TPC4: Análise de Desempenho com Benchmark Ativo e Passivo
- Engenharia dos sistemas de Computação -

autor: Daniel Malhadas

***Resumo*—O presente documento apresenta um estudo aprofundado sobre duas técnicas distintas de análise de desempenho, sendo elas benchmark ativo e benchmark passivo. De forma a entender as diferenças entre os resultados obtidos com cada uma destas técnicas ambas são usadas para analisar o desempenho de aplicações nas mesmas condições (como aplicação de referência será usado IOzone), sendo depois os resultados comparados e avaliados tornando-se possível observar as diferenças e vantagens do método ativo para o método passivo. Note-se ainda que todo este estudo foi realizado em ambiente Solaris 11 e, por essa razão, os dados instrumentados serão relativos somente a esse ambiente.**

***Index Terms*—Análise de Desempenho, Benchmark Ativo, Benchmark Passivo, IOzone, DTrace, Solaris11, Computação Paralela e Distribuída.**

I. Introdução

A. Contextualização e Motivação

Com este projeto pretende-se recorrer ao uso de IOzone enquanto aplicação de referência para fazer a avaliação de desempenho para uma multiplicidade de operações, incluindo entre outras: Ler/escrever, reler/reescrever, ler/escrever para trás, etc., em diferentes sistemas de ficheiros (ufs, zfs, nfs). Esta ferramenta irá permitir-nos obter resultados relativos a um benchmark passivo. Será também realizado um benchmark ativo da aplicação de referência IOzone com recurso a DTrace e outras ferramentas auxiliares cujos resultados serão comparados com os do benchmark passivo tornando-se possível observar as diferenças e as vantagens do método ativo em relação ao método passivo. Por consequência acaba também por se fazer uma análise das desvantagens do uso de IOzone em comparação com DTrace para benchmarking.
Estudar estes métodos de benchmark e compreender as suas diferenças é essencial no desenvolvimento de aplicações onde a performance é importante já que muitos sistemas operativos são afinados para terem um bom desempenho de entradas e saídas de dados em algumas das aplicações mais utilizadas, mas tal afinação pode não ser apropriada para outras aplicações[1]. Outras vezes, os padrões de acesso aos dados podem mudar, por exemplo passando de leituras e escritas sequenciais para leituras e escritas com padrões aleatórios, o que se pode traduzir em aumentos significativos dos tempos esperados para as operações nos sistemas de entrada e saída de dados[1]. Logo o uso destas técnicas de benchmarking torna-se relevante de forma a detetar estas situações para, com isso em mente, construir melhores aplicações que tirem proveito de características da máquina em questão. Estas técnicas de benchmarking permitem também que um possível comprador da máquina entenda se esta se adapta ou não para os propósitos que pretende, sendo assim possível uma decisão ponderada sobre se deve ou não comprar essa máquina[1].

B. Em que Consiste Benchmark Passivo?

Com o benchmark passivo, um determinado teste é deixado livremente a correr e a análise dos resultados executada por uma dada ferramenta é feita apenas no final sem que seja realizada uma contra-análise que permita atestar a veracidade dos resultados obtidos.
Neste projeto usaremos os resultados da própria ferramenta usada como teste de referência, IOzone, de forma a realizar o benchmark passivo.

C. Em que Consiste Benchmark Ativo?

A ideia do benchmark ativo é analisar a aplicação enquanto o teste de referência está a correr (e não apenas depois de concluído), usando outras ferramentas. Desta forma é possível não apenas confirmar a correção dos testes realizados, mas, ao mesmo tempo, obter informações específicas e relevantes sobre o sistema ou aplicação em análise que permitam, por exemplo, identificar os reais limitadores do sistema em teste, ou do próprio benchmark.
Neste projeto usaremos a ferramenta DTrace e outras auxiliares de forma a realizar este benchmark ativo.

II. Considerações Iniciais

A. Caracterização do Ambiente de Testes

De forma a realizar os testes propostos no capítulo anterior foi escolhido como ambiente de testes o sistema operativo Solaris11 e, por essa razão, os dados instrumentados serão relativos somente a esse ambiente. Mais concretamente, a máquina utilizada encontra-se caracterizada pormenorizadamente na tabela que se segue:

Sistema Operativo	Solaris11
Fabricante	Intel
Modelo do CPU	Dual Intel Xeon E5-2650 v2
Microarquitetura do CPU	Ivy Bridge
Frequência do Clock	2.60 GHz
#Cores	8, com 16 threads (2 por core)
Cache	L1: 32kB por core, L2: 256kB por core, L3: 20480kB partilhada
Largura de Banda de acesso à memória	59.7GB/s
Memória RAM	64GB
FileSystems	zfs, ufs, nfs, smb, autofs, smbfs

B. Em que consiste a ferramenta IOzone?

IOzone é uma ferramenta para benchmark de sistemas de ficheiros, sendo que para efetuar esse benchmarking executa e avalia de forma passiva diversas operações com ficheiros[1]. A ferramenta encontra-se em várias máquinas e sistemas operativos distintos não sendo portanto específica a nenhum tipo de ambiente.
IOzone é útil para analisar a performance de uma dada máquina relativamente a operações com ficheiros, nomeadamente:[1]

read, write
Estes testes medem a performance de escrita num novo ficheiro (write) e de leitura de um ficheiro já existente (read). Quando se escreve num novo ficheiro temos um overhead chamado 'metadata' que representa o impacto na performance de ter de guardar os dados escritos no ficheiro e de registar onde os dados se localizam na mídia de armazenamento.
re-read, re-write
Estes testes medem a performance de leitura de um ficheiro que foi lido recentemente (re-read) ou de escrita num ficheiro que já existe (re-write). É normal que estes testes resultem numa melhor performance que um simples read ou write já que, para o re-read, vamos ter os dados necessários em cache e, para o re-write, já vamos ter o overhead 'metadata' previamente realizado.
read backwards
Este teste mede a performance de ler um ficheiro para trás. Embora muitos sistemas operativos estejam otimizados para leituras para a frente, mais convencionais, poucos estão para uma leitura mais fora do normal como esta para trás.
read striped
Este teste mede o desempenho de ler um ficheiro com um acesso espaçado. Um exemplo seria: Ler no offset zero para um comprimento de 4 Kbytes, em seguida, procurar 200 Kbytes e ler para um comprimento de 4 Kbytes, depois, procurar 200 Kbytes e assim por diante. Aqui o padrão é ler 4 Kbytes e depois procurar 200 Kbytes e repetir o padrão. Este é um comportamento típico para aplicações que têm estruturas de dados contidas dentro de um ficheiro e estão a tentar aceder a uma região específica da estrutura de dados. Este comportamento de acesso também pode, por vezes, produzir anomalias de desempenho interessantes.
fread, fwrite, freread, frewrite
Estes testes fazem o mesmo que os testes de read, write, reread, rewrite, no entanto utilizam a função de biblioteca fwrite(). Desta forma usa-se um buffer ao nível do utilizador e reduz-se as chamadas ao sistema, o que pode melhorar bastante a performance.
random read, random write, random mix
Estes testes medem a performance de ler um ficheiro com vários acessos aleatórios (random read), escrever um ficheiro com vários acessos aleatórios (random write) e de escrever e ler um ficheiro com vários acessos aleatórios (random mix). O impacto deste tipo de atividades
record rewrite
Este teste mede o desempenho de escrever e reescrever num determinado local dentro de um ficheiro. Se o tamanho do local é pequeno o suficiente para caber na cache de dados do CPU, o desempenho é muito alto. No entanto níveis diferentes de desempenho serão obtidos noutras situações.

async I/O
Teste especializado que mede a performance do mecanismo POSIX async I/O utilizando coisas como aio_write(), aio_read() e aio_error().
mmap
Teste especializado que mede a performance do uso do mecanismo mmap() para operações de I/O.

Como mencionado no capítulo anterior, por vezes uma dada máquina e um dado sistema operativo podem estar otimizados para determinados tipos de aplicações específicos sendo que o uso de ferramentas como IOzone permite detetar as situações para as quais a máquina se encontra otimizada permitindo, com isso em mente, construir melhores aplicações que tirem proveito dessas características da máquina em questão ou até de um possível comprador da máquina entender se esta se adapta ou não para os propósitos que pretende, sendo assim possível uma decisão ponderada sobre se deve ou não comprar essa máquina[1].

Note-se ainda que esta ferramenta permite o uso de diversas flags diferentes. No Anexo A pode encontrar-se então uma tabela que menciona cada flag existente e uma breve descrição sendo que as flags utilizadas serão futuramente referenciadas sem uma explicação adicional, fica portanto aqui uma tabela que permite ao leitor entender o uso de uma determinada flag.

C. *Em que consiste a ferramenta DTrace?*

A Oracle Solaris DTrace é uma ferramenta de rastreio/traçado avançada usada para testar troços problemáticos de um programa em tempo real. Para isto, esta ferramenta permite observar questões de performance tanto em pequenas aplicações como do próprio sistema operativo em si de forma dinâmica e segura permitindo a quem a utilize a identificação de problemas que seriam difíceis de detetar com outras ferramentas similares por estarem demasiado disfarçados sobre várias camadas de software.
A ferramenta permite também a instrumentação de várias estatísticas em tempo real relativas ao programa a testar. Estatísticas como: consumo de memória, tempo de CPU despendido, que chamadas de função foram realizadas, etc.[2]
De forma a poder traçar o que está a acontecer, a ferramenta DTrace recorre à monitorização de diversos "sinais"/"pontos de interesse"marcados no sistema operativo a que se dá o nome de **probes**. Estes probes são lançados em momentos específicos e, cabe ao utilizador da ferramenta DTrace, saber quais os probes que deve intercetar e o que fazer quando os intercetar de forma a alcançar os resultados que pretende. Segue-se uma tabela com os probes disponíveis em ambiente Solaris 11 e uma breve descrição: [3]

Common DTrace Providers	Description
dtrace	Start, end and error probes
syscall	Entry and return probes for all system calls
fbt	Entry and return probes for all kernel calls
profile	Timer driven probes
proc	Process creation and lifecycle probes
pid	Entry and return probes for all user-level processes
io	Probes for all I/O related events
sdt/usdt	Developer defined probes at arbitrary locations/names within source code for kernel and user-level processes
sched	Probes for scheduling related events
lockstat	Probes for locking behavior within the operating system

III. Medições de Desempenho - Benchmark Passivo

A. *Estudo do Ambiente Disponível*

De forma a fazer um benchmark passivo da máquina descrita anteriormente utiliza-se, como mencionado anteriormente, a ferramenta IOzone. Inicialmente determina-se a versão instalada com o comando **iozone -v** o que nos permite saber que a versão presente na máquina em questão é a versão 3.434, como é possível comprovar na seguinte imagem que demonstra o output do comando indicado:

```
a72293@solarisEdu:~/a72293$ /opt/csw/bin/iozone -v
       'Iozone' Filesystem Benchmark Program

        Version $Revision: 3.434 $
        Compiled for 64 bit mode.

       Original Author: William Norcott (wnorcott@us.oracle.com)
              4 Dunlap Drive
              Nashua, NH 03060

       Enhancements: Don Capps (capps@iozone.org)
                 7417 Crenshaw
                 Plano, TX 75025

  Copyright 1991, 1992, 1994, 1998, 1999, 2002   William D. Norcott

  License to freely use and distribute this software is hereby granted
  by the author, subject to the condition that this copyright notice
  remains intact.  The author retains the exclusive right to publish
  derivative works based on this work, including, but not limited to,
  revised versions of this work

  Other contributors:

  Don Capps       (Network Appliance)   capps@iozone.org

a72293@solarisEdu:~/a72293$
```

De seguida, com o comando **df -h** foi possível determinar que sistemas de ficheiros estão montados no sistema, podendo assim observar o sistemas sobre os quais os testes serão realizados. Segue-se uma imagem demonstrativa do resultado obtido para este comando:

```
a72293@solarisEdu:~/a72293$ df -h
Filesystem             Size   Used  Available Capacity  Mounted on
rpool/ROOT/solaris     109G    21G        60G    26%    /
/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                   8.6G   1.6M       8.6G     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
                        81G    21G        60G    26%    /lib/libc.so.1
fd                       0K     0K         0K     0%    /dev/fd
rpool/ROOT/solaris/var
                       109G   363M        60G     1%    /var
swap                   9.1G   465M       8.6G     6%    /tmp
rpool/VARSHARE         109G   2.1M        60G     1%    /var/share
rpool/export           109G    32K        60G     1%    /export
rpool/export/home      109G    54K        60G     1%    /export/home
rpool/export/home/a57812
                       109G    35K        60G     1%    /export/home/a57812
rpool/export/home/a64365
                       109G    35K        60G     1%    /export/home/a64365
rpool/export/home/a70377
                       109G    35K        60G     1%    /export/home/a70377
rpool/export/home/a70719
                       109G    20M        60G     1%    /export/home/a70719
rpool/export/home/a70880
                       109G   2.1M        60G     1%    /export/home/a70880
rpool/export/home/a71184
                       109G    35K        60G     1%    /export/home/a71184
rpool/export/home/a71240
                       109G    35K        60G     1%    /export/home/a71240
rpool/export/home/a71492
                       109G   3.1M        60G     1%    /export/home/a71492
rpool/export/home/a71874
                       109G    62K        60G     1%    /export/home/a71874
rpool/export/home/a72227
                       109G    53K        60G     1%    /export/home/a72227
rpool/export/home/a72293
                       109G   105K        60G     1%    /export/home/a72293
rpool/export/home/a72424
                       109G   4.7M        60G     1%    /export/home/a72424
rpool/export/home/a72502
                       109G    35K        60G     1%    /export/home/a72502
rpool/export/home/a73269
                       109G   110K        60G     1%    /export/home/a73269
rpool/export/home/admin
                       109G   975K        60G     1%    /export/home/admin
rpool/export/home/amp
                       109G    11G        60G    16%    /export/home/amp
rpool/export/home/pg31384
                       109G    15M        60G     1%    /export/home/pg31384
rpool/export/home/pg31544
                       109G    35K        60G     1%    /export/home/pg31544
rpool/export/home/pg32995
                       109G    35K        60G     1%    /export/home/pg32995
rpool/export/home/vo   109G    35K        60G     1%    /export/home/vo
rpool                  109G   4.9M        60G     1%    /rpool
rpool/VARSHARE/zones   109G    31K        60G     1%    /system/zones
rpool/VARSHARE/pkg     109G    32K        60G     1%    /var/share/pkg
rpool/VARSHARE/pkg/repositories
                       109G    31K        60G     1%    /var/share/pkg/repositories
172.27.6.237:/storage/san/v2p1
                       1.8T   201G       1.5T    12%    /share/jade
/dev/dsk/c3d0s0        147G   2.3G       143G     2%    /discoHitachi
a72293@solarisEdu:~/a72293$
```

Dentro dos sistemas montados destacam-se os sistemas montados em **/discoHitachi** e **/share/jade** que estão formatados, respetivamente, com sistema de ficheiros **UFS** e **NFS**, permitindo assim testar resultados para estes dois tipos de sistemas de ficheiros. Podemos ainda testar o sistema de ficheiros **ZFS** montado noutras partições presentes na máquina como, por exemplo, em **home/a72293**. Ficam assim estabelecidos os três tipos de sistemas de ficheiros que serão testados ao longo deste documento.

Torna-se agora importante referir que, para cada um destes três tipos de sistemas de ficheiro disponíveis teve-se a intenção de realizar três testes principais com IOzone:

- **1 -**
  Primeiro Será testado em ambiente single-threaded/single-process os seguintes testes: write e rewrite, read e reread, random read e random write, backwards read e stride. Para isto as flags **-i0 -i1 -i2 -i5** serão utilizadas. Com este teste poder-se-á observar em pormenor a performance da máquina na escrita de um ficheiro não existente e na escrita num ficheiro já existente, a leitura de um ficheiro não lido anteriormente e da leitura de um ficheiro lido recentemente, a leitura e escrita de um ficheiro em pontos aleatórios e a leitura de um ficheiro com um certo espaçamento. Será apresentado um gráfico individual para cada um destes testes.
- **2 -**
  De seguida utilizar-se-á as flags **-i0 -i1 -i2 -i5 -l# -u#** de maneira a realizar os testes em ambiente multi-threaded/multi-proccess. Desta forma podemos ver como a máquina se comporta em paralelo. Tendo em conta que máquina tem oito cores físicos então torna-se relevante testar desde um a oito threads/processos de forma a ver quão bem escala com o número de cores físicos.
- **3 -**
  Por último, depois de se ter um melhor conhecimento da performance da máquina, utilizar-se-á as flags **-i0 -i1 -i2 -i3 -i5 -I** de maneira a realizar os testes utilizando DIRECT I/O em todas as operações de ficheiros. Desta forma podemos ver como a máquina se comporta se tiver que ir sempre ao disco em cada operação de ficheiros.

Infelizmente o não foi possível realizar o terceiro teste para o sistema de ficheiros ZFS, podemos ver no seguinte output que **directio** não se encontrou disponível:

```
a72293@solarisEdu:~/a72293£ /opt/csw/bin/iozone -a -+u -i0 -i1 -i2 -i5 -R -b z2 -s524288 -I
        Iozone: Performance Test of File I/O
                Version £Revision: 3.434 £
                Compiled for 64 bit mode.
                Build: Solaris10

        Contributors:William Norcott, Don Capps, Isom Crawford, Kirby Collins
                     Al Slater, Scott Rhine, Mike Wisner, Ken Goss
                     Steve Landherr, Brad Smith, Mark Kelly, Dr. Alain CYR,
                     Randy Dunlap, Mark Montague, Dan Million, Gavin Brebner,
                     Jean-Marc Zucconi, Jeff Blomberg, Benny Halevy, Dave Boone,
                     Erik Habbinga, Kris Strecker, Walter Wong, Joshua Root,
                     Fabrice Bacchella, Zhenghua Xue, Qin Li, Darren Sawyer,
                     Vangel Bojaxhi, Ben England, Vikentsi Lapa,
                     Alexey Skidanov.

        Run began: Wed May  3 18:44:02 2017

        Auto Mode
        CPU utilization Resolution = 0.000 seconds.
        CPU utilization Excel chart enabled
        Excel chart generation enabled
        File size set to 524288 kB
        O_DIRECT feature enabled
        Command line used: /opt/csw/bin/iozone -a -+u -i0 -i1 -i2 -i5 -R -b z2 -s524288 -I
        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.
                                                                random    random     bkwd    r

                kB  reclen    write  rewrite    read    reread    read     write     read   re
    fread  freread
            524288       4
directio not available.
```

Entenda-se que nenhuma informação sobre a cache da máquina em questão será fornecida à ferramenta IOzone pois esta serve para testar o desempenho da máquina para aplicações potencialmente genéricas que não irão, por razões óbvias, conhecer esses valores. Ao não fornecer valores da cache (como o tamanho de uma linha e o tamanho de um set) simulamos melhor o funcionamento com aplicações genéricas e os resultados são assim mais relevantes. Quando nenhum valor é fornecido, para esta máquina, o IOzone assume um tamanho de linha de cache de 32 bytes e um tamanho de set de 1024 KBytes, como podemos ver na seguinte imagem que demonstra um exemplo do output inicial da ferramenta IOzone:

```
Auto Mode
Command line used: /opt/csw/bin/iozone -a -i0 -i2 -b ver2.csv
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.
```

Por fim pode-se ler nas regras de utilização da ferramenta IOzone [4] que, para o modo automático, o tamanho do buffer da cache deve ser menor ou igual a 128 Megabytes de forma a garantir que são realizados testes dentro e fora do buffer

da cache. Utilizando o comando **sysdef** obtemos o seguinte output:

```
*
* Tunable Parameters
*
513724416       maximum memory allowed in buffer cache (bufhwm)
   30000        maximum number of processes (v.v_proc)
      99        maximum global priority in sys class (MAXCLSYSPRI)
   29995        maximum processes per user id (v.v_maxup)
      30        auto update time limit in seconds (NAUTOUP)
      25        page stealing low water mark (GPGSLO)
       1        fsflush run rate (FSFLUSHR)
      25        minimum resident memory for avoiding deadlock (MINARMEM)
      25        minimum swapable memory for avoiding deadlock (MINASMEM)
*
* Utsname Tunables
*
```

Por aqui vemos que o tamanho do buffer disponível é 513724416 KB = 501684 MB. No entanto o tamanho de buffer a usar neste cálculo não será diretamente o valor apresentado na imagem anterior, pois sabemos que o tamanho do buffer é truncado se ultrapassar o menor dos seguintes valores: [5] 20% da memória física, 2 TB, ou 1/4 do tamanho máximo da kernel heap. Por esta lógica temos que o tamanho resultante do buffer em questão é 13107,3 MB. Sendo este valor bastante superior a 128 MB então percebemos que os testes realizados com o modo automático não iriam mostrar resultados que caiam fora da cache, por essa razão o método automático não foi utilizado e teve que se escolher o tamanho para os ficheiros de teste. Voltando a olhar para as regras do uso de IOzone [4] vemos que, para comparação de performance entre vários discos se aconselha um tamanho de ficheiro de pelo menos 3*tamanho do buffer da cache disponível. Sendo 3*tamanho do buffer = 39321,6 MB, escolheu-se então um tamanho de ficheiro para teste que fosse maior do que 39321,6 MB. Um tamanho relevante encontrado foi de **40GB**, decidiu-se ainda utilizar um tamanho de **20 GB** de forma a notar a possível aceleração ao testar com um ficheiro muito menor do que os 3*buffer cache aconselhados. Quanto ao record (tamanho de cada leitura/escrita num teste) optou-se pelas opções fornecidas com a flag -a que irá testar para os seguintes tamanhos: **4, 8, 16, 32, 64, 128, 256, 512, 1024, 2048, 4096, 8192, 16384 KB**. Não se achou relevante testar com maiores records, pois poucas aplicações escrevem ou leem mais do que 0.5 GB de uma só vez, para além disso, estes testes cobrem uma vasta gama de records permitindo ver o comportamento da máquina com records muito pequenos, médios e grandes. É ainda utilizada a opção -+u de forma a obter informação relativa à utilização do CPU.

B. Análise dos Resultados com Benchmark Passivo - Sistema ZFS

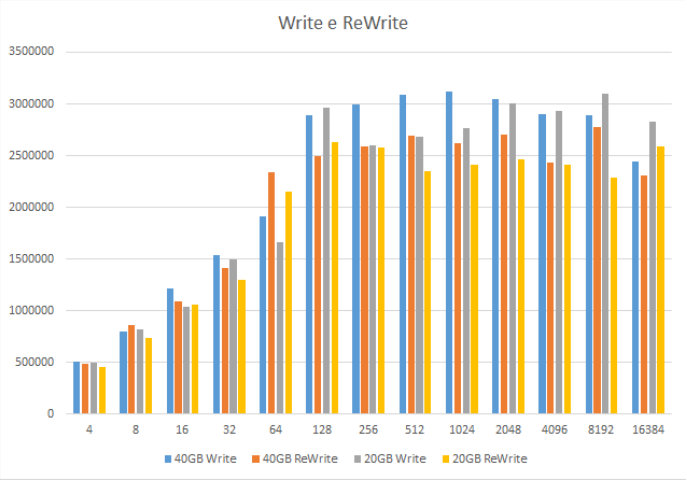
Mostram-se e analisam-se agora os resultados para os testes descritos no subcapítulo anterior.

TESTE 1:

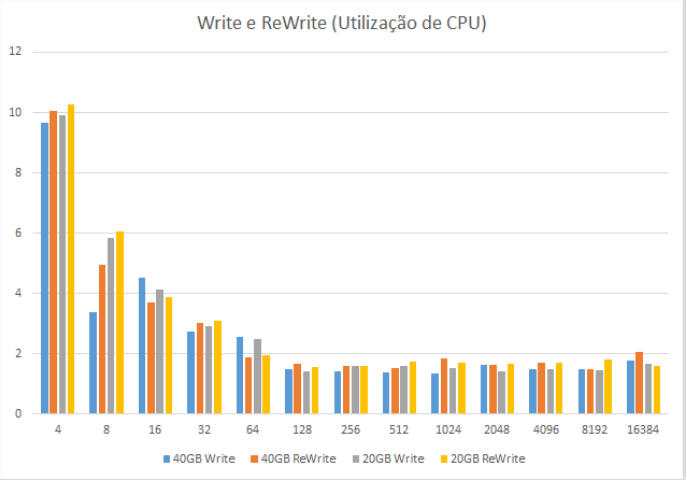
Usando os comandos:
/opt/csw/bin/iozone -+u -a -i0 -R -b teste1ZFS40.xls -s 40g
/opt/csw/bin/iozone -+u -a -i0 -R -b teste1ZFS20.xls -s 20g

obtiveram-se os seguintes gráficos:

(note-se que neste primeiro gráfico o eixo y representa KB/s e o eixo x o tamanho do record testado)



(Neste segundo gráfico o eixo y representa a percentagem de utilização do CPU e o eixo x o tamanho do record testado)



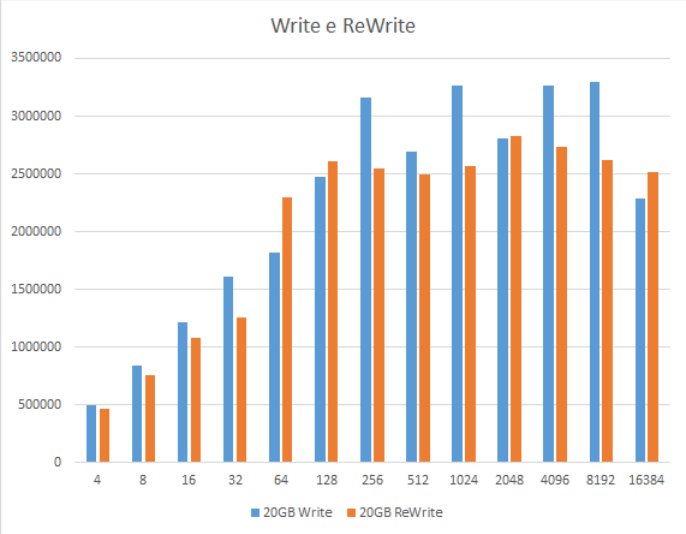
Olhando para estes gráfico que relatam os resultados para os testes de write e rewrite obtidos com -i0, observa-se que, de forma geral, para ambos os tamanhos de ficheiro (40 GB e 20 GB), há um menor número de KB por segundo para testes de rewrite e uma maior percentagem de utilização do CPU, no entanto estes resultados são muito contra intuitivos, visto que, aquando do primeiro write temos um overhead chamado 'metadata' que representa o impacto na performance de ter de guardar os dados escritos no ficheiro e de registar onde os dados se localizam no disco. Quando fazemos um rewrite este overhead não ocorre e temos ainda o ficheiro em cache, por esta razão rewrite devia ter um maior número de KB por segundo e menor percentagem de utilização do CPU do que o write. De forma a tentar compreender melhor o que ocorreu correu-se o seguinte programa:

/opt/csw/bin/iozone -a -i0 -R -b teste1ZFS.xls -s 40g -N

Note-se o uso da flag **-N** que nos permitirá obter os resultados em microsegundos por operação. O seguinte gráfico foi obtido:

(note-se que neste gráfico o eixo y representa

microsegundos/operação e o eixo x o tamanho do record testado)



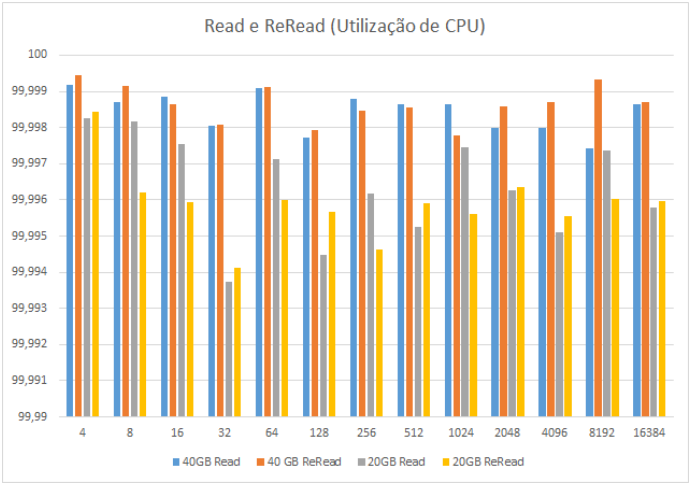
Neste gráfico vemos que, de forma geral, os testes para uma primeira escrita (write) demoram mais tempo por operação, isto realmente seria de esperar. Talvez os resultados contra intuitivos obtidos nos gráficos anteriores possam ser resultado de uma situação esporádica fora do comum, como por exemplo o disco estar mais quente aquando dos testes de rewrite do que nos testes de write o que poderia levar a uma pior performance.

Correndo agora os comandos:
/opt/csw/bin/iozone -+u -a -i0 -i1 -R -b teste12ZFS40.xls -s 40g
/opt/csw/bin/iozone -+u -a -i0 -i1 -R -b teste12ZFS20.xls -s 20g

Foram obtidos os seguintes gráficos:

(note-se que neste primeiro gráfico o eixo y representa KB/s e o eixo x o tamanho do record testado)

(Neste segundo gráfico o eixo y representa a percentagem de utilização do CPU e o eixo x o tamanho do record testado)



Pelo primeiro gráfico facilmente se observa que a operação de read resulta numa pior performance do que reread já que é realizada com um número inferior de KB/s. Isto seria de esperar pois várias leituras consecutivas (reread) irão tirar um melhor proveito da cache do que uma primeira leitura isolada. Pelo segundo gráfico pouco se conclui pois embora graficamente os valores pareçam oscilar bastante, ao olhar para os valores no eixo y verifica-se que na verdade estão todos extremamente próximos uns dos outros, já que variam só na ordem das milésimas, o que leva a crer que, a nível de utilização do CPU não há muita diferença entre read e reread. Podemos ainda notar que estas operações de read e reread alcançaram um uso de CPU muito superior às operações de write e rewrite, mas tendo em conta que com read e reread também se obteve um maior número de KB/s em praticamente todas as situações então não é de estranhar que tenha havido um maior uso do CPU.

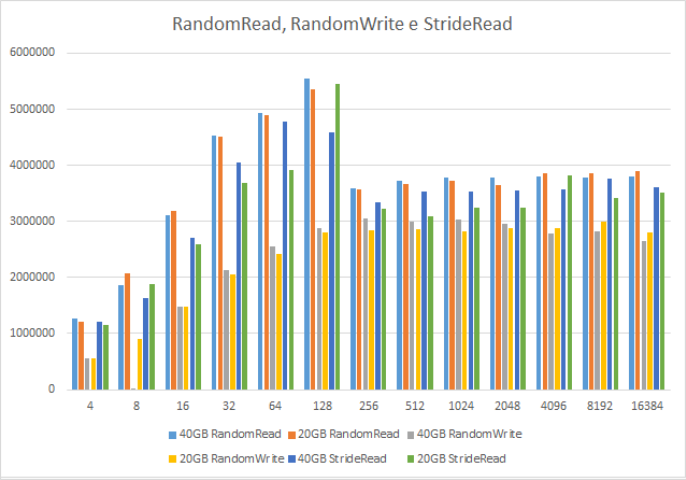
Correndo agora os seguintes comandos (note-se o uso de vários comandos em vez de um só com várias flags para evitar que testes para as flags finais beneficiem demasiado do uso da cache o que poderia retornar resultados um pouco irrealistas não replicáveis numa aplicação genérica):

/opt/csw/bin/iozone -+u -a -i0 -i2 -R -b teste131ZFS40.xls -s 40g
/opt/csw/bin/iozone -+u -a -i0 -i5 -R -b teste133ZFS40.xls -s 40g

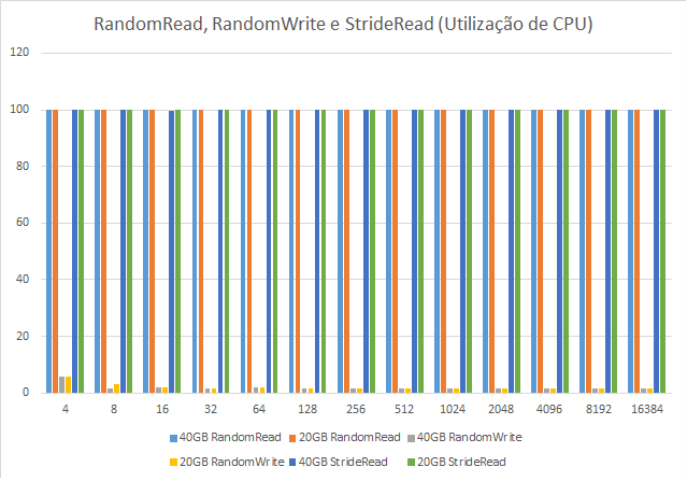
/opt/csw/bin/iozone -+u -a -i0 -i2 -R -b teste131ZFS20.xls -s 20g
/opt/csw/bin/iozone -+u -a -i0 -i5 -R -b teste133ZFS20.xls -s 20g

Foram obtidos os seguintes gráficos:

(note-se que neste primeiro gráfico o eixo y representa KB/s e o eixo x o tamanho do record testado)



(Neste segundo gráfico o eixo y representa a percentagem de utilização do CPU e o eixo x o tamanho do record testado)



Nota-se que há um maior número de KB por segundo para as operações de RandomRead e de StrideRead do que para a operação de RandomWrite, isto mantêm-se coerente com os resultados obtidos nos gráficos anteriores onde a mesma relação se encontrava entre read, reread e write, rewrite. De facto é intuitivo que uma leitura obtenha uma melhor performance que uma escrita. Mais uma vez se nota para as operações de escrita uma muito menor utilização do CPU do que para as operações de escrita mantendo-se coerente com os os resultados anteriores.

Num panorama geral, olhando agora para todos os gráficos até agora, Podemos derivar algumas conclusões. Para tamanhos de record muito pequenos nota-se uma melhor performance com o ficheiro de 40GB, mas para tamanhos grandes de record nota-se que o ficheiro de 20GB obtém melhores resultados, provavelmente o facto de ser mais pequeno proporciona um ganho por parte do uso da cache mais acentuado permitindo que este ficheiro escale melhor para maiores records. Nota-se ainda que, no geral, a partir de records de 256KB se nota uma descida na performance para os dois ficheiros.

TESTE 2:

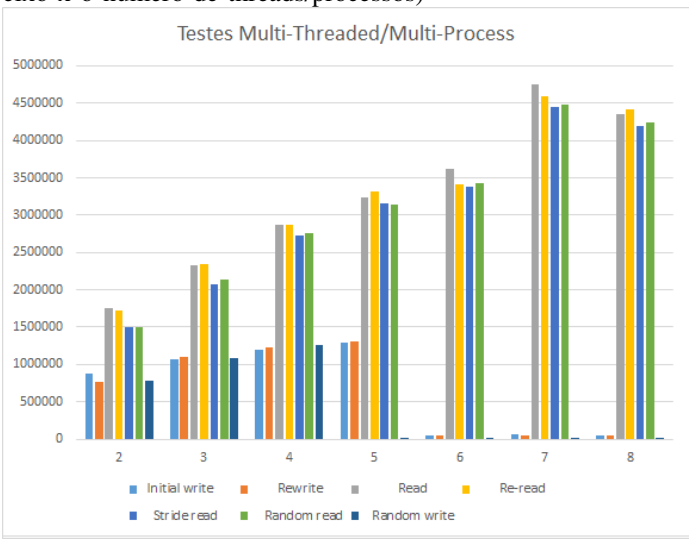
Testa-se agora o comportamento multi-threaded/multi-processo, mas tendo em conta o grande número de testes a realizar para este comportamento apenas se usa um tamanho de record (4KB) e de ficheiro (20GB) para teste para cada thread/processo.

Correndo agora os comandos:

/opt/csw/bin/iozone -i0 -R -b teste21ZFS20.xls -s 20g -l2 -u8
/opt/csw/bin/iozone -i0 -i1 -R -b teste22ZFS20.xls -s 20g -l2 -u8
/opt/csw/bin/iozone -i0 -i2 -R -b teste231ZFS20.xls -s 20g -l2 -u8
/opt/csw/bin/iozone -i0 -i5 -R -b teste233ZFS20.xls -s 20g -l2 -u8

foi possível obter o seguinte gráfico:

(note-se que neste gráfico o eixo y representa KB/s e o eixo x o número de threads/processos)



Observa-se que, com a exceção de com 2 threads/processos, se obtém uma maior performance ao fazer um rewrite do que um initial write, o que, pelas razões já indicadas anteriormente seria de esperar. No panorama geral vê-se também de novo que se obtém maior performance com reread do que com read.

Nota-se também que com qualquer um dos números de threads/processos testado se obtém uma performance bastante superior ao obtido com os testes single-threaded/single-proccess. Isto seria de esperar já que se está a usar um número de threads <= ao número de cores físicos na máquina. Isto apenas não acontece para os testes de Initial Write, ReWrite e Random Write já que vemos uma quebra bastante acentuada na performance destes testes a partir das 6 threads/processos o que leva a crer que operações de escrita, nesta máquina, têm o seu pico de escalabilidade por volta das 5 threads/processos. Vemos que escalam muito menos nesta máquina do que operações de leitura já que estas têm um speed-up bastante acentuado até às 8 threads/processos não inclusive. Há uma descida na performance para as operações

de leitura ao usar 8 threads/processos o que indica que, nesta máquina, o pico de escalabilidade para estas operações de leitura são as 7 threads/processos.

TESTE 3:

Como mencionado no subcapítulo anterior não foi possível realizar este teste nesta partição.

C. Análise dos Resultados com Benchmark Passivo - Sistema UFS

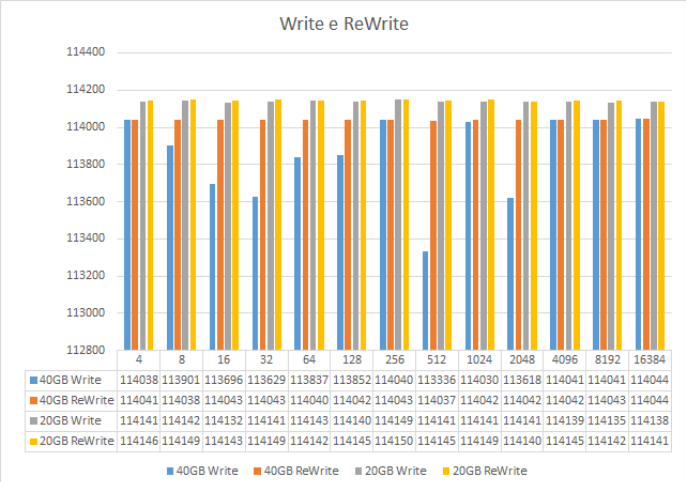
TESTE 1:

Usando os comandos:
/opt/csw/bin/iozone -f /share/jade/iozone.tmp -+u -a -i0 -R -b teste1UFS40.xls -s 40g
/opt/csw/bin/iozone -f /share/jade/iozone.tmp -+u -a -i0 -i1 -R -b teste12UFS40.xls -s 40g
/opt/csw/bin/iozone -f /share/jade/iozone.tmp -+u -a -i0 -i2 -R -b teste131UFS40.xls -s 40g
/opt/csw/bin/iozone -f /share/jade/iozone.tmp -+u -a -i0 -i5 -R -b teste133UFS40.xls -s 40g

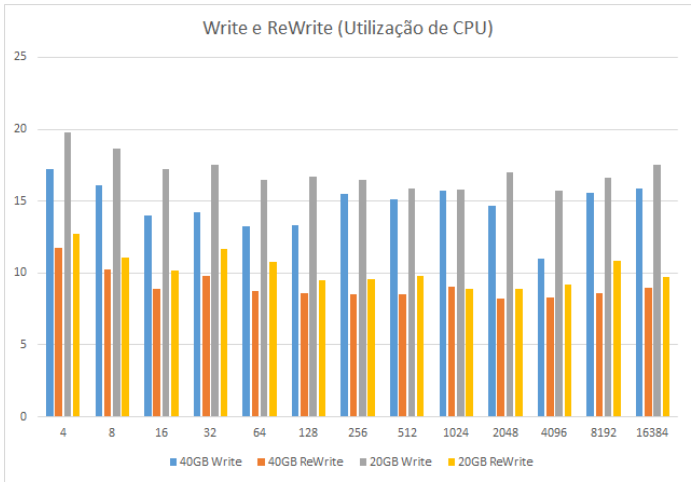
/opt/csw/bin/iozone -f /share/jade/iozone.tmp -+u -a -i0 -R -b teste1UFS20.xls -s 20g
/opt/csw/bin/iozone -f /share/jade/iozone.tmp -+u -a -i0 -i1 -R -b teste12UFS20.xls -s 20g
/opt/csw/bin/iozone -f /share/jade/iozone.tmp -+u -a -i0 -i2 -R -b teste131UFS20.xls -s 20g
/opt/csw/bin/iozone -f /share/jade/iozone.tmp -+u -a -i0 -i5 -R -b teste133UFS20.xls -s 20g

obtiveram-se os seguintes gráficos:

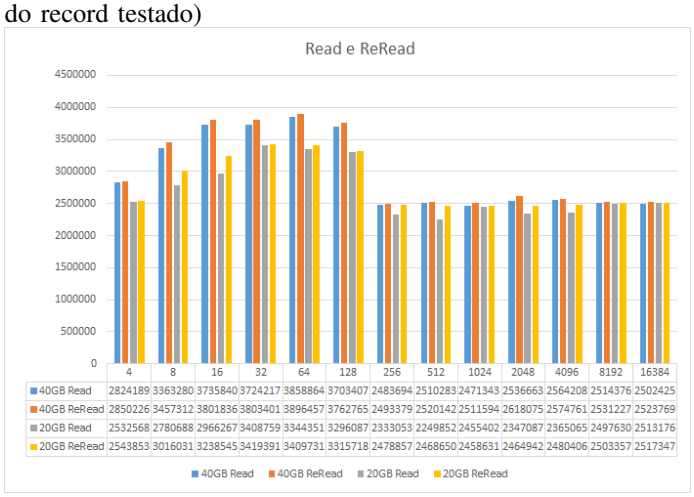
(Neste gráfico o eixo y representa KB/s e o eixo x o tamanho do record testado)



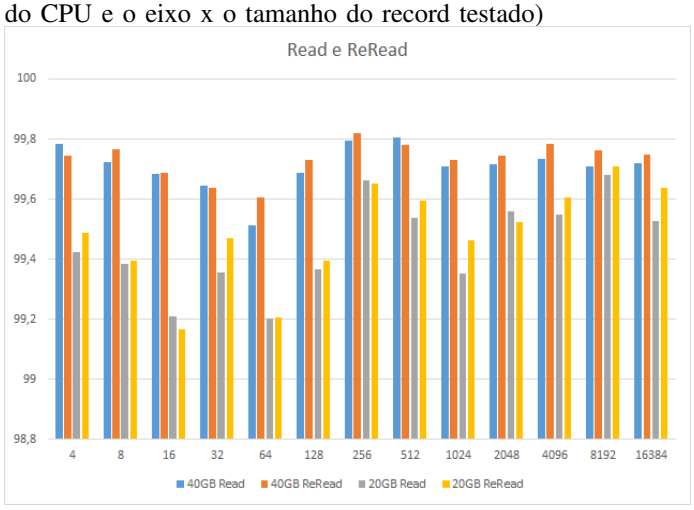
(Neste gráfico o eixo y representa a percentagem de utilização do CPU e o eixo x o tamanho do record testado)



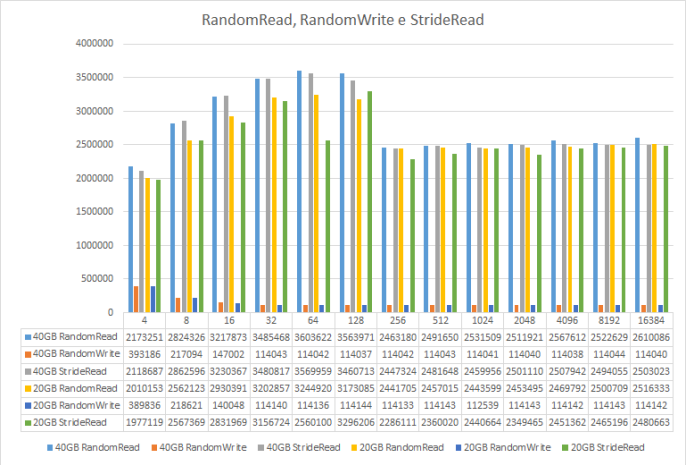
(Neste gráfico o eixo y representa KB/s e o eixo x o tamanho do record testado)



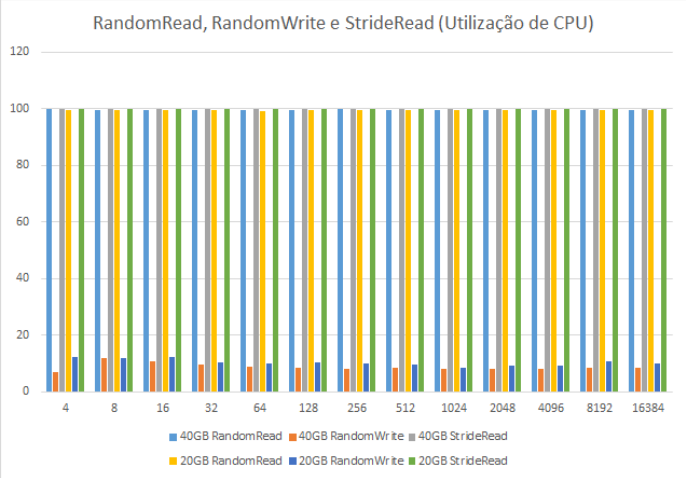
(Neste gráfico o eixo y representa a percentagem de utilização do CPU e o eixo x o tamanho do record testado)



(Neste gráfico o eixo y representa KB/s e o eixo x o tamanho do record testado)



(Neste gráfico o eixo y representa a percentagem de utilização do CPU e o eixo x o tamanho do record testado)



Olhando para os gráficos de um panorama geral vemos que as conclusões alcançadas para os gráficos do ssitema de ficheiros testado anteriormente (ZFS) se mantêm. Notamos apenas que, no primeiro gráfico (Write e ReWrite), se obtém uma maior performance com ReWrite do que Write que era o esperado, mas que não estava a acontecer nos primeiros testes ao sistema de ficheiros ZFS

Comparando agora os resultados com os obtidos no sistema de ficheiros anterior nota-se que o sistema anterior permitiu consistentemente a obtenção de uma maior performance já que os valores de KB por segundo eram bastante superiores em todos os gráficos. Da mesma maneira que aconteceu no sistema de ficheiros anterior, a partir de records de tamanho 256KB nota-se uma queda abrupta na performance, sendo que até esse tamanho se vai notando subidas na performance na maior parte dos casos. Isto indica novamente que este tamanho de record poderá ser o limite da escalabilidade para este tipo de testes nesta máquina para ambos estes dois discos e tipos de sistemas de ficheiros. Novamente se nota, como esperado, uma melhor performance por parte das leituras do que por parte das escritas.

TESTE 2:

Testa-se agora o comportamento multi-threaded/multi-processo, mas tendo em conta o grande número de testes a realizar para este comportamento apenas se usa um tamanho de record (4KB) e de ficheiro (20GB) para teste para cada thread/processo.

Correndo agora os comandos:

/opt/csw/bin/iozone -i0 -R -b teste21UFS20.xls -s 20g -l2 -u8 -F /share/jade/iozone1.tmp /share/jade/iozone2.tmp /share/jade/iozone3.tmp /share/jade/iozone4.tmp /share/jade/iozone5.tmp /share/jade/iozone6.tmp /share/jade/iozone7.tmp /share/jade/iozone8.tmp

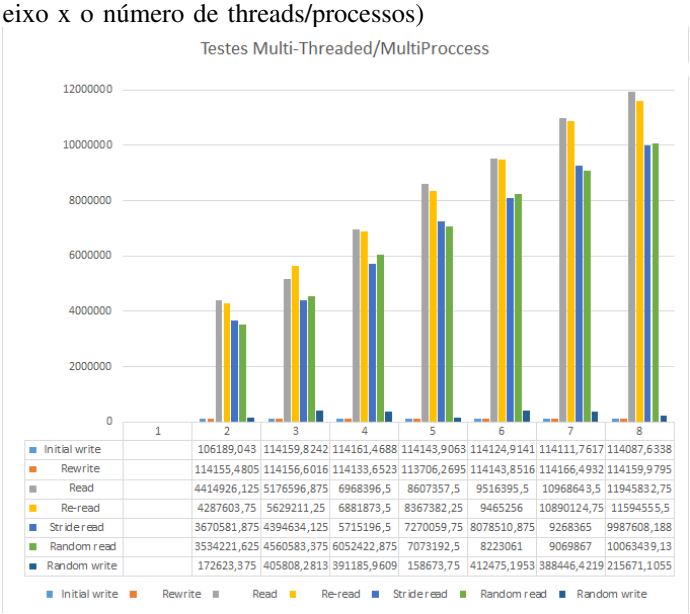
/opt/csw/bin/iozone -i0 -i1 -R -b teste22UFS20.xls -s 20g -l2 -u8 -F /share/jade/iozone1.tmp /share/jade/iozone2.tmp /share/jade/iozone3.tmp /share/jade/iozone4.tmp /share/jade/iozone5.tmp /share/jade/iozone6.tmp /share/jade/iozone7.tmp /share/jade/iozone8.tmp

/opt/csw/bin/iozone -i0 -i2 -R -b teste231UFS20.xls -s 20g -l2 -u8 -F /share/jade/iozone1.tmp /share/jade/iozone2.tmp /share/jade/iozone3.tmp /share/jade/iozone4.tmp /share/jade/iozone5.tmp /share/jade/iozone6.tmp /share/jade/iozone7.tmp /share/jade/iozone8.tmp

/opt/csw/bin/iozone -i0 -i5 -R -b teste233UFS20.xls -s 20g -l2 -u8 -F /share/jade/iozone1.tmp /share/jade/iozone2.tmp /share/jade/iozone3.tmp /share/jade/iozone4.tmp /share/jade/iozone5.tmp /share/jade/iozone6.tmp /share/jade/iozone7.tmp /share/jade/iozone8.tmp

foi possível obter o seguinte gráfico:

(note-se que neste gráfico o eixo y representa KB/s e o eixo x o número de threads/processos)



Neste teste facilmente notamos que este sistema de ficheiros consegue alcançar uma performance muito superior ao sistema ZFS em ambiente multi-threaded/multi-proccess. Notamos que, embora as operações de escrita não sofram muita variação de speed-up para um diferente número de threads/processos,

as operações de leitura sofrem um speed-up brusco com o aumento de threads/processos não dando possibilidade de detetar o limite da escalabilidade (Ao contrário do sistema ZFS onde a escalabilidade tinha o seu pico nas 7 threads/processos). Por aqui podemos concluir então que, se o objetivo for realizar operações em ficheiros em ambiente multi-threaded/multi-processos então é vantajoso o sistema de ficheiros UFS, caso contrário, em ambiente single-threaded/single-proccess então ZFS será uma escolha mais acertada.

TESTE 3:

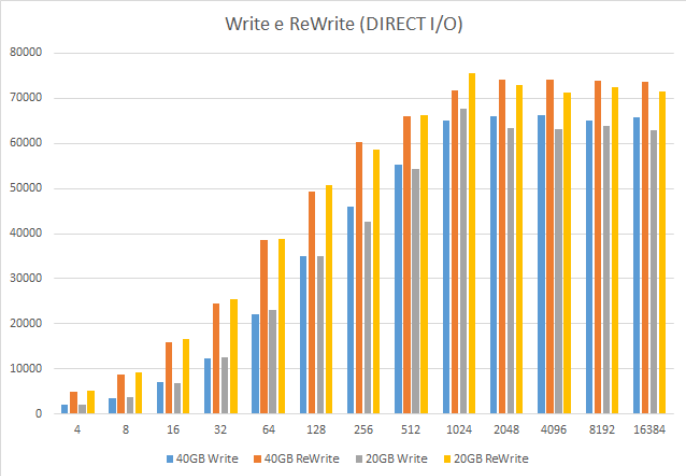
Testa-se agora o terceiro teste que não foi possível realizar para o sistema de ficheiros ZFS, como mencionado anteriormente.

Usando os comandos:
/opt/csw/bin/iozone -f /share/jade/iozone.tmp -+u -a -i0 -R -b teste3UFS40.xls -s 40g -I
/opt/csw/bin/iozone -f /share/jade/iozone.tmp -+u -a -i0 -i1 -R -b teste32UFS40.xls -s 40g -I
/opt/csw/bin/iozone -f /share/jade/iozone.tmp -+u -a -i0 -i2 -R -b teste331UFS40.xls -s 40g -I
/opt/csw/bin/iozone -f /share/jade/iozone.tmp -+u -a -i0 -i5 -R -b teste333UFS40.xls -s 40g -I

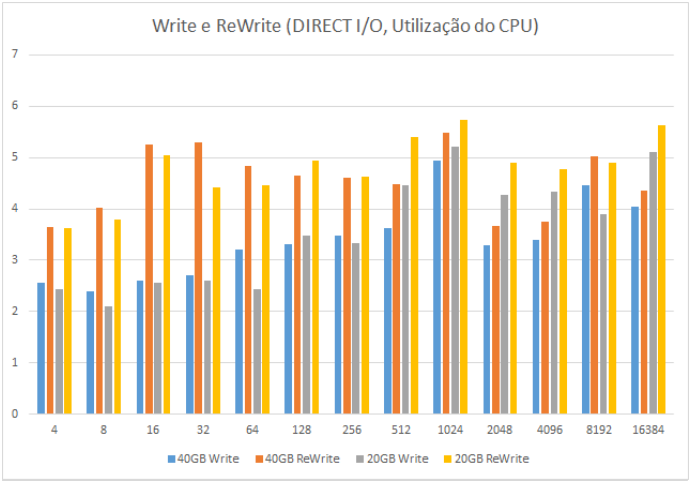
/opt/csw/bin/iozone -f /share/jade/iozone.tmp -+u -a -i0 -R -b teste3UFS20.xls -s 20g -I
/opt/csw/bin/iozone -f /share/jade/iozone.tmp -+u -a -i0 -i1 -R -b teste32UFS20.xls -s 20g -I
/opt/csw/bin/iozone -f /share/jade/iozone.tmp -+u -a -i0 -i2 -R -b teste331UFS20.xls -s 20g -I
/opt/csw/bin/iozone -f /share/jade/iozone.tmp -+u -a -i0 -i5 -R -b teste333UFS20.xls -s 20g -I

obtiveram-se os seguintes gráficos:

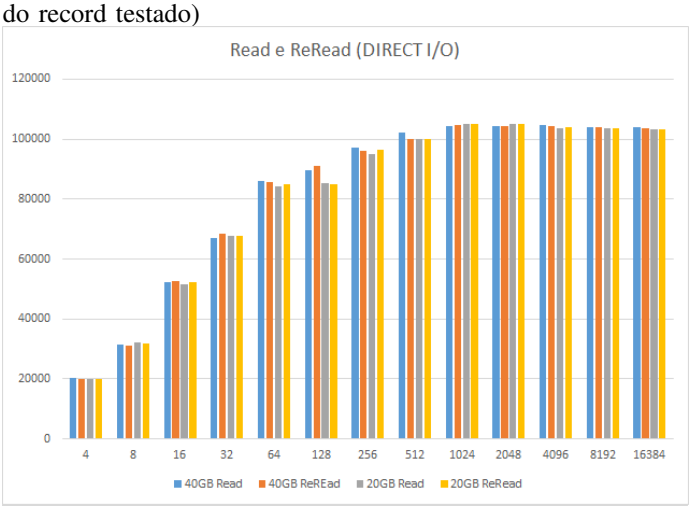
(Neste gráfico o eixo y representa KB/s e o eixo x o tamanho do record testado)



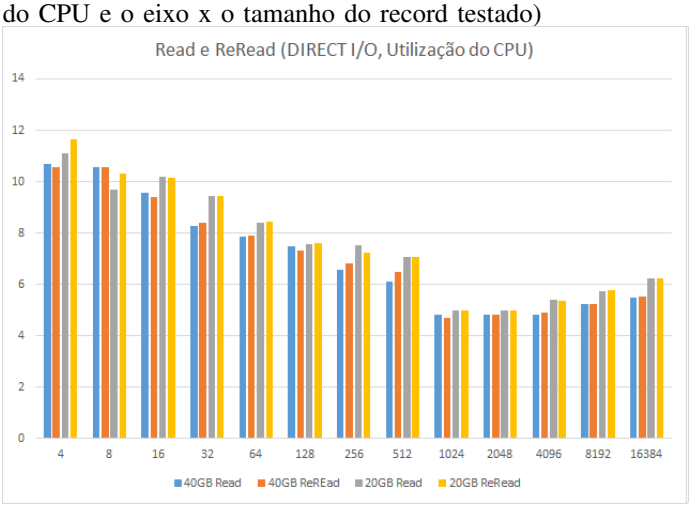
(Neste gráfico o eixo y representa a percentagem de utilização do CPU e o eixo x o tamanho do record testado)



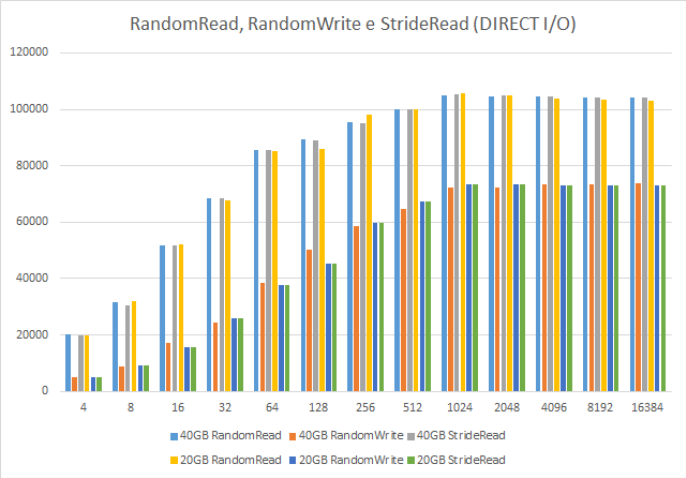
(Neste gráfico o eixo y representa KB/s e o eixo x o tamanho do record testado)



(Neste gráfico o eixo y representa a percentagem de utilização do CPU e o eixo x o tamanho do record testado)



(Neste gráfico o eixo y representa KB/s e o eixo x o tamanho do record testado)



(Neste gráfico o eixo y representa a percentagem de utilização do CPU e o eixo x o tamanho do record testado)

Neste teste nota-se que é obtida uma performance muito inferior à performance obtida neste sistema de ficheiros sem o uso de DIRECT I/O. Isto é um comportamento que já se esperava, pois acessos diretos ao disco em cada operação com ficheiros têm um impacto na performance muito superior a simplesmente aceder, por exemplo, a uma cache externa ao disco. No entanto note-se ainda que, ao contrário de quando não temos acessos diretos ao disco, não se nota o limite de escalabilidade para o tamanho de record. Isto leva a querer que esse limite de escalabilidade estava a acontecer por se estarem a usar tamanhos de record demasiado grandes para se conseguir tirar completo partido dos benefícios da cache da mesma forma que se obtinha para records menores. Como estes testes não utilizam uma cache e vão diretos ao disco então nunca há benefícios desses e por isso estima-se que a performance não sofra um impacto tão acentuado assim de repente como sofria sem acessos diretos ao disco.

D. Análise dos Resultados com Benchmark Passivo - Sistema NFS

TESTE 1:

Usando os comandos:

/opt/csw/bin/iozone -f /discoHitachi/iozone.tmp -+u -a -i0 -R -b teste1NFS40.xls -s 40g
/opt/csw/bin/iozone -f /discoHitachi/iozone.tmp -+u -a -i0 -i1 -R -b teste12NFS40.xls -s 40g
/opt/csw/bin/iozone -f /discoHitachi/iozone.tmp -+u -a -i0 -i2 -R -b teste131NFS40.xls -s 40g
/opt/csw/bin/iozone -f /discoHitachi/iozone.tmp -+u -a -i0 -i5 -R -b teste133NFS40.xls -s 40g

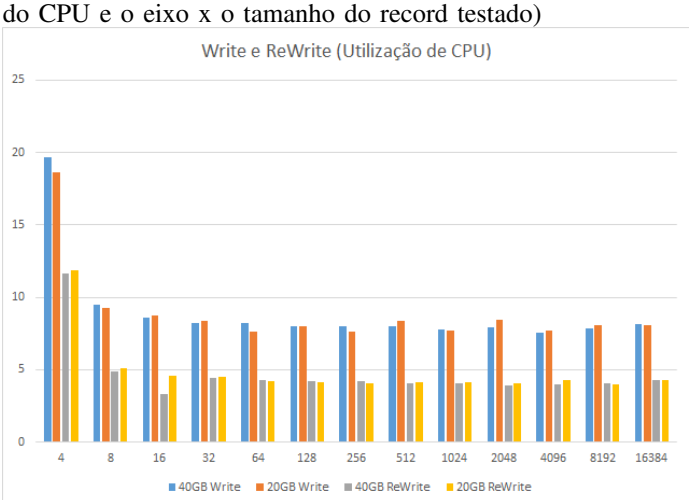
/opt/csw/bin/iozone -f /discoHitachi/iozone.tmp -+u -a -i0 -R -b teste1NFS20.xls -s 20g
/opt/csw/bin/iozone -f /discoHitachi/iozone.tmp -+u -a -i0 -i1 -R -b teste12NFS20.xls -s 20g
/opt/csw/bin/iozone -f /discoHitachi/iozone.tmp -+u -a -i0 -i2 -R -b teste131NFS20.xls -s 20g
/opt/csw/bin/iozone -f /discoHitachi/iozone.tmp -+u -a -i0 -i5 -R -b teste133NFS20.xls -s 20g

obtiveram-se os seguintes gráficos:

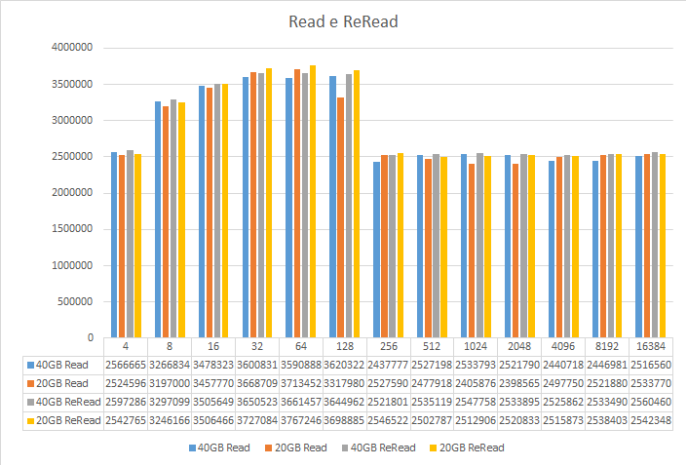
(Neste gráfico o eixo y representa KB/s e o eixo x o tamanho do record testado)

	4	8	16	32	64	128	256	512	1024	2048	4096	8192	16384
40GB Write	67010	72159	71948	72079	72089	71686	71679	71912	72151	72646	72295	71988	72074
20GB Write	64217	72276	71972	71960	71403	72181	72394	72410	72151	71888	72234	72097	72411
40GB ReWrite	72098	70999	72571	73707	71568	71371	72588	72819	74634	69865	71644	72603	73214
20GB ReWrite	75642	72886	73258	73532	74119	72380	72681	75170	74322	75005	74650	74950	77696

(Neste gráfico o eixo y representa a percentagem de utilização do CPU e o eixo x o tamanho do record testado)

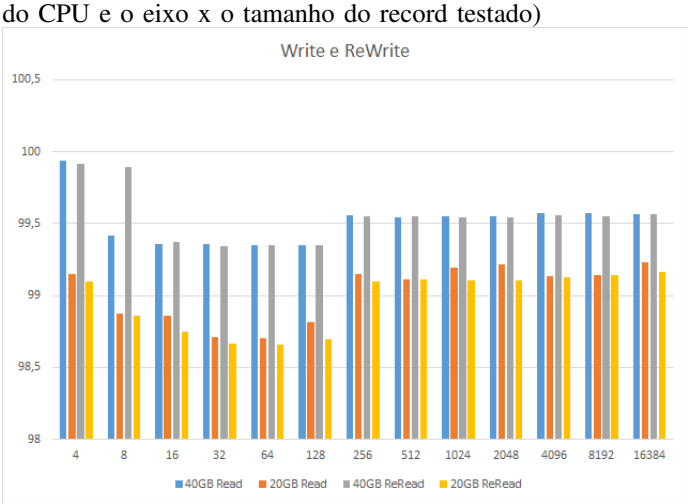


(Neste gráfico o eixo y representa KB/s e o eixo x o tamanho do record testado)

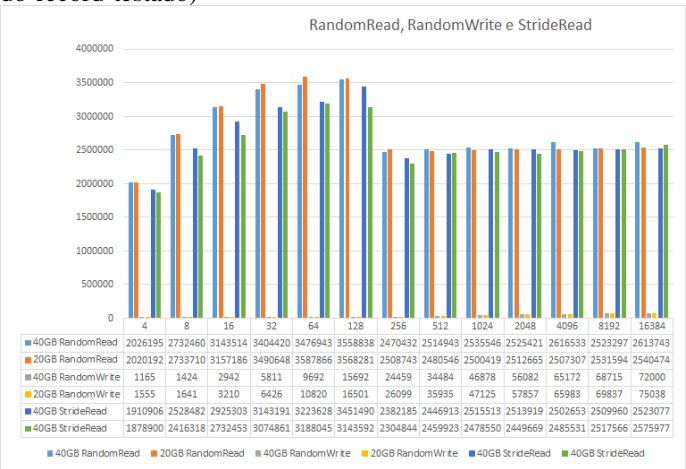


	4	8	16	32	64	128	256	512	1024	2048	4096	8192	16384
40GB Read	2566665	3266834	3478323	3600831	3590888	3620322	2437777	2527198	2533793	2521790	2440718	2446981	2516560
20GB Read	2524596	3197000	3457770	3668709	3713452	3317980	2527590	2477918	2405876	2398565	2497750	2521880	2533770
40GB ReRead	2597286	3297099	3505649	3650523	3661457	3644962	2521801	2535119	2547758	2533895	2525862	2533490	2560460
20GB ReRead	2542765	3246166	3506466	3727084	3767246	3698885	2546522	2502787	2512906	2520833	2515873	2538403	2542348

(Neste gráfico o eixo y representa a percentagem de utilização do CPU e o eixo x o tamanho do record testado)

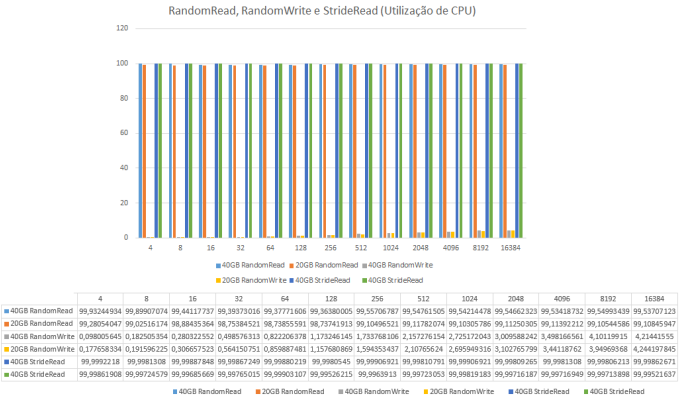


(Neste gráfico o eixo y representa KB/s e o eixo x o tamanho do record testado)



	4	8	16	32	64	128	256	512	1024	2048	4096	8192	16384
40GB RandomRead	2026195	2732460	3143514	3404420	3476943	3558838	2470432	2514943	2535546	2525421	2616533	2523297	2613743
20GB RandomRead	2020192	2733710	3157186	3490648	3587866	3568281	2508743	2480546	2500419	2512665	2507307	2531594	2540474
40GB RandomWrite	1165	1424	2942	5811	9692	15692	24459	34484	46878	56082	65172	68715	72000
20GB RandomWrite	1555	1641	3210	6426	10820	16501	26099	35935	47125	57857	65983	69837	75038
40GB StrideRead	1910906	2528482	2925303	3143191	3223628	3451490	2382185	2446913	2515513	2513919	2502653	2509960	2523077
40GB StrideRead	1878900	2416318	2732453	3074861	3188045	3143592	2304844	2459923	2478550	2449669	2485531	2517566	2575977

(Neste gráfico o eixo y representa a percentagem de utilização do CPU e o eixo x o tamanho do record testado)



	4	8	16	32	64	128	256	512	1024	2048	4096	8192	16384
40GB RandomRead	99,93244934	99,89907074	99,44117737	99,39373016	99,37771606	99,36380005	99,55706787	99,54761505	99,54214478	99,54662323	99,53418732	99,54931439	99,53707123
20GB RandomRead	99,28054047	99,02516174	98,88435364	98,75384521	98,73855591	98,73741913	99,10496521	99,11782074	99,10305786	99,11250305	99,11392212	99,10544586	99,10845947
40GB RandomWrite	0,098005645	0,182505354	0,280322552	0,498576313	0,822206378	1,173246145	1,733768106	2,157276154	2,725172043	3,009588242	3,498166561	4,10119915	4,21441555
20GB RandomWrite	0,177658334	0,191596225	0,306657523	0,564150751	0,859887481	1,157680869	1,594353437	2,10765624	2,695949316	3,102765799	3,44118762	3,94969368	4,244197845
40GB StrideRead	99,9992218	99,9981308	99,99887848	99,99867249	99,99880219	99,9980545	99,99906921	99,99810791	99,99906921	99,99809265	99,9981308	99,99806213	99,99862671
40GB StrideRead	99,99861908	99,99724579	99,99685669	99,99765015	99,99903107	99,99526215	99,9963913	99,99723053	99,99819183	99,99716187	99,99716949	99,99713898	99,99521637

Dos três sistemas de ficheiros testados este parece ser o que obtêm a pior performance, em especial para as escritas que é onde a discrepância de resultados entre este e os anteriores se nota mais acentuada. Notam-se que, de um modo geral, os KB/s obtidos são menores do que os ibtidos nos testes anteriores. Da mesma forma que com os outros sistemas de ficheiros, nota-se um impacto na performance com o uso de records de tamanho 256KB ou mais, entende-se que isto ocorre pelas razões já mencionadas anteriormente e, o facto de isto acontecer nos três sistemas de ficheiros, não surpreende, pois embora os sistemas de ficheiros sejam diferentes, a máquina é a mesma e, por consequência, a cache que têm ao seu dispor é também a mesma e terá as mesmas limitações.

TESTE 2:

Testa-se agora o comportamento multi-threaded/multi-processo, mas tendo em conta o grande número de testes a realizar para este comportamento apenas se usa um tamanho de record (4KB) e de ficheiro (20GB) para teste para cada thread/processo.

Correndo agora os comandos:

/opt/csw/bin/iozone -i0 -R -b teste21NFS20.xls -s 20g -l2 -u8 -F /discoHitachi/iozone1.tmp /discoHitachi/iozone2.tmp /discoHitachi/iozone3.tmp /discoHitachi/iozone4.tmp /discoHitachi/iozone5.tmp /discoHitachi/iozone6.tmp /discoHitachi/iozone7.tmp /discoHitachi/iozone8.tmp

/opt/csw/bin/iozone -i0 -i1 -R -b teste22NFS20.xls -s 20g -l2 -u8 -F /discoHitachi/iozone1.tmp /discoHitachi/iozone2.tmp /discoHitachi/iozone3.tmp /discoHitachi/iozone4.tmp /discoHitachi/iozone5.tmp /discoHitachi/iozone6.tmp /discoHitachi/iozone7.tmp /discoHitachi/iozone8.tmp

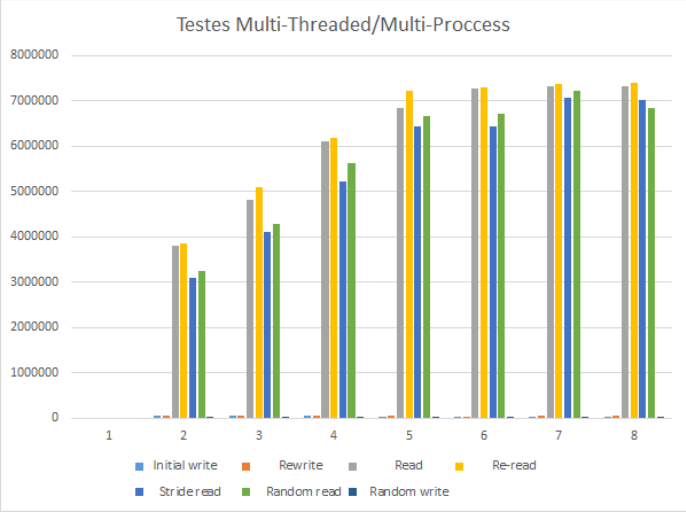
/opt/csw/bin/iozone -i0 -i2 -R -b teste231NFS20.xls -s 20g -l2 -u8 -F /discoHitachi/iozone1.tmp /discoHitachi/iozone2.tmp /discoHitachi/iozone3.tmp /discoHitachi/iozone4.tmp /discoHitachi/iozone5.tmp /discoHitachi/iozone6.tmp /discoHitachi/iozone7.tmp /discoHitachi/iozone8.tmp

/opt/csw/bin/iozone -i0 -i5 -R -b teste233NFS20.xls -s 20g -l2 -u8 -F /discoHitachi/iozone1.tmp

/discoHitachi/iozone2.tmp /discoHitachi/iozone3.tmp
/discoHitachi/iozone4.tmp /discoHitachi/iozone5.tmp
/discoHitachi/iozone6.tmp /discoHitachi/iozone7.tmp
/discoHitachi/iozone8.tmp

foi possível obter o seguinte gráfico:

(note-se que neste gráfico o eixo y representa KB/s e o eixo x o número de threads/processos)



Observamos agora que, embora os testes single-threaded/single-proccess se tenham revelado os testes com a pior performance entre os três sistemas de ficheiros, estes testes multi-threaded/multi-proccess revelam uma melhor performance que o sistema de ficheiros ZFS, mas pior que o sistema de ficheiros UFS. Novamente, assim como para o sistema de ficheiros UFS, esta melhoria na performance é só para as operações de leitura, pois as operações de escrita não demonstram uma performance particularmente significante, praticamente não escalando com o número de threads/processos. Para as operações de leitura, a nível da escalabilidade com o número de threads/processos, vemos que escala melhor que o sistema ZFS, pois não se nota uma descida na performance em nenhum momento, no entanto a escalabilidade parece inferior do que com o sistema de ficheiros UFS, pois vemos que a curva de speed-up neste gráfico parece estagnar por volta das 5 threads/processos levando a crer que o pico de escalabilidade se encontra bastante próximo.

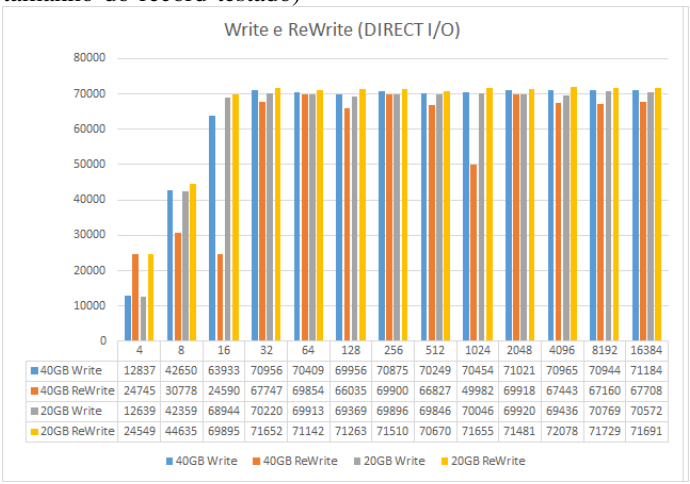
TESTE 3:

Usando os comandos:
/opt/csw/bin/iozone -f /discoHitachi/iozone.tmp -+u -a -i0 -R -b teste3NFS40.xls -s 40g -I
/opt/csw/bin/iozone -f /discoHitachi/iozone.tmp -+u -a -i0 -i1 -R -b teste32NFS40.xls -s 40g -I
/opt/csw/bin/iozone -f /discoHitachi/iozone.tmp -+u -a -i0 -i2 -R -b teste331NFS40.xls -s 40g -I
/opt/csw/bin/iozone -f /discoHitachi/iozone.tmp -+u -a -i0 -i5 -R -b teste333NFS40.xls -s 40g -I

/opt/csw/bin/iozone -f /discoHitachi/iozone.tmp -+u -a -i0 -R -b teste3NFS20.xls -s 20g -I
/opt/csw/bin/iozone -f /discoHitachi/iozone.tmp -+u -a -i0 -i1 -R -b teste32NFS20.xls -s 20g -I
/opt/csw/bin/iozone -f /discoHitachi/iozone.tmp -+u -a -i0 -i2 -R -b teste331NFS20.xls -s 20g -I
/opt/csw/bin/iozone -f /discoHitachi/iozone.tmp -+u -a -i0 -i5 -R -b teste333NFS20.xls -s 20g -I

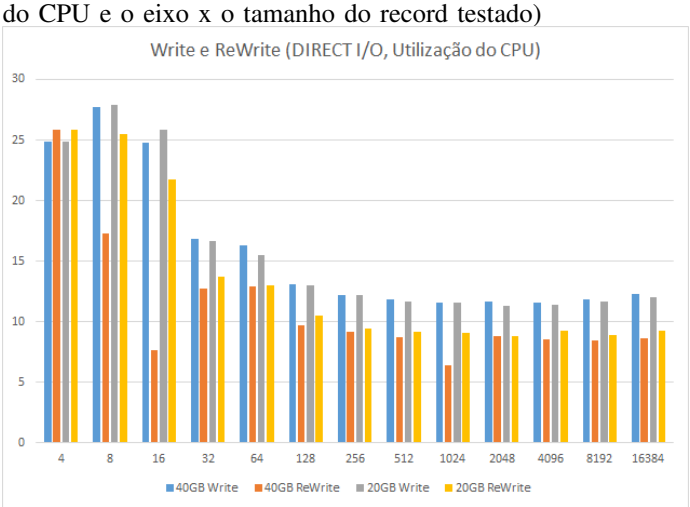
obtiveram-se os seguintes gráficos:

(Neste gráfico o eixo y representa KB/s e o eixo x o tamanho do record testado)

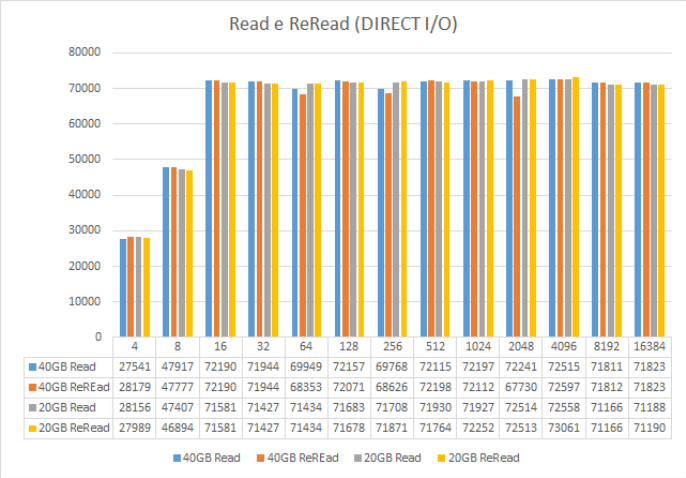


	4	8	16	32	64	128	256	512	1024	2048	4096	8192	16384
40GB Write	12837	42650	63933	70956	70409	69956	70875	70249	70454	71021	70965	70944	71184
40GB ReWrite	24745	30778	24590	67747	69854	66035	69900	66827	49982	69918	67443	67160	67708
20GB Write	12639	42359	68944	70220	69913	69369	69896	69846	70046	69920	69436	70769	70572
20GB ReWrite	24549	44635	69895	71652	71142	71263	71510	70670	71655	71481	72078	71729	71691

(Neste gráfico o eixo y representa a percentagem de utilização do CPU e o eixo x o tamanho do record testado)

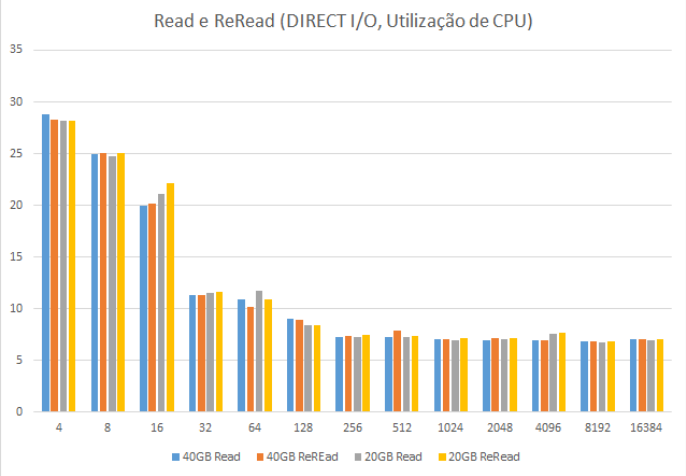


(Neste gráfico o eixo y representa KB/s e o eixo x o tamanho do record testado)

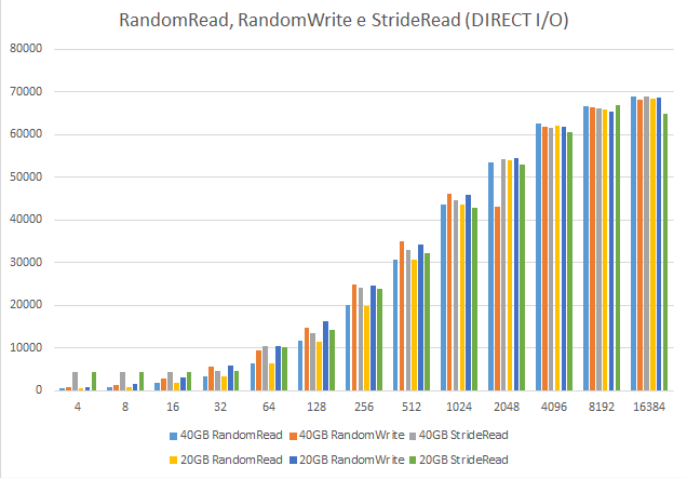


	4	8	16	32	64	128	256	512	1024	2048	4096	8192	16384
40GB Read	27541	47917	72190	71944	69949	72157	69768	72115	72197	72241	72515	71811	71823
40GB ReREad	28179	47777	72190	71944	68353	72071	68626	72198	72112	67730	72597	71812	71823
20GB Read	28156	47407	71581	71427	71434	71683	71708	71930	71927	72514	72558	71166	71188
20GB ReRead	27989	46894	71581	71427	71434	71678	71871	71764	72252	72513	73061	71166	71190

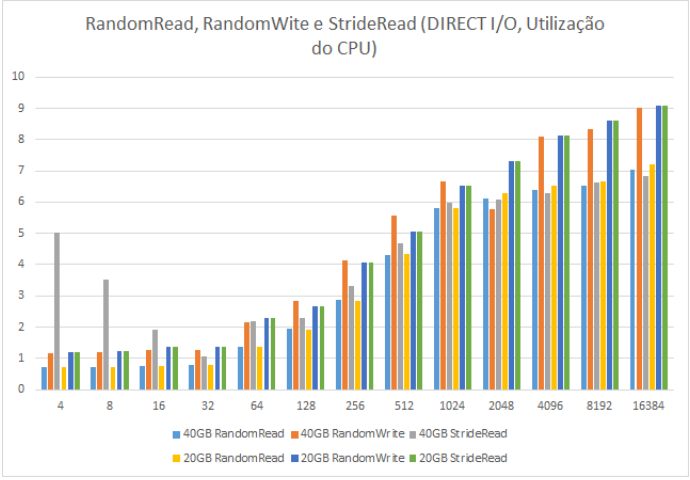
(Neste gráfico o eixo y representa a percentagem de utilização do CPU e o eixo x o tamanho do record testado)



(Neste gráfico o eixo y representa KB/s e o eixo x o tamanho do record testado)



(Neste gráfico o eixo y representa a percentagem de utilização do CPU e o eixo x o tamanho do record testado)



Nestes gráficos vemos uma performance pior do que sem acessos ao disco, como já se esperava pelas razões mencionadas para os mesmos testes feitos para o sistema de ficheiros UFS. A performance das escritas é a que se encontra mais próxima com a performance obtida para estes testes no sistema UFS, mas a das leituras é bastante inferior. Vê-se também que a performance escala com o tamanho dos records, como era de esperar e já se tinha visto no sistema UFS.

E. Análise dos Resultados com Benchmark Passivo - Conclusão

Por fim, depois de um minucioso estudo da ferramenta IOzone nos três sistemas de ficheiros indicados podemos tirar algumas conclusões. Vemos que, em ambiente single-threaded/multi-threaded o sistema ZFS é sem dúvida o que obtém a melhor performance e portanto deve ser preferido quando usado nestas condições na presente máquina. Em ambiente multi-threaded/multi-proccess a análise torna-se mais complexa. Para operações de leitura vemos que o sistema UFS é o que obtém a melhor performance e escalabilidade a nível de threads/processos de longe, no entanto, para operações de escrita nota-se que o sistema ZFS é o que apresenta a melhor escalabilidade (mesmo que seja apenas até às 5 threads/processos) e a melhor performance. Nestas condições, caso se queira fazer operações de leitura, deve-se preferir o sistema UFS, mas para operações de escrita é preferível o sistema ZFS.

IV. Medições de Desempenho - Benchmark Ativo

A. Estudo do Ambiente com Truss

De forma a usar DTrace para replicar as medições realizadas por IOzone utilizou-se a ferramenta **truss**. Esta ferramenta permite saber em tempo real as chamadas de sistema que uma determinada aplicação invoca. Ao executarmos o seguinte comando: **truss -o outputTruss.txt -df /opt/csw/bin/iozone -+u -i0 -i1 -i2 -i5 -s200**, é-nos possível interpretar o funcionamento da ferramenta IOzone de forma a replicá-la em benchmark ativo.

Segue-se um excerto do output de truss com o comando anterior que nos demonstra os testes realizados para -i0:

```
24055:   1.1665 open("iozone.tmp", O_WRONLY|O_CREAT|O_TRUNC, 0640) = 3
24055:   1.1666 close(3)                                           = 0
24055:   1.1667 open("iozone.tmp", O_RDWR)                         = 3
24055:   1.1849 fdsync(3, FSYNC)                                   = 0
24055:   1.1850 write(3, " y y y y y y y y\0\0\0\0".., 4096)       = 4096
24055:   1.1851 write(3, " y y y y y y y y\0\0\0\0".., 4096)       = 4096
24055:   1.1852 write(3, " y y y y y y y y\0\0\0\0".., 4096)       = 4096
24055:   1.1853 write(3, " y y y y y y y y\0\0\0\0".., 4096)       = 4096
24055:   1.1854 write(3, " y y y y y y y y\0\0\0\0".., 4096)       = 4096
24055:   1.1855 write(3, " y y y y y y y y\0\0\0\0".., 4096)       = 4096
24055:   1.1856 write(3, " y y y y y y y y\0\0\0\0".., 4096)       = 4096
24055:   1.1857 write(3, " y y y y y y y y\0\0\0\0".., 4096)       = 4096
24055:   1.1858 write(3, " y y y y y y y y\0\0\0\0".., 4096)       = 4096
24055:   1.1858 write(3, " y y y y y y y y\0\0\0\0".., 4096)       = 4096
24055:   1.1859 write(3, " y y y y y y y y\0\0\0\0".., 4096)       = 4096
24055:   1.1860 write(3, " y y y y y y y y\0\0\0\0".., 4096)       = 4096
24055:   1.1861 write(3, " y y y y y y y y\0\0\0\0".., 4096)       = 4096
24055:   1.1862 write(3, " y y y y y y y y\0\0\0\0".., 4096)       = 4096
24055:   1.1863 write(3, " y y y y y y y y\0\0\0\0".., 4096)       = 4096
24055:   1.1864 write(3, " y y y y y y y y\0\0\0\0".., 4096)       = 4096
24055:   1.1865 write(3, " y y y y y y y y\0\0\0\0".., 4096)       = 4096
24055:   1.1866 write(3, " y y y y y y y y\0\0\0\0".., 4096)       = 4096
24055:   1.1867 write(3, " y y y y y y y y\0\0\0\0".., 4096)       = 4096
24055:   1.1868 write(3, " y y y y y y y y\0\0\0\0".., 4096)       = 4096
24055:   1.1869 write(3, " y y y y y y y y\0\0\0\0".., 4096)       = 4096
24055:   1.1870 write(3, " y y y y y y y y\0\0\0\0".., 4096)       = 4096
24055:   1.1871 write(3, " y y y y y y y y\0\0\0\0".., 4096)       = 4096
24055:   1.1872 write(3, " y y y y y y y y\0\0\0\0".., 4096)       = 4096
24055:   1.1873 write(3, " y y y y y y y y\0\0\0\0".., 4096)       = 4096
24055:   1.1874 write(3, " y y y y y y y y\0\0\0\0".., 4096)       = 4096
24055:   1.1876 write(3, " y y y y y y y y\0\0\0\0".., 4096)       = 4096
24055:   1.1877 write(3, " y y y y y y y y\0\0\0\0".., 4096)       = 4096
24055:   1.1878 write(3, " y y y y y y y y\0\0\0\0".., 4096)       = 4096
24055:   1.1879 write(3, " y y y y y y y y\0\0\0\0".., 4096)       = 4096
24055:   1.1880 write(3, " y y y y y y y y\0\0\0\0".., 4096)       = 4096
24055:   1.1881 write(3, " y y y y y y y y\0\0\0\0".., 4096)       = 4096
24055:   1.1882 write(3, " y y y y y y y y\0\0\0\0".., 4096)       = 4096
24055:   1.1883 write(3, " y y y y y y y y\0\0\0\0".., 4096)       = 4096
24055:   1.1884 write(3, " y y y y y y y y\0\0\0\0".., 4096)       = 4096
24055:   1.1884 write(3, " y y y y y y y y\0\0\0\0".., 4096)       = 4096
24055:   1.1885 write(3, " y y y y y y y y\0\0\0\0".., 4096)       = 4096
24055:   1.1886 write(3, " y y y y y y y y\0\0\0\0".., 4096)       = 4096
24055:   1.1887 write(3, " y y y y y y y y\0\0\0\0".., 4096)       = 4096
24055:   1.1888 write(3, " y y y y y y y y\0\0\0\0".., 4096)       = 4096
24055:   1.1888 write(3, " y y y y y y y y\0\0\0\0".., 4096)       = 4096
24055:   1.1889 write(3, " y y y y y y y y\0\0\0\0".., 4096)       = 4096
24055:   1.1890 write(3, " y y y y y y y y\0\0\0\0".., 4096)       = 4096
24055:   1.1891 write(3, " y y y y y y y y\0\0\0\0".., 4096)       = 4096
24055:   1.1892 write(3, " y y y y y y y y\0\0\0\0".., 4096)       = 4096
24055:   1.1892 write(3, " y y y y y y y y\0\0\0\0".., 4096)       = 4096
24055:   1.1893 write(3, " y y y y y y y y\0\0\0\0".., 4096)       = 4096
24055:   1.1894 write(3, " y y y y y y y y\0\0\0\0".., 4096)       = 4096
24055:   1.1895 write(3, " y y y y y y y y\0\0\0\0".., 4096)       = 4096
24055:   1.1896 write(3, " y y y y y y y y\0\0\0\0".., 4096)       = 4096
24055:   1.2034 fdsync(3, FSYNC)                                   = 0
24055:   1.2034 close(3)                                           = 0
24055:   1.2035 getrusage(0xFFFF80FFBFFFF7D0)                      = 0
24055:   1.2035 getrusage(0xFFFF80FFBFFFF7D0)                      = 0
```

Por observação vemos que, no ponto circundado 1, para os testes -i0 (write e rewrite) é aberto um ficheiro de nome **iozone.tmp** com permissões para leitura e escrita. De seguida, no ponto circundado 2, vemos já vários rewrites consecutivos a serem feitos no mesmo ficheiro. Observamos que se trata do mesmo ficheiro pois é usado o descritor devolvido pela chamada de sistema open no ponto circundado 1 (valor 3), vemos que é escrito um record de 4KB em cada write. No ponto circundado 3 vemos o ficheiro a ser fechado depois de realizados todos os testes para i0. Por fim, no ponto circundado 4, vê-se ainda que a chamada de sistema textbfgetrusage() é utilizada para medir os valores de utilização do CPU.

Veja-se agora um excerto do output relativo à leitura do ficheiro para os testes em -i1 (read e reread):

```
24055:   1.2179 getrusage(0xFFFF80FFBFFFF800)                      = 0
24055:   1.2179 open("iozone.tmp", O_RDONLY)                       = 3
24055:   1.2180 fdsync(3, FSYNC)                                   = 0
24055:   1.2181 read(3, " y y y y y y y y\0\0\0\0".., 4096)       = 4096
24055:   1.2182 lseek(3, 0, SEEK_SET)                              = 0
24055:   1.2182 read(3, " y y y y y y y y\0\0\0\0".., 4096)       = 4096
24055:   1.2183 read(3, " y y y y y y y y\0\0\0\0".., 4096)       = 4096
24055:   1.2184 read(3, " y y y y y y y y\0\0\0\0".., 4096)       = 4096
24055:   1.2184 read(3, " y y y y y y y y\0\0\0\0".., 4096)       = 4096
24055:   1.2185 read(3, " y y y y y y y y\0\0\0\0".., 4096)       = 4096
24055:   1.2186 read(3, " y y y y y y y y\0\0\0\0".., 4096)       = 4096
24055:   1.2186 read(3, " y y y y y y y y\0\0\0\0".., 4096)       = 4096
24055:   1.2187 read(3, " y y y y y y y y\0\0\0\0".., 4096)       = 4096
24055:   1.2188 read(3, " y y y y y y y y\0\0\0\0".., 4096)       = 4096
24055:   1.2188 read(3, " y y y y y y y y\0\0\0\0".., 4096)       = 4096
24055:   1.2189 read(3, " y y y y y y y y\0\0\0\0".., 4096)       = 4096
24055:   1.2190 read(3, " y y y y y y y y\0\0\0\0".., 4096)       = 4096
24055:   1.2190 read(3, " y y y y y y y y\0\0\0\0".., 4096)       = 4096
24055:   1.2191 read(3, " y y y y y y y y\0\0\0\0".., 4096)       = 4096
24055:   1.2192 read(3, " y y y y y y y y\0\0\0\0".., 4096)       = 4096
24055:   1.2193 read(3, " y y y y y y y y\0\0\0\0".., 4096)       = 4096
24055:   1.2193 read(3, " y y y y y y y y\0\0\0\0".., 4096)       = 4096
24055:   1.2194 read(3, " y y y y y y y y\0\0\0\0".., 4096)       = 4096
24055:   1.2195 read(3, " y y y y y y y y\0\0\0\0".., 4096)       = 4096
24055:   1.2195 read(3, " y y y y y y y y\0\0\0\0".., 4096)       = 4096
24055:   1.2196 read(3, " y y y y y y y y\0\0\0\0".., 4096)       = 4096
24055:   1.2197 read(3, " y y y y y y y y\0\0\0\0".., 4096)       = 4096
24055:   1.2197 read(3, " y y y y y y y y\0\0\0\0".., 4096)       = 4096
24055:   1.2198 read(3, " y y y y y y y y\0\0\0\0".., 4096)       = 4096
24055:   1.2199 read(3, " y y y y y y y y\0\0\0\0".., 4096)       = 4096
24055:   1.2199 read(3, " y y y y y y y y\0\0\0\0".., 4096)       = 4096
24055:   1.2200 read(3, " y y y y y y y y\0\0\0\0".., 4096)       = 4096
24055:   1.2201 read(3, " y y y y y y y y\0\0\0\0".., 4096)       = 4096
24055:   1.2201 read(3, " y y y y y y y y\0\0\0\0".., 4096)       = 4096
24055:   1.2202 read(3, " y y y y y y y y\0\0\0\0".., 4096)       = 4096
24055:   1.2203 read(3, " y y y y y y y y\0\0\0\0".., 4096)       = 4096
24055:   1.2203 read(3, " y y y y y y y y\0\0\0\0".., 4096)       = 4096
24055:   1.2204 read(3, " y y y y y y y y\0\0\0\0".., 4096)       = 4096
24055:   1.2205 read(3, " y y y y y y y y\0\0\0\0".., 4096)       = 4096
24055:   1.2206 read(3, " y y y y y y y y\0\0\0\0".., 4096)       = 4096
24055:   1.2206 read(3, " y y y y y y y y\0\0\0\0".., 4096)       = 4096
24055:   1.2207 read(3, " y y y y y y y y\0\0\0\0".., 4096)       = 4096
24055:   1.2208 read(3, " y y y y y y y y\0\0\0\0".., 4096)       = 4096
24055:   1.2208 read(3, " y y y y y y y y\0\0\0\0".., 4096)       = 4096
24055:   1.2209 read(3, " y y y y y y y y\0\0\0\0".., 4096)       = 4096
24055:   1.2210 read(3, " y y y y y y y y\0\0\0\0".., 4096)       = 4096
24055:   1.2210 read(3, " y y y y y y y y\0\0\0\0".., 4096)       = 4096
24055:   1.2211 read(3, " y y y y y y y y\0\0\0\0".., 4096)       = 4096
24055:   1.2212 read(3, " y y y y y y y y\0\0\0\0".., 4096)       = 4096
24055:   1.2212 read(3, " y y y y y y y y\0\0\0\0".., 4096)       = 4096
24055:   1.2213 read(3, " y y y y y y y y\0\0\0\0".., 4096)       = 4096
24055:   1.2214 read(3, " y y y y y y y y\0\0\0\0".., 4096)       = 4096
24055:   1.2214 read(3, " y y y y y y y y\0\0\0\0".., 4096)       = 4096
24055:   1.2215 read(3, " y y y y y y y y\0\0\0\0".., 4096)       = 4096
24055:   1.2216 read(3, " y y y y y y y y\0\0\0\0".., 4096)       = 4096
24055:   1.2217 fdsync(3, FSYNC)                                   = 0
24055:   1.2217 close(3)                                           = 0
24055:   1.2218 getrusage(0xFFFF80FFBFFFF800)                      = 0
24055:   1.2218 getrusage(0xFFFF80FFBFFFF800)                      = 0
```

Vemos no ponto circundado 1 o mesmo ficheiro a ser aberto e, no ponto circundado 2, consecutivas leituras. No ponto 3 o ficheiro é novamente fechado e no 4 vemos a chamada de sistema textbfgetrusage() que é utilizada para medir os valores de utilização do CPU.

Observamos agora uma terceira e ultima imagem:

```
24055:   1.2389 lseek(3, 0x0001D000, SEEK_SET)                     = 118784
24055:   1.2389 write(3, " y y y y y y y y\0\0\0\0".., 4096)       = 4096
24055:   1.2390 lseek(3, 45056, SEEK_SET)                          = 45056
24055:   1.2391 write(3, " y y y y y y y y\0\0\0\0".., 4096)       = 4096
24055:   1.2391 lseek(3, 0x0002C000, SEEK_SET)                     = 180224
24055:   1.2392 write(3, " y y y y y y y y\0\0\0\0".., 4096)       = 4096
24055:   1.2473 fdsync(3, FSYNC)                                   = 0
24055:   1.2473 close(3)                                           = 0
24055:   1.2474 getrusage(0xFFFF80FFBFFFF7E0)                      = 0
24055:   1.2475 write(1, "         3 1 9 0 3", 9)                  = 9
24055:   1.2476 write(1, "         3 0 3 5 8", 9)                  = 9
24055:   1.2477 getrusage(0xFFFF80FFBFFFF810)                      = 0
24055:   1.2478 open("iozone.tmp", O_RDONLY)                       = 3
24055:   1.2478 fdsync(3, FSYNC)                                   = 0
24055:   1.2479 stat("iozone.tmp", 0xFFFF80FFBFFFF810)             = 0
24055:   1.2480 lseek(3, 0xFFFFFFFFFFFFF000, SEEK_END)             = 200704
24055:   1.2481 read(3, " y y y y y y y y\0\0\0\0".., 4096)       = 4096
24055:   1.2481 lseek(3, 0xFFFFFFFFFFFFE000, SEEK_CUR)             = 196608
24055:   1.2482 read(3, " y y y y y y y y\0\0\0\0".., 4096)       = 4096
24055:   1.2482 lseek(3, 0xFFFFFFFFFFFFE000, SEEK_CUR)             = 192512
24055:   1.2483 read(3, " y y y y y y y y\0\0\0\0".., 4096)       = 4096
24055:   1.2484 lseek(3, 0xFFFFFFFFFFFFE000, SEEK_CUR)             = 188416
```

Por aqui percebemos que é usada a chamada ao sistema **lseek()** com o argumento **SEEK_SET** para serem realizados os testes de random write/read. Mais em baixo vemos o uso de **lseek()** com o argumento **SEEK_CURL** para se realizar os testes de stride read. Pode-se provar esta interpretação do objetivo do uso de **lseek** ao olhar para esta tabela tirada da documentação da chamada ao sistema [6]:

Value	Meaning
SEEK_SET	Offset is to be measured in absolute terms.
SEEK_CUR	Offset is to be measured relative to the current location of the pointer.
SEEK_END	Offset is to be measured relative to the end of the file.

Aqui vemos que devemos usar **SEEK_SET** quando a próxima posição a procurar no ficheiro não depende da posição atual já que é relativo a todo o ficheiro, para além disso vemos na imagem anterior que quando **lseek** usa **SEEK_SET** como argumento tem um offset sempre diferente (aleatório) como segundo argumento. Nota-se também que devemos usar **SEEK_CURL** quando a próxima posição depende da posição atual já que é relativo ao local a que se aponta agora, para além disso na imagem anterior, quando **SEEK_CURL** é usado, vemos um offset sempre igual (stride) a ser fornecido como segundo argumento.

B. Desenvolvimento do script DTrace

Podemos agora já ter uma ideia sobre como instrumentar corretamente a ferramenta IOzone em tempo real com DTrace de forma a validar, com benchmark ativo, os valores obtidos com IOzone. O nosso objetivo deverá ser o de capturar, com um script DTrace, os probes relativos à chamada de sistema open() para o nome de ficheiro que vemos no output das imagens anteriores e depois as chamadas close() correspondentes de forma a imprimir informação sobre um determinado teste que foi encontrada entretanto. Serão também capturados os probes lançados por write() e read() correspondentes de forma a calcular o total de bytes escritos/lidos e o tempo despendido nestas operações. No final poderemos usar estes valores de bytes e o valor do tempo de forma a saber os bytes escritos/lidos por segundo tanto para first write, para rewrites, para reads, rereads, random read, random write e stride read. Mostra-se agora o script DTrace resultante deste raciocínio:

```
#!/usr/bin/dtrace -s
#pragma D option quiet

dtrace:::BEGIN {
        printf("\n");
        printf("\tInstrumentador para IOzone\n");
        printf("\t-------------------------");

        self->ok         = 0;
        self->teste      = 1;
        self->kb_read    = 0;
        self->kb_write   = 0;
        self->time_read  = 0;
        self->time_write = 0;
        self->time_close = 0;

        self->read  = 0;
        self->write = 0;

        total_kb   = 0;
        total_time = 0;
        self->time_init  = 0;
}

syscall::openat*:entry {
        self->fd = arg1;
}

syscall::openat*:return
/self->fd/ {
        self->ok = (strstr(copyinstr(self->fd), $$1) != NULL) ? 1 : 0;
}

syscall::read*:entry
/self->ok/ {
        self->kb_read   = self->kb_read  + (arg2/1024);

        self->time_init = timestamp;
}

syscall::read*:return
/self->ok && arg1/ {
        self->time_read = self->time_read + ((timestamp - self->time_init));
}

syscall::write*:entry
/self->ok/ {
        self->kb_write  = self->kb_write + (arg2/1024);

        self->time_init = timestamp;
}

syscall::write*:return
/self->ok && arg1/ {
        self->time_write = self->time_write + ((timestamp - self-
>time_init));
}

syscall::close*:return
/(self->kb_read>0 || self->kb_write>0) && self->ok/ {

        kb          = self->kb_write+self->kb_read;
    total_kb    = total_kb + kb;
        time        = self->time_read + self->time_write;
        total_time = total_time + time;
        kbr         = self->kb_read;
        kbw         = self->kb_write;

        printf("\n");
        printf("
----------------------------------------------------------------------
\n");
        printf(" /      Test File Closed     \n");
        printf("|       ----------------     \n");
    printf("| %s\n", $$2);
        printf("|%5s|%15s|%15s|%15s|%16s|%17s|%20s|\n", "Test", "KB", "KB
Read", "KB Write", "Time Spent", "Time Read", "Time Write");
        printf("|%5d|%15d|%15d|%15d|%16d|%17d|%20d|\n", self->teste, kb,
kbr, kbw, time, self->time_read, self->time_write);
        printf("|\n");
        printf(" \\\n");
        printf("
----------------------------------------------------------------------
\n");

        self->ok         = 0;
        self->teste      = self->teste +1;
        self->time_init  = 0;
        self->kb_read    = 0;
        self->kb_write   = 0;
    self->fd         = 0;
        self->time_read  = 0;
        self->time_write = 0;
}

dtrace:::END {

        printf(" ---------------------------------------\n");
        printf("| DTrace stoped.                        |\n");
        printf(" ---------------------------------------\n");
        printf("| Total Time spent: %20d|\n", total_time);
        printf("| Total KB:         %20d|\n", total_kb);
        printf(" ---------------------------------------\n\n");
}
```

Note-se como, no ponto circundado 1, se usa o ficheiro com

o nome argumento ($$1) como o ficheiro a observar (onde se espera que os testes do iozone sejam realizados). O facto de fornecermos este argumento permite-nos usar este script também para instrumentar os testes multi-threaded/multi-process e não só os single-threaded/single-process. O segundo argumento ($$2) será uma indicação por parte do utilizador sobre o tipo de teste que está a realizar, por exemplo: "multi-threded/process Thread/processo: 1". E aparecerá apresentado junto com os valores imprimidos.
Nos pontos circundados 2 e 3 podemos já notar como quando é capturado um probe entry de uma chamada ao sistema read ou write se regista os KB envolvidos na chamada e se começa a contar o tempo que irá demorar. Nos pontos circundados 4 e 5 vemos a serem capturados os probes dos respetivos return das chamadas anteriores e a terminar a contagem do tempo demorado (em nanosegundos). Desta forma conseguimos obter para qualquer tipo de teste os KB envolvidos no teste e o tempo que passou.
No ponto circundado 6 vemos que os valores que foram encontrados são imprimidos no ecrã quando é capturado um probe da chamada ao sistema close. O facto de ser encontrada esta chamada implica que se acabou um teste (um initial write ou um rewrite, etc.), os valores não globais são então reinicializados a zero para agora guardarem informação relativa ao teste seguinte.
Por fim o ponto circundado 7 demonstra a impressão dos resultados globais que foram sendo guardados ao longo de todos os testes. Esta situação vai acontecer quando o script DTrace for parado, por exemplo com o uso de **ĉ**. O script deve ser parado quando a ferramenta IOzone terminar.

```
a72293@solarisEdu:~/a72293$ dtrace -qs d1 a72293iozone.tmp single-threaded

      Instrumentador para IOzone
      --------------------------
 ------------------------------------------------------------------------------------------
/      Test File Closed
|      ---------------
| single-threaded
| Test|            KB|       KB Read|      KB Write|      Time Spent|       Time Read|         Time Write|
|    0|           200|             0|           200|         1373166|               0|            1373166|
|
 \
  ------------------------------------------------------------------------------------------

 ------------------------------------------------------------------------------------------
/      Test File Closed
|      ---------------
| single-threaded
| Test|            KB|       KB Read|      KB Write|      Time Spent|       Time Read|         Time Write|
|    1|           200|             0|           200|          733610|               0|             733610|
|
 \
  ------------------------------------------------------------------------------------------

 ------------------------------------------------------------------------------------------
/      Test File Closed
|      ---------------
| single-threaded
| Test|            KB|       KB Read|      KB Write|      Time Spent|       Time Read|         Time Write|
|    2|           204|           204|             0|          332241|          332241|                  0|
|
 \
  ------------------------------------------------------------------------------------------

 ------------------------------------------------------------------------------------------
/      Test File Closed
|      ---------------
| single-threaded
| Test|            KB|       KB Read|      KB Write|      Time Spent|       Time Read|         Time Write|
|    3|           204|           204|             0|          313269|          313269|                  0|
|
 \
  ------------------------------------------------------------------------------------------

 ------------------------------------------------------------------------------------------
/      Test File Closed
|      ---------------
| single-threaded
| Test|            KB|       KB Read|      KB Write|      Time Spent|       Time Read|         Time Write|
|    4|           200|           200|             0|          325743|          325743|                  0|
|
 \
  ------------------------------------------------------------------------------------------

 ------------------------------------------------------------------------------------------
/      Test File Closed
|      ---------------
| single-threaded
| Test|            KB|       KB Read|      KB Write|      Time Spent|       Time Read|         Time Write|
|    5|           200|             0|           200|          737491|               0|             737491|
|
 \
  ------------------------------------------------------------------------------------------

 ------------------------------------------------------------------------------------------
/      Test File Closed
|      ---------------
| single-threaded
| Test|            KB|       KB Read|      KB Write|      Time Spent|       Time Read|         Time Write|
|    6|           200|           200|             0|          322978|          322978|                  0|
|
 \
  ------------------------------------------------------------------------------------------
^C
 ------------------------------------
| DTrace stoped.                     |
 ------------------------------------
| Total Time spent:          4138498|
| Total KB:                     1408|
 ------------------------------------

a72293@solarisEdu:~/a72293$
```

e o output da ferramenta IOzone:

```
a72293@solarisEdu:~/a72293$ /opt/csw/bin/iozone -fa72293iozone.tmp -i0 -i1 -i2 -
i5 -s200 -r4
        Iozone: Performance Test of File I/O
                Version $Revision: 3.434 $
                Compiled for 64 bit mode.
                Build: Solaris10

        Contributors:William Norcott, Don Capps, Isom Crawford, Kirby Collins
                     Al Slater, Scott Rhine, Mike Wisner, Ken Goss
                     Steve Landherr, Brad Smith, Mark Kelly, Dr. Alain CYR,
                     Randy Dunlap, Mark Montague, Dan Million, Gavin Brebner,
                     Jean-Marc Zucconi, Jeff Blomberg, Benny Halevy, Dave Boone,
                     Erik Habbinga, Kris Strecker, Walter Wong, Joshua Root,
                     Fabrice Bacchella, Zhenghua Xue, Qin Li, Darren Sawyer,
                     Vangel Bojaxhi, Ben England, Vikentsi Lapa,
                     Alexey Skidanov.

        Run began: Mon May  8 04:32:59 2017

        File size set to 200 kB
        Record Size 4 kB
        Command line used: /opt/csw/bin/iozone -fa72293iozone.tmp -i0 -i1 -i2 -i
5 -s200 -r4
        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.
                                                                    random    random
   bkwd    record    stride
             kB  reclen    write  rewrite    read    reread    read     write
   read   rewrite      read   fwrite frewrite    fread  freread
            200       4   130201   230722   450563   454468   408044   221463
                      399304

iozone test complete.
a72293@solarisEdu:~/a72293$
```

C. Testes ao script DTrace

Antes de usar o cript desenvolvido e organizar os seus resultados em gráficos, realizaram-se dois testes para garantir que se encontra a funcionar corretamente.

Primeiro quis-se testar em ambiente single-threaded/single-process, para isso correu-se o script com o seguinte comando: **dtrace -qs instrumentador.d a72293iozone.tmp single-threaded**
noutro terminal, correu-se o seguinte comando em paralelo: **/opt/csw/bin/iozone -fa72293iozone.tmp -i0 -i1 -i2 -i5 -s200 -r4**

Observe-se agora o output do script desenvolvido:

Notam-se aqui algumas discrepâncias nos resultados. Nota-se, por exemplo, que o script DTrace obteve resultados maiores

de KB/s. Se olharmos para o output podemos, por exemplo, ver que o script obtém para o teste de Initial write (que é o teste 0 apresentado na imagem do output) os seguintes valores: 1373166 nanosegundos = 0.001373166 segundos, logo KB/s = 200/0.001373166 = 145648.81449 KB/s. Para o teste de ReWrite (que é o teste 1 apresentado na imagem do output) podemos ver os seguintes valores: 733610 nanosegundos = 0.00073361 segundos, logo KB/s = 200/0.00073361 = 272624.21897 KB/s. No entanto para a ferramenta IOzone foram obtidos os valores 130201 KB/s e 230722 KB/s para Initial Write e ReWrite respetivamente. O facto de haver esta diferença revela já diferenças entre o cálculo dos valores com benchmark passivo e com benchmark ativo. Uma possível razão poderá ser simplesmente a certeza dos cálculos dos tempos já que o valor de KB envolvidos nos testes são os mesmos com o script e com a ferramenta IOzone, logo a única variável que pode variar significantemente é o tempo. Vemos no início do output de IOzone que este tem uma resolução de 6 casas decimais no cálculo dos segundos, no entanto o nosso script calcula o tempo em nanosegundos por isso tem uma resolução de 9 casas decimais logo para valores temporais muito grandes estima-se que os resultados sejam ainda mais distintos, já que o erro inserido no cálculo dos tempos por parte da ferramenta IOzone se notará ainda mais.

De seguida quis-se testar em ambiente multi-threaded/multi-process, para isso correu-se o script com os seguintes comandos (cada um em paralelo):

(este comando vai observar a thread/processo 1)
dtrace -qs instrumentador.d a72293iozone1.tmp thread/proces_1

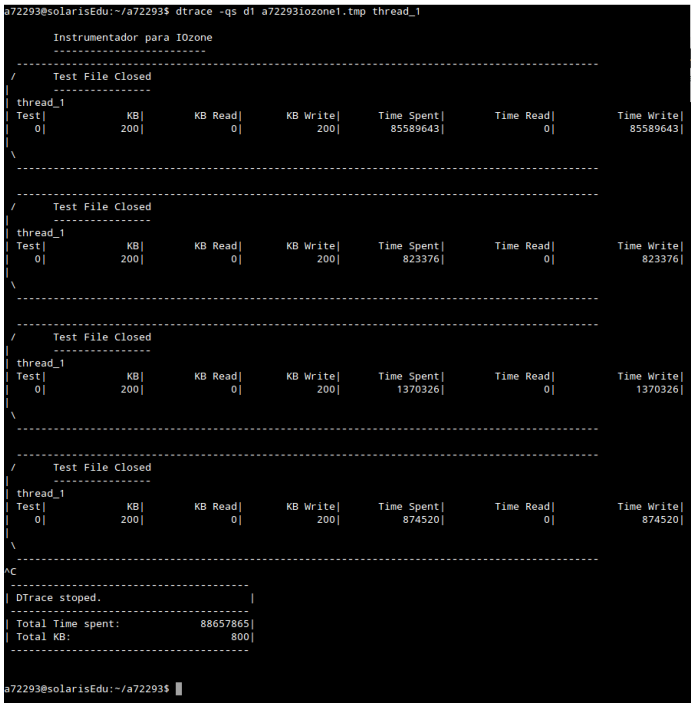
(este comando vai observar a thread/processo 2)
dtrace -qs instrumentador.d a72293iozone2.tmp thread/process_2

(este comando vai observar a thread/processo 3)
dtrace -qs instrumentador.d a72293iozone3.tmp thread/process_3

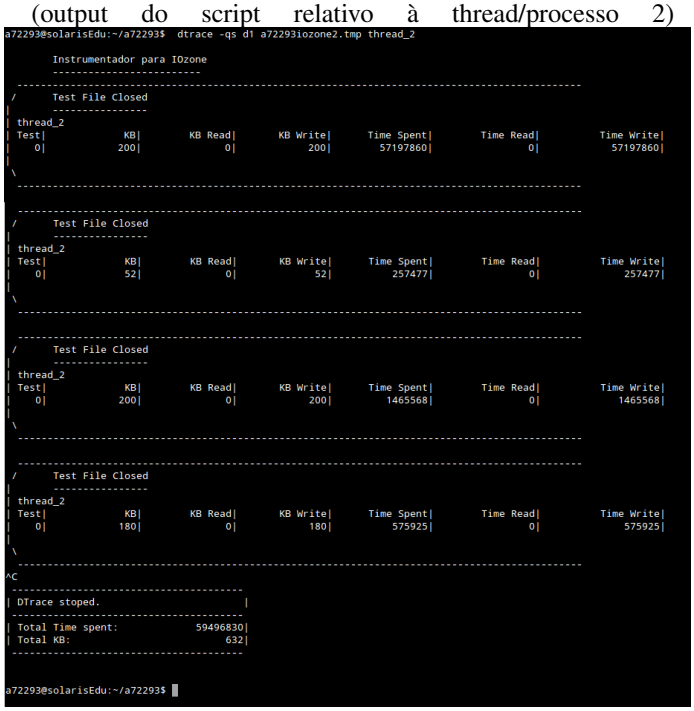
noutro terminal, correu-se o seguinte comando em paralelo:
/opt/csw/bin/iozone -i0 -s200 -r4 -l2 -u3 -F a72293iozone1.tmp a72293iozone2.tmp a72293iozone3.tmp

Observe-se agora o output do script desenvolvido:

(output do script relativo à thread/processo 1)



(output do script relativo à thread/processo 2)



(output do script relativo à thread/processo 3)

```
^Ca72293@solarisEdu:~/a72293$  dtrace -qs d1 a72293iozone3.tmp thread_3

       Instrumentador para IOzone
       --------------------------
 ------------------------------------------------------------------------------------------
 /      Test File Closed
        ---------------
 | thread_3
 | Test|           KB|        KB Read|       KB Write|      Time Spent|        Time Read|         Time Write|
 |    0|          200|              0|            200|        73429381|                0|           73429381|
 |
 \
  ------------------------------------------------------------------------------------------

  ------------------------------------------------------------------------------------------
 /      Test File Closed
 |      ---------------
 | thread_3
 | Test|           KB|        KB Read|       KB Write|      Time Spent|        Time Read|         Time Write|
 |    0|          148|              0|            148|         654701|                0|            654701|
 |
 \
  ------------------------------------------------------------------------------------------
^C
 ---------------------------------------
 | DTrace stoped.                      |
 ---------------------------------------
 | Total Time spent:          74084082|
 | Total KB:                        348|
 ---------------------------------------


a72293@solarisEdu:~/a72293$
```

e um excerto do output da ferramenta IOzone:

```
   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.
   Min process = 2
   Max process = 3
   Throughput test with 2 processes
   Each process writes a 200 kByte file in 4 kByte records

   Children see throughput for  2 initial writers  =    3488.03 kB/sec
   Parent sees throughput for  2 initial writers   =    1173.14 kB/sec
   Min throughput per process                      =    3488.03 kB/sec
   Max throughput per process                      =    3488.03 kB/sec
   Avg throughput per process                      =    1744.02 kB/sec
   Min xfer                                        =     200.00 kB

   Children see throughput for  2 rewriters        =  333403.55 kB/sec
   Parent sees throughput for  2 rewriters         =    8159.50 kB/sec
   Min throughput per process                      =  137536.42 kB/sec
   Max throughput per process                      =  195867.12 kB/sec
   Avg throughput per process                      =  166701.77 kB/sec
   Min xfer                                        =      48.00 kB


   Each process writes a 200 kByte file in 4 kByte records

   Children see throughput for  3 initial writers  =  228248.72 kB/sec
   Parent sees throughput for  3 initial writers   =    2090.13 kB/sec
   Min throughput per process                      =       0.00 kB/sec
   Max throughput per process                      =  128781.98 kB/sec
   Avg throughput per process                      =   76082.91 kB/sec
   Min xfer                                        =       0.00 kB

   Children see throughput for  3 rewriters        =  617299.25 kB/sec
   Parent sees throughput for  3 rewriters         =   11372.62 kB/sec
   Min throughput per process                      =  178649.66 kB/sec
   Max throughput per process                      =  251018.80 kB/sec
   Avg throughput per process                      =  205766.42 kB/sec
   Min xfer                                        =     144.00 kB
```

Observando o output da ferramenta IOzone vemos que, para 2 processos, a média, por thread/processo, do thoughput para o teste de Initial Write foi de 1744.02 KB/s. Observamos agora o output do script de forma a ver qual a média obtida. Como estamos a ver a situação com dois processos apenas olhamos para o output relativo ao processo 1 e ao processo 2. Vemos que o throughput do teste initial write para o processo 1 foi de 2336.73132 KB/s e, para o segundo processo, foi de 3496.63431 KB/s. Assim, temos uma média de 2916.68282 KB/s. Mais uma vez, assim como no teste anterior, pode notar-se que o script obteve resultados maiores. No entanto note-se que o facto de obter resultados maiores acontece simplesmente por estarmos a testar com ficheiros muito pequenos porque, como iremos ver no próximo subcapítulo, para grandes ficheiros, a média foi o script devolver resultados menores.

Ao fim destes dois testes comprava-se que o script retorna resultados com sentido e apresenta output para todos os testes exatamente como pretendido, assim como output final com a acumulação dos valores intermédios. Na falta de alguma maneira de saber exatamente os valores mais corretos iremos confiar que o script nos providencia os melhores valores, visto que estes foram obtidos com uma instrumentação constante em tempo real o que, por si só, seria capaz em teoria de retornar valores mais fidedignos do que a instrumentação obtida pela ferramenta passiva IOzone. Para além disto temos ainda a questão mencionada anteriormente da precisão no cálculo dos tempos onde, como antes explicado, o script tem mais precisão. Isto pode levar a valores cada vez mais distintos entre o script e a ferramenta IOzone à medida que se aumenta o tamanho do ficheiro e, por conseguinte, se aumenta o tempo demorado nos testes.

Depois de testado o script desenvolvido e de detetadas as principais diferenças entre o benchmark com IOzone e com este script, apresentam-se os resultados para os testes realizados anteriormente para cada sistema de ficheiros, mas agora usando o script para obter resultados com benchmark ativo. Ao realizar estes testes podemos validar se as conclusões alcançadas com benchmark passivo estavam corretas. Dos testes realizados anteriormente apenas não se realizaram os testes em que cada operação acedia diretamente ao disco, pois os outros testes serão suficientes para provar certas ou provar erradas as conclusões alcançadas anteriormente relativamente à eficiência de cada sistema de ficheiros em relação a cada teste específico.

D. Análise dos Resultados com Benchmark Ativo - Sistema ZFS

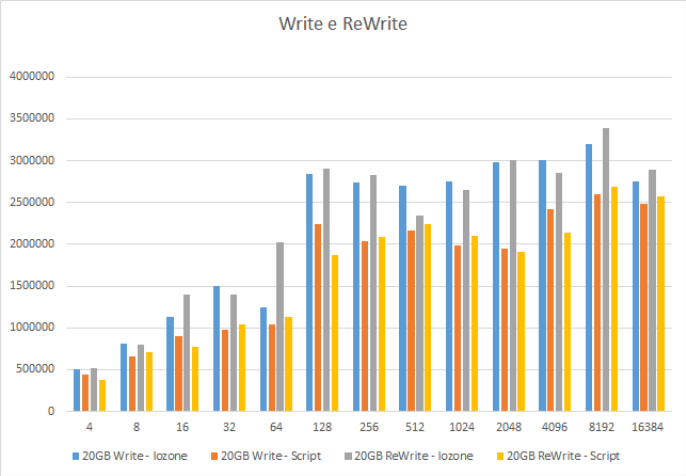
TESTE 1:

Usando os comandos:
/opt/csw/bin/iozone -f a72293iozone.tmp -a -i0 -s 20g
/opt/csw/bin/iozone -f a72293iozone.tmp -a -i0 -i1 -s 20g
/opt/csw/bin/iozone -f a72293iozone.tmp -a -i0 -i2 -s 20g
/opt/csw/bin/iozone -f a72293iozone.tmp -a -i0 -i5 -s 20g

dtrace -qs instrumentador.d a72293iozone.tmp single-threaded

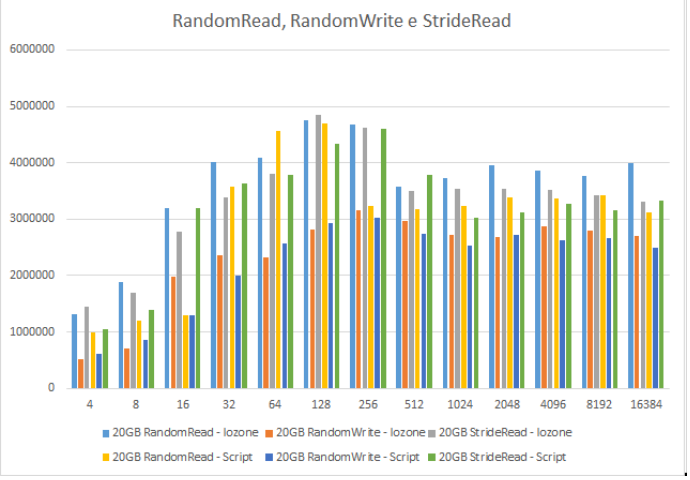
obtiveram-se os seguintes gráficos:

(Neste gráfico o eixo y representa KB/s e o eixo x o tamanho do record testado)



(Neste gráfico o eixo y representa KB/s e o eixo x o tamanho do record testado)

(Neste gráfico o eixo y representa KB/s e o eixo x o tamanho do record testado)



Com estes gráficos vemos que o panorama geral se manteve ao estudar as operações de I/O com o script, a diferença principal acabou por ser o facto de o script demonstrar valores muito menores de KB/s na grande maioria dos casos. Nota-se também que alguns resultados obtidos com o script revelam uma escalabilidade quanto ao tamanho do record usado menor. É normal que o script detecte nuances deste género de forma muito mais realista do que a ferramenta IOzone, pois o script faz uma instrumentação constante sempre que há um read ou um write enquanto que a ferramenta IOzone apenas reúne no final a quantidade de KB final envolvidos e o tempo total final e deriva os seus resultados a partir daí. Com a instrumentação em tempo real por parte do script então é possível (e realmente foi possível) determinar pequenas perturbações que possam ter acontecido. Por exemplo, como se vê no segundo gráfico, o teste de read tem uma subida brusca para o último record calculado pelo script. Esta subida não foi detetada pela ferramenta IOzone. Desta forma notamos que o script nos permite uma observação mais pormenorizada e realista, enquanto que os resultados da ferramenta IOzone são mais uma média do que se espera ter acontecido. Claro que, o facto do script instrumentar em tempo real poderá ter algum impacto na performance das operações I/O no ficheiro em si, por esta razão é sempre melhor, se queremos comparar máquinas ou sistemas de ficheiros distintos, compara-los com dados instrumentados obtidos da mesma forma, pois como se vê, se comparássemos os resultados obtidos com o script com os obtidos noutro lado com a ferramenta IOzone de forma a ver a melhor máquina seria injusto quanto á maquina onde se usou o script.

TESTE 2:

Testa-se agora o comportamento multi-threaded/multi-processo, mas tendo em conta o grande número de testes a realizar para este comportamento apenas se usa um tamanho de record (4KB).

Correndo agora os comandos:

/opt/csw/bin/iozone -i0 -s 20g -l2 -u8 -F a72293iozone1.tmp a72293iozone2.tmp a72293iozone3.tmp a72293iozone4.tmp a72293iozone5.tmp a72293iozone6.tmp a72293iozone7.tmp a72293iozone8.tmp

/opt/csw/bin/iozone -i0 -i1 -s 20g -l2 -u8 -F a72293iozone1.tmp a72293iozone2.tmp a72293iozone3.tmp a72293iozone4.tmp a72293iozone5.tmp a72293iozone6.tmp a72293iozone7.tmp a72293iozone8.tmp

/opt/csw/bin/iozone -i0 -i2 -s 20g -l2 -u8 -F a72293iozone1.tmp a72293iozone2.tmp a72293iozone3.tmp a72293iozone4.tmp a72293iozone5.tmp a72293iozone6.tmp a72293iozone7.tmp a72293iozone8.tmp

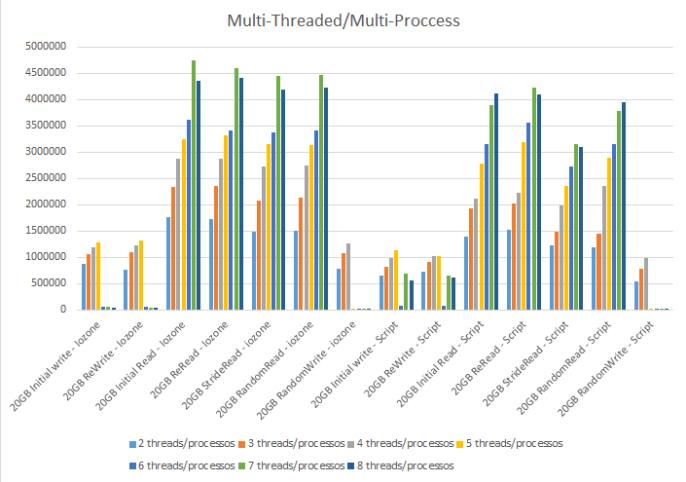
/opt/csw/bin/iozone -i0 -i5 -s 20g -l2 -u8 -F a72293iozone1.tmp a72293iozone2.tmp a72293iozone3.tmp a72293iozone4.tmp a72293iozone5.tmp a72293iozone6.tmp a72293iozone7.tmp a72293iozone8.tmp

dtrace -qs instrumentador.d a72293iozone1.tmp thread_-1 dtrace -qs instrumentador.d a72293iozone2.tmp thread_2 dtrace -qs instrumentador.d a72293iozone3.tmp thread_3 dtrace -qs instrumentador.d a72293iozone4.tmp thread_4 dtrace -qs instrumentador.d a72293iozone5.tmp thread_5 dtrace -qs instrumentador.d a72293iozone6.tmp thread_6

dtrace -qs instrumentador.d a72293iozone7.tmp thread_7
dtrace -qs instrumentador.d a72293iozone8.tmp thread_8

foi possível obter o seguinte gráfico:

(note-se que neste gráfico o eixo y representa KB/s e o eixo x o número de threads/processos)



Embora os resultados do script sejam muito parecidos com os obtidos pela ferramenta pode de facto notar-se diferenças, principalmente no teste de StrideRead onde os resultados são muito piores para qualquer número de threads do que os obtidos com IOzone. Nota-se também uma escalabilidade bastante diferente para as operações de Initial write e ReWrite. Vemos que com 7 e 8 threads/processos obtêm-se resultados ainda acima dos resultados sequenciais, enquanto que a ferramenta IOzone mostra resultados muito piores. Vemos com os resultados do script que com 6 threads/processos temos um grande impacto na performance por alguma razão que não se pode determinar, mas vemos que logo a seguir com 7 e 8 threads/processos esse impacto parece ultrapassado. O facto de a ferramenta IOzone demonstrar este enorme impacto para 6, 7 e 8 threads/processos revela que não foi capaz de detetar o problema como estando somente na situação de 6 threads/processos e induzindo-nos assim em erro levando-nos a crer que os testes em questão não escalavam minimamente acima do valor sequêncial para 7 e 8 threads/processos, quando de facto vemos pelos resultados do script que escalam.

Vemos de seguida os testes em abiente single-threaded/single-proccess para os restantes sistemas de ficheiros. Entendeu-se que não seria muito relevante testar extensivamente os restantes sistemas de ficheiros quanto aos testes em ambiente multi-threaded/multi-process, pois com alguns pequenos testes realizados com tamanhos de ficheiros mais pequenos notou-se que se verificava o mesmo que se verificou neste sistema de ficheiros: resultados menores em média e algumas diferenças pequenas, mas relevantes o suficiente para se notarem minimamente, na escalabilidade ao nível das threads, em especial dos testes que envolvem operações de escrita. Por esta razão não foram realizados gráficos para este segundo teste nos restantes sistemas de ficheiros.

E. Análise dos Resultados com Benchmark Ativo - Sistema UFS

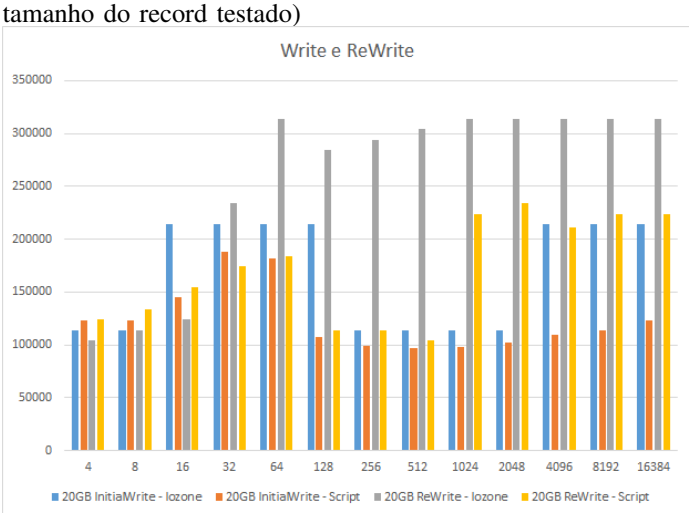
TESTE 1:

Usando os comandos:
/opt/csw/bin/iozone -f /share/jade/a72293iozone.tmp -a -i0 -s 20g
/opt/csw/bin/iozone -f /share/jade/a72293iozone.tmp -a -i0 -i1 -s 20g
/opt/csw/bin/iozone -f /share/jade/a72293iozone.tmp -a -i0 -i2 -s 20g
/opt/csw/bin/iozone -f /share/jade/a72293iozone.tmp -a -i0 -i5 -s 20g

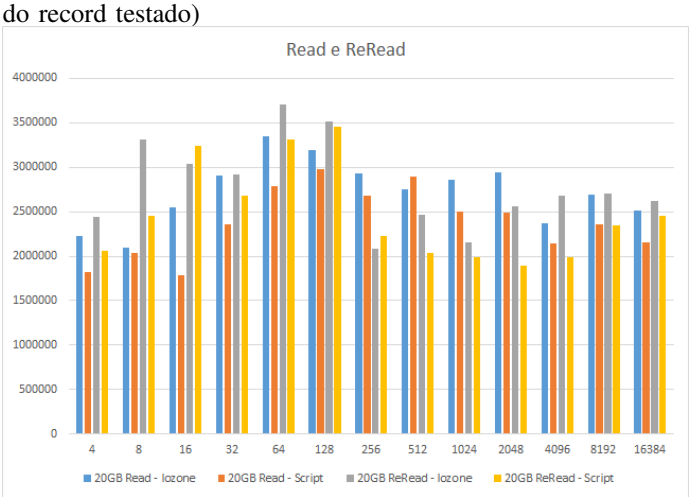
dtrace -qs instrumentador.d a72293iozone.tmp single-threaded

obtiveram-se os seguintes gráficos:

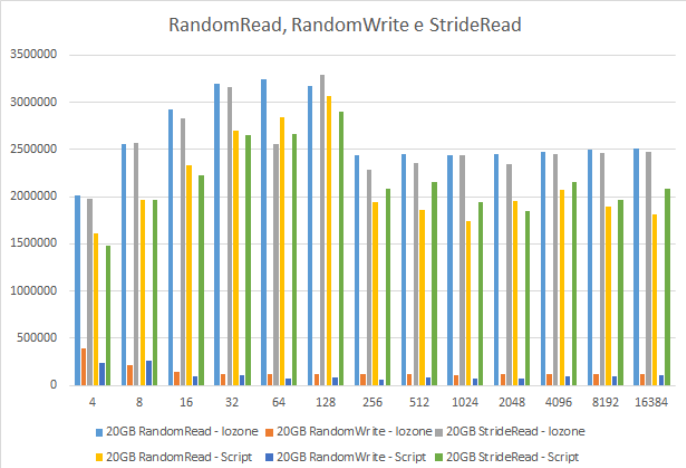
(Neste gráfico o eixo y representa KB/s e o eixo x o tamanho do record testado)

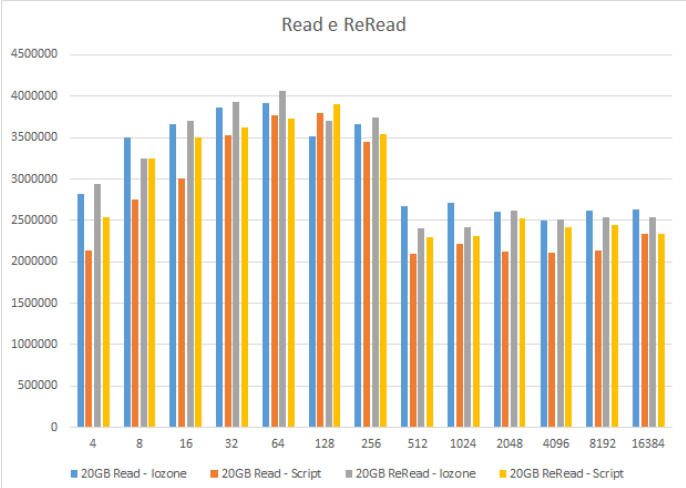


(Neste gráfico o eixo y representa KB/s e o eixo x o tamanho do record testado)



(Neste gráfico o eixo y representa KB/s e o eixo x o tamanho do record testado)





(Neste gráfico o eixo y representa KB/s e o eixo x o tamanho do record testado)

F. Análise dos Resultados com Benchmark Ativo - Sistema NFS

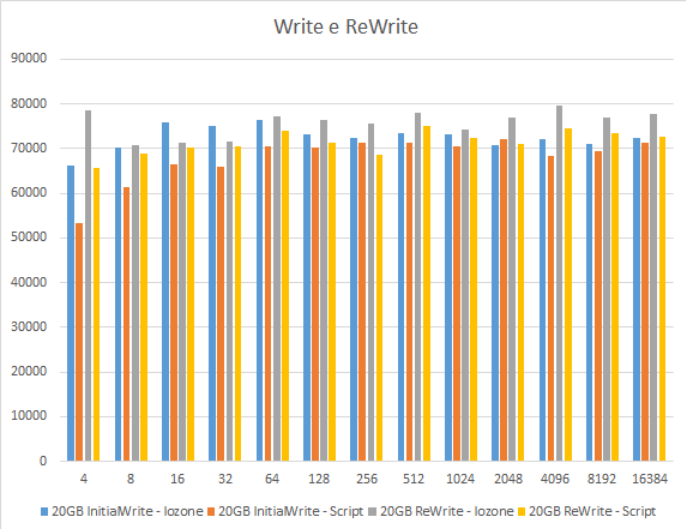
TESTE 1:

Usando os comandos:
/opt/csw/bin/iozone -f /discoHitachi/a72293iozone.tmp -a -i0 -s 20g
/opt/csw/bin/iozone -f /discoHitachi/a72293iozone.tmp -a -i0 -i1 -s 20g
/opt/csw/bin/iozone -f /discoHitachi/a72293iozone.tmp -a -i0 -i2 -s 20g
/opt/csw/bin/iozone -f /discoHitachi/a72293iozone.tmp -a -i0 -i5 -s 20g

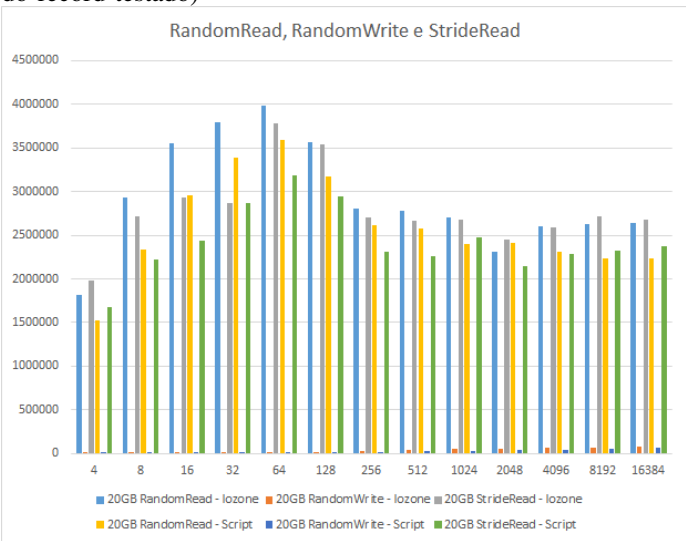
dtrace -qs instrumentador.d a72293iozone.tmp single-threaded

obtiveram-se os seguintes gráficos:

(Neste gráfico o eixo y representa KB/s e o eixo x o tamanho do record testado)



(Neste gráfico o eixo y representa KB/s e o eixo x o tamanho do record testado)



G. Análise dos Resultados com Benchmark Ativo - Conclusão

Por fim, depois de um minucioso estudo do script e comparando com a ferramenta IOzone nos três sistemas de ficheiros indicados podemos tirar algumas conclusões. Vemos que em média se mantêm as conclusões obtidas anteriormente sendo que se nota a diferença entre a performance de escritas e de leituras de uma forma um pouco mais acentuada, embora tal só tenha sido notado ao calcular o rácio dos resultados o que leva a crer que esta diferença poderá ser mera coincidência, pois é normal que dois testes exatamente iguais efetuados em alturas temporais diferentes retornem resultados que possam ser um pouco diferentes já que as condições nunca são as mesmas, poderá sempre haver mais ou menos processos externos ao nosso controlo a correr em background ou a máquina pode estar mais quente que o normal em determinado momento não permitindo a obtenção de uma melhor performance em determinados casos. em suma, novamente se afirma que o sistema ZFS é sem dúvida o que obtém a melhor performance em ambiente single-threaded/single-process e portanto deve ser preferido quando usado nestas condições na presente máquina. Em ambiente multi-threaded/multi-proccess, pelos pequenos testes realizados nos restantes sistemas de ficheiros,

vemos que se manteve a tendência do sistema UFS ser o que obtém a melhor performance e escalabilidade a nível de threads/processos para operações de escrita e leitura. Para operações de escrita continuou o sistema ZFS a apresentar a melhor escalabilidade e a melhor performance.

V. Conclusões e Trabalho Futuro

Terminado o estudo proposto verificaram-se diferenças bastante significativas entre benchmark passivo e ativo. Notou-se que, com benchmark passivo não podemos verdadeiramente afirmar sobre o que estamos a medir realmente. Neste estudo em específico realmente as conclusões alcançadas com ambos os tipos de benchmark foram semelhantes, mas noutro tipo de situação isto podia já não acontecer. Isto porque em nenhum lado a ferramenta IOzone nos dá certezas da forma como os seus resultados são calculados, pois não temos como saber se os tempos calculados incluem o tempo de sincronização dos dados no disco com o uso da chamada fsync ou não. No script esse tempo não é contabilizado, mas como os resultados de tempo são em média maiores podemos assumir em principio que a ferramenta IOzone também não os contabiliza, no entanto, sem ter feito este script e estes testes não poderiamos saber isso. Aqui vemos já um grande problema de fazer benchmark passivo: Achamos que medimos uma coisa, mas na realidade estamos a medir uma completamente distinta. Isto porque. nada impediria alguém de usar a ferramenta IOzone e assumir erradamente que esta contabilizava os tempos com fsync chegando assim a conclusões que poderiam não estar realmente corretas. Outro problema de benchmark passivo já mencionado anteriormente consiste no facto de pequenas perturbações no meio de um teste afetarem o resultado de forma ambígua. Com isto quer-se dizer: Se a meio de um teste que consiste em 100 leituras, uma das leituras, por alguma situação fora do normal, demora muito mais tempo que as restantes então este tempo a mais vai afetar o resultado do teste todo (100 leituras) impossibilitando que se compreenda se foi uma só leitura que demorou mais do que o normal (por isso devia até ser ignorada) ou então se foram todas as leituras que demoraram muito (o que implica uma baixa performance de parte da máquina). O benchmark ativo permitiria imprimir o resultado de cada leitura individualmente e assim o utilizador seria capaz de distinguir estas duas situações descritas.

Olhando agora para as conclusões alcançadas relativaente á performance entre os 3 tipos de sistemas de ficheiro testados pretende-se apenas acrescentar que os resultados não surpreenderam já que, o sistema que obteve a melhor performance em ambiente single-threaded/single-process (ZFS) encontrava-se montado num disco SSD que tem acessos à memória inerentemente mais rapidos do que um disco HDD, como o disco onde estava montado o sistema de ficheiros UFS, por exemplo.

Por fim deixa-se algum tabalho futuro que se acredita que poderia ser relevante. Olhando para o script desenvolvido nota-se que talvez não revele suficientemente benchmark ativo. Seria interessante imprimir os resultados de cada leitura/escrita individual em tempo real e assim ver se determinadas descidas bruscas na performance se devem a uma só leitura/escrita que demorou demasiado ou se realmente foi uma culpa do todo.

VI. Bibliografia

Referências

[1] http://www.iozone.org/docs/IOzone_msword_98.pdf

[2] http://www.oracle.com/technetwork/server-storage/solaris11/documentation/dtrace-cheatsheet-1930738.pdf

[3] http://dtrace.org/blogs/brendan/2011/11/09/solaris-11-dtrace-syscall-provider-changes/

[4] http://gec.di.uminho.pt/minf/cpd1314/PAC/material1314/IOZONE-Run_rules.pdf

[5] http://docs.oracle.com/cd/E23823_01/html/817-0404/chapter2-37.html

[6] http://codewiki.wikidot.com/c:system-calls:lseek

VII. Anexos

A. Anexo A

Flag	Descrição
-a	modo automático completo. Produz output que cobre todos os testes disponíveis para tamanhos de registo de 4k até 16M e tamanhos de ficheiro de 64k até 512M.
-A	igual a -a, mas permite uma análise mais densa dos testes realizados ao fazer testes até mesmo para situações pouco relevantes, em troca é muito mais demoroso do que -a.
-b filename	É criado um ficheiro binário compatível com Excel com o output dos resultados
-B	Faz com que todos os testes que acedem a ficheiros temporários que os criam e acedam com recurso à interface mmap(). É útil para saber como a máquina se comportaria com aplicações que tratam um ficheiro como um arrays de memória.
-c	Inclui close() no cálculo dos tempos. Permite observar o comportamento do sistema operativo em relação a close().
-C	Mostrar os bytes transferidos em testes de throughput. Permite observar problemas de starvation no I/O de ficheiros ou no gerênciamento de processos.
-d #	Permite atrasar um dado tempo em milisegundos entre o qual se liberta cada processo/thread durante testes de throughput.
-D	Provoca que todos os dados em espaço mmap seja escrito para o disco de forma assíncrona.
-e	Inclui flush (fsync,fflush) no cálculo dos tempos.
-f filename	Usado para especificar o nome do ficheiro temporário a ser usado durante os testes.
-F filename ...	Usado para especificar o nome dos ficheiros temporários a usar durante os testes de throughtput. O número de ficheiros deve ser igual ao número de processos/threads especificado.
-g #	Definir um tamanho máximo em Kbytes para o modo automático.
-G	Provoca que todos os dados em espaço mmap seja escrito para o disco de forma síncrona.
-h	Apresenta um ecrã de ajuda.
-H #	Permite o uso de POSIX async I/O com # operações asíncronas.
-i #	Usado para específicar o tipo de testes a executar (0=write/rewrite, 1=read/re-read, 2=random-read/write, 3=Read-backwards, 4=Re-write-record, 5=stride-read, 6=fwrite/re-fwrite, 7=fread/Re-fread, 8=random mix, 9=pwrite/Re-pwrite, 10=pread/Re-pread, 11=pwritev/Re-pwritev, 12=preadv/Repreadv). A opção 0 terá de ser sempre escolhida e várias podem ser escolhidas em simultânio para permitir vários testes diferentes de uma só vez.

Flag	Descrição
-I	Usa DIRECT I/O para todas as operações de ficheiros. Diz ao sistema de ficheiros para ignorar o buffer da cache e ir direto ao disco. em Solaris esta opção também irá utilizar directio().
-j #	Especifica o tamanho do espaçamento de leitura (que será # * tamanho do record) para testes de stride read.
-J #	Atrasa # milisegundos antes de cada operação de I/O.
-k #	Usar POSIX async I/O (sem bcopy) com # operações asíncronas.
-K	Gerar alguns acessos aleatórios durante os testes.
-l #	Especificar o número mínimo de processos/threads a usar quando se correrem testes de throughtput.
-L #	Especifica o tamanho da linha da cache em bytes. Usado internamente para acelerar os testes.
-m	Provoca o usos de multiplos buffers internamente em vez de um só. Assim simula-se aplicações que leiam para um mesmo buffer continuamente e aplicações que leiam para um array de buffers.
-M	Coloca a string resultante da chamada de uname() no ficheiro de output.
-n #	Define o tamanho de ficheiro mínimo para o modo automático.
-N	Reporta os resultados em microsgegundos por operação.
-o	Executa cada write de forma síncrona no disco abrindo os ficheiros sempre com a flag O_SYNC.
-O	Retorna os resultados em operações por segundo.
-p	Limpa a cache do processador entre cada operação de ficheiro permite observar o comportamento dos testes sem a aceleração obtida com o uso da cache em questão.
-q #	Define o tamanho máximo para o tamanho do record em Kbytes para o modo automático.
-Q	Gera ficheiros com dados sobre offset/latência de forma a permitir observar o impacto na latência que determinados offsets provocam.
-r #	Usado para especificar o tamanho do record em Kbytes a usar.
-R	Gera um ficheiro com o output obtido que pode ser lido com Excel e escreve-o no stdout.
-s #	Usado para especificar o tamanho em Kbytes do ficheiro a testar.
-S #	Indica o tamanho da cache do processador em Kbytes. É usado internamente, por exemplo, para fazer alinhamento dos buffers.

Flag	Descrição
-t #	Correr os testes em modo throughtput. Permite especificar o número de processos/threads a usar.
-T	Usa POSIX pthreads para os testes de throughtput.
-u #	Define o limite superior para o número de processos/threads a utilizar em testes de throughtput.
-U mountpoint	Para fazer mount e unmount entre cada teste, desta forma garante-se que a cache não tem nenhuma parte do ficheiro usado nos testes.
-v	Apresenta a versão da ferramente IOzone.
-V #	Especifica um padrão a ser escrito no ficheiro temporário e validado em cada um dos testes de leitura.
-w	Deixar os ficheiros temporários presentes no sistema de ficheiros depois da sua utilização.
-W	Reservar o acesso aos ficheiros aquando de uma leitura ou de uma escrita.
-x	Não usar stone-walling (técnica usada nos testes de throughtput).
-X filename	Usar o ficheiro especificado para escrever triplos com a seguinte informação: Byte offset, size of transfer, compute delay in milliseconds.
-y #	Especificar o tamanho mínimo para o tamanho de record em Kbytes para o modo automático.
-Y filename	Usar o ficheiro especificado para ler triplos com a seguinte informação: Byte offset, size of transfer, compute delay in milliseconds.
-z	Usado em conjunção com -a para testar todos os tamanhos de record possíveis, já que -a ignora os tamanhos demasiado pequenos quando os tamanhos de ficheiro são demasiado grandes.
-Z	Permite o uso de mixing mmap I/O e file I/O.
-+m filename	Usa o ficheiro especificado para obter informação sobre diversos clients para teste em cluster. O ficheiro deve conter uma linha por client e cada linha deve ter três campos separados por espaço. O primeiro campo é o nome do client, o segundo o caminho no cliente para a directoria onde IOzone deve trabalhar e o terceiro é o caminho no cliente para o executavel do IOzone.
-+n	Usar esta flag para impedir retests.
-+N	Impedir a truncagem ou eliminação sw ficheiros de teste anteriores antes do teste de escrita sequencial.
-+u	Permitir modo de utilização do CPU.
-+d	Permitir modo de diagnóstico.

Flag	Descrição
-+p percentage	Definir a percentagem de threads/processos que vão realizar testes de random read. Apenas válido em modo de throughtput com mais de 1 processo/thread.
-+r	Permitir O_RSYNC e O_SYNC para todos os testes de I/O.
-+t	Permitir testes de performance de rede.
-+A	Permitir madvise. 0 = normal, 1=random, 2=sequential, 3=dontneed, 4=willneed. Para uso com opções que ativem mmap().