Tal como aconteceu para o sistema de ficheiros ZFS, concluo que no sistema de ficheiros UFS os valores conseguidos com a script dtrace para as operações de leitura são um pouco mais realistas que os correspondentes resultados obtidos com a script para contra-análise feita sobre as operações de escrita do IOzone.

9. Comparação dos resultados SSD (FS: ZFS) VS HDD (FS: UFS)

Depois de feitos todos os testes, decidi juntar todos os resultados numa tabela. A tabela 1 mostra todos os resultados obtidos para os testes realizados.

Disco		Read (KB/s)	Reread (KB/s	Write (KB/s)	Rewrite (KB/s)
SSD (ZFS) /export/home/	IOzone	1143884	1292605	204502	400301
	Dtrace	1141390		233046	
HDD (UFS) /diskHitachi	IOzone	1243217	1505287	275238	517379
	Dtrace	1227457		299940	

Table 1: *Tabela de resultados para testes com discos SSD e HDD*

De acordo com a observação da tabela consigo comprovar alguma da informação que encontrei durante o meu estudo sobre IOzone inclusivamente alguma da informação encontrada nas notas de apoio [1] . Em ambos os discos, verifica-se uma maior performance nas operações de re-leitura comparativamente às operações de leitura. Possivelmente devendo-se ao facto do sistema operativo fazer caching dos dados relativos a ficheiros lidos recentemente. Igualmente nas operações de escrita, comprovei que as re-escritas (rewrites) têm uma maior performance do que as escritas (writes) em ambos os discos. Isto deve-se ao facto de que quando um ficheiro que não existe é escrito pela primeira vez, é necessário escrever no disco os metadados relativos ao ficheiro além dos dados do ficheiro em si. Esses metadados contem informação necessária para posteriormente permitir encontrar o ficheiro no disco. Assim, a performance das re-escritas será naturalmente superior à performance das escritas.

O resultado que me suscitou maior surpresa foi ter-se verificado uma performance tanto nas leituras como nas escritas no disco HDD. Os motivos para tal acontecer podem vir de vários factores. Um dos aspectos fisicos que mais diferenciam os dois tipos de discos está na componente mecânica de funcionamento. Enquanto nos HDD temos partes móveis que necessitam de movimento mecânico para fazer acesso aos dados e realizar as operações de escrita e leitura tal não acontece nos discos SSD. Neste, o acesso aos dados é feito através de um controlador embutido, que é a chave para a rapidez de acesso no que toca a operações e velocidade de leitura e escrita. Embora um bom ou mau controlador embutido num disco SSD possa fazer uma notória diferença no que toca a velocidades de leitura e escrita, não seria de esperar que tantos as leituras como as escritas tivessem uma performance superior num disco HDD.

Além disso, num disco HDD a procura dos ficheiros é feita de forma sequencial enquanto que num disco SSD tal é feito de forma aleatória (pelo menos nos discos SSD cuja memória é baseada em memórias DRAM).

10. Conclusão

Ao longo deste estudo foram explorados os diversos conceitos relacionados com performance e benchmarking de operações de input/output em dois tipos de discos diferentes (HDD VS SSD). Todo o trabalho desenvolvido foi feito com base no conceito de benchmarking ativo. Creio que com o trabalho que realizei fui capaz de realizar uma contra-análise ao benchmarking feito pelo IOzone com as scripts criadas em Dtrace. Essencialmente foram explorados apenas os testes de leitura e escrita para ambos os discos

com as scripts dtrace. Todo o trabalho foi dividido em duas componentes: Uma primeira componente relativa à experimentação e familiarização com as ferramentas **dtrace**, **truss** e **IOzone** e perceber os testes que poderia tentar replicar. Todas as ferramentas foram exploradas nessa fase inicial com diferentes flags por forma a realizar diferentes testes.

De um modo geral percebi que teria que utilizar a ferramenta **dtrace** para criar um conjunto de scripts que me permitissem replicar os testes que o IOZone realiza e comparar os resultados. Esse foi o trabalho que desenvolvi na segunda fase. Ao longo da criação das scripts dtrace surgiram algumas dificuldades o que condicionou o número de testes do IOzone replicados com sucesso. A nível de resultados do benchmarking feito pelo IOzone e pelo dtrace, apesar dos valores conseguidos com as scripts dtrace não serem iguais aos medidos pelo IOzone em todos os casos encontram-se dentro de um intervalo razoável, sobretudo mais próximos nas operações de leitura.

[3] [2] [1]

References

[1] Iozone Filesystem Benchmark william d. norcott. `http://gec.di.uminho.pt/minf/cpd1314/PAC/material1314/IOzone_msword_98.pdf`.

[2] Brendan Gregg. *Systems Performance: Enterprise and the Cloud*. Prentice Hall Press, Upper Saddle River, NJ, USA, 1st edition, 2013.

[3] Brendan Gregg and Jim Mauro. *DTrace: Dynamic Tracing in Oracle Solaris, Mac OS X and FreeBSD*. Prentice Hall Press, Upper Saddle River, NJ, USA, 1st edition, 2011.