Performance e Precisão Relativas de Filesystem Benchmarks e Disk Benchmarks

Active Benchmarking da ferramenta iozone recorrendo a DTrace em Solaris 11

Filipe Oliveira Departamento de Informática
Universidade do Minho
Email: a57816@alunos.uminho.pt

26 de Abril de 2016

Introdução – Contextualização da Ferramenta iozone

A necessidade de recurso a ferramentas de Benchmarking está implicitamente associada à necessidade de recolha de informação dos sistemas de computação no seu todo, ou de pontos específicos do mesmo, e sua posterior comparação e interpretação. Ora, quando queremos estudar especificamente a performance de um sistema de ficheiros e/ou de um determinado tipo de dispositivo de armazenamento de informação, podemos recorrer a ferramentas de traçado dinâmico (como o dtrace) por forma a confirmar os resultados obtidos e/ou complementar os mesmos, de uma forma que outrora era de extrema complexidade e dificuldade.

É importante realçar que a performance relativa dos sistemas de ficheiros e dispositivos de armazenamento está implicitamente ligada, uma vez que através de caching, buffering, e I/O assíncrono um sistema de ficheiros poderá "esconder"do nível da aplicação determinadas propriedades e lacunas do dispositivo físico de armazenamento. Retornando às ferramentas de benchmarking de sistemas de ficheiros e/ou dispositivos de armazenamento podemos compreender que as técnicas de software/hardware de otimização de tempos acesso aos dados, poderão implicar a benchmarking pouco precisa dos sistemas físicos presentes no sistema.

Todos os métodos de redução de tempos de acesso, ou seja, todos os métodos que tentam esconder a latência de acesso aos dados, deverão ser tidas em contacto no momento do benchmarking. Este trabalho prende-se precisamente com o estudo do quão precisa é a ferramenta de benchmarking iozone em sistemas de computação Solaris based, nomeadamente na máquina descrita na tabela 1.

Tabela 1: Características de Hardware do sistema de computação

Sistema	compute-641
# CPUs	2
CPU	Intel® Xeon® E5-2650 v2
Arquitectura de Processador	Ivy Bridge
# Cores por CPU	8
# Threads por CPU	16
Freq. Clock	2.6 GHz
Cache L1	256KB (32KB por Core)
Cache L2	2048KB (256KB por Core)
Cache L3	20480KB (partilhada)
Ext. Inst. Set	SSE4.2, AVX
#Memory Channels	4
Memória Ram Disponível	64GB
Peak Memory BW Fab. CPU	59.7 GB/s
Local FileSystems	zfs, ufs
NFS FileSystems	nfs, smb, autofs, smbfs

Aquando da realização de benchmarking via iozone, iremos recorrer a outras ferramentas (nomeadamente truss e dtrace) para procedermos à nossa própria benchmark – vulgo Active Benchmarking. Pretendemos com isso determinar quais os principais "bottlenecks"da porção do sistema a analisar, e ao mesmo tempo desenvolver capacidade crítica na análise de dados relativos a performance de sistemas de computação de complexidade acrescida.

Precisamos primeiramente de analisar de uma forma geral o comportamento da aplicação iozone no que concerne a system calls, traçando de seguida um plano de traçado dinâmico que corrobore/desminta o apresentado pela ferramenta. I

1 Passive Benchmarking :: uma primeira análise à ferramenta

Definida a ferramenta de benchmarking - iozone - existem diferentes opções a ter em conta, nomeadamente:

- tipo de operação – read, write, re-read, re-write, e outros
- tamanho do I/O
- forma de acesso aos dados – sequencial ou random
- Memory mapping – acesso á memória mmap em detrimento de read/write.

Teremos que ter ainda em conta na escolha do tamanho do dataset a memória disponível, assim como a quantidade dados passíveis de estarem presentes na memória cache. Só assim conseguiremos garantir que o recurso a uma system call irá resultar numa leitura/escrita em disco, ultrapassando os vários níveis de disfarce de latência.

Ora, dado que é na escrita dos dados que acreditamos que existirá uma maior possibilidade de discrepância de resultados pelo iozone versus os que iremos obter por medição direta (dados os inúmeros mecanismos como caching, buffering, e I/O assíncrono, permitirem a uma kernel efetuar uma escrita sem necessariamente os dados serem imediatamente escritos no disco), tomaremos especial atenção à performance de escrita medida pela ferramenta.

Ora, e atendendo ao seguinte excerto das Iozone Run rules:

> **For disk performance comparisons:**
> 1. For single stream results, be sure the file size is 3 times the size of the buffer cache. For throughput results, be sure the aggregate of files is 3 times the size of the buffer cache.

Daqui retiramos que o tamanho do ficheiro deverá ser 3 vezes superior à buffer cache disponível nos sistema de computação. Precisamos portanto de determinar esse valor, dado pelo parâmetro **UFS bufhwm**. Atente na sua descrição:

> **UFS Parameters**
> **bufhwm and bufhwm_pct**
> **Description:**
> Defines the maximum amount of memory for caching I/O buffers. The buffers are used for writing file system metadata (superblocks, inodes, indirect blocks, and directories). Buffers are allocated as needed until the amount of memory (in KB) to be allocated exceed bufhwm. At this point, metadata is purged from the buffer cache until enough buffers are reclaimed to satisfy the request.
>
> `http://docs.oracle.com/cd/E23823_01/html/817-0404/chapter2-37.html`

Ora, recorrendo ao comando **sysdef** podemos extrair a informação desejada:

```
 sysdef
 ...
*
* Tunable Parameters
*
1373298688 maximum memory allowed in buffer cache (bufhwm)
...
```

Poderíamos ser induzidos em erro e considerar como 3 * 1373298688 KB o tamanho do ficheiro a utilizar. Contudo, um olhar mais atento à continuação da definição de **bufhwm**:

> **Range**
> 80 KB to 20 percent of physical memory, or 2 TB, whichever is less. Consequently, bufhwm_pct can be between 1 and 20.
> **Validation**
> If bufhwm is less than its lower limit of 80 KB or **greater than its upper limit** (the lesser of 20 percent of physical memory, 2 TB, or one quarter (1/4) of the maximum amount of kernel heap), it is reset to the upper limit. The following message appears on the system console and in the /var/adm/messages file if an invalid value is attempted:
>
> `http://docs.oracle.com/cd/E23823_01/html/817-0404/chapter2-37.html`

leva-nos a calcular o valor máximo de bufhwm como (20% de 65501000 KB) * 3 = 39300600 KB = 39.3006 GB.

1.1 Command Line Options

Analisando ainda as "Command Line Options" da ferramenta podemos desde já enumerar algumas opções que devemos incluir:

- **-i 0** – Used to specify which tests to run. (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/Re- preadv).
- **-s 39g** – Used to specify the size, in Kbytes, of the file to test. One may also specify -s k (size in Kbytes) or -s m (size in Mbytes) or -s g (size in Gbytes).
- **-S 20480** – Set processor cache size to value (in Kbytes). This tells Iozone the size of the processor cache. It is used internally for buffer alignment and for the purge functionality.
- **-l 1** – Set the lower limit on number of processes to run. When running throughput tests this option allows the user to specify the least number of processes or threads to start. This option should be used in conjunction with the -u option.
- **-u 1** – Set the upper limit on number of processes to run. When running throughput tests this option allows the user to specify the greatest number of processes or threads to start. This option should be used in conjunction with the -l option.
- **-b /export/home/a57816/ESC_ACTIVE_BENCHMARKING_HOME/teste_write.xls** – Iozone will create a binary file format file in Excel compatible output of results.
- **-R** – Generate Excel report. Iozone will generate an Excel compatible report to standard out. This file may be imported with Microsoft Excel (space delimited) and used to create a graph of the filesystem performance. Note: The 3D graphs are column oriented. You will need to select this when graphing as the default in Excel is row oriented data.

Assim, analisemos resultado do seguinte comando iozone:

```
/opt/csw/bin/iozone -+u -R -i 0 -S 20480 -s 39g -b /export/home/a57816/↩
    ESC_ACTIVE_BENCHMARKING_HOME/teste_write.xls -l 1 -u 1
```

que apresenta o seguinte resultado:

```
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 Apr 18 19:34:12 2016

CPU utilization Resolution = 0.000 seconds.
CPU utilization Excel chart enabled
Excel chart generation enabled
File size set to 40894464 kB
Command line used: /opt/csw/bin/iozone -+u -R -i 0 -S 20480 -s 39g -b /↩
    export/home/a57816/ESC_ACTIVE_BENCHMARKING_HOME/teste_write.xls -l 1 -↩
    u 1
Output is in kBytes/sec
Time Resolution = 0.000001 seconds.
Processor cache size set to 20480 kBytes.
Processor cache line size set to 32 bytes.
File stride size set to 17 * record size.
Min process = 1
Max process = 1
Throughput test with 1 process
Each process writes a 40894464 kByte file in 4 kByte records

Children see throughput for  1 initial writers  =  489166.09 kB/sec
Parent sees throughput for  1 initial writers  =  406767.30 kB/sec
Min throughput per process    =  489166.09 kB/sec
Max throughput per process    =  489166.09 kB/sec
Avg throughput per process    =  489166.09 kB/sec
Min xfer      = 40894464.00 kB
CPU Utilization: Wall time   83.600    CPU time   77.460    CPU ↩
    utilization  92.65 %


Children see throughput for  1 rewriters  =  463768.00 kB/sec
Parent sees throughput for  1 rewriters  =  402978.57 kB/sec
Min throughput per process    =  463768.00 kB/sec
Max throughput per process    =  463768.00 kB/sec
Avg throughput per process    =  463768.00 kB/sec
Min xfer      = 40894464.00 kB
CPU utilization: Wall time   88.179    CPU time   71.719    CPU ↩
    utilization  81.33 %




"Throughput report Y-axis is type of test X-axis is number of processes"
"Record size = 4 kBytes "
"Output is in kBytes/sec"

"  Initial write "  489166.09

```

```
"        Rewrite "   463768.00

"CPU utilization report Y-axis is type of test X-axis is number of processes"
"Record size = 4 kBytes "
"Output is in CPU%"

"   Initial write "       92.65

"        Rewrite "       81.33

iozone test complete.
```

1.2 Take 1 – uma análise com iostat

Ora, dos resultados anteriores, retiramos que o máximo de throughput do sistema de ficheiros e dispositivo de armazenamento físico tem o valor de 489166.09 KB/s para operações de escrita e 463768.00 KB/s para operações de re-escrita. Este último resultado apresentado leva-nos a duvidar da precisão de medição da ferramenta. Outro indicador que nos leva a duvidar da veracidade da medição prende-se com os valores altíssimos de utilização de CPU time de uma aplicação implicitamente **IO BOUND**.

Corramos novamente a ferramenta analisando agora o input/output recorrendo à ferramenta **iostat**, criando uma relação gráfica entre os valores apresentados por ambas as leituras na figura 1.

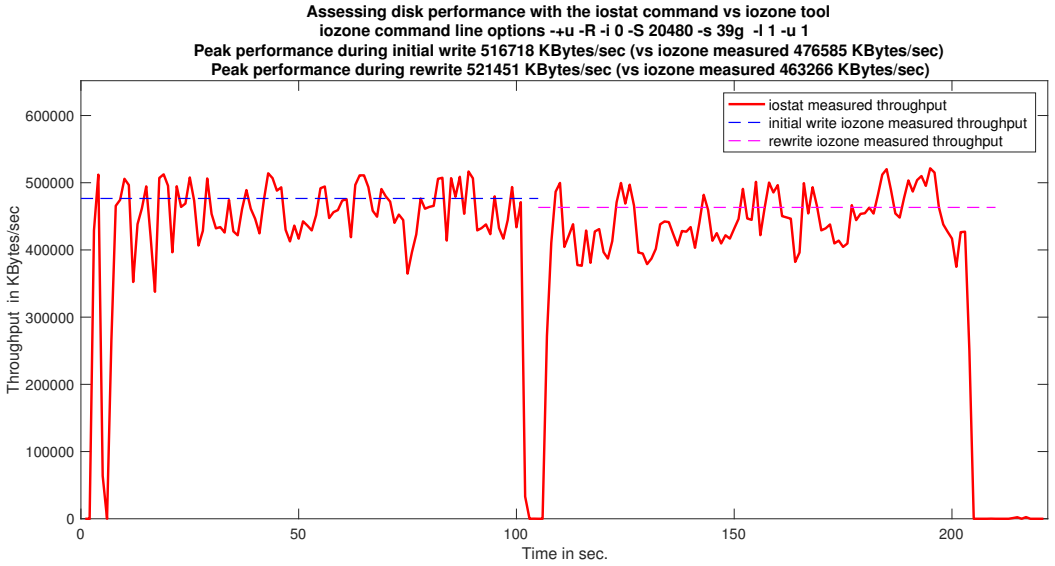

Figura 1: Relação de disk performance medida pela ferramenta iostat vs performance medida pela ferramenta **iozone**. Para command line options /opt/csw/bin/iozone -+u -R -i 0 -S 20480 -s 39g -b /export/home/a57816/ESC_ACTIVE_BENCHMARKING_HOME/teste_write.xls -l 1 -u 1

Tal como suspeitávamos existe alguma imprecisão nos valores apresentados pela ferramenta **iozone** vs os apresentados pela ferramenta **iostat**. Devemos sempre considerar esta análise com o iostat como algo introdutória à análise do problema – não devemos descurar o facto de o o iostat apresentar valores globais do sistema (qualquer processo de outro utilizador poderá influenciar os resultados medidos). Bastaria outro utilizador efectuar alguma operação de escrita/leitura para invalidar os nossos resultados. Devemos então considerar este primeiro como uma avaliação "brusca".

Como era previsível o peak throughput aconteceu na acção de rewrite (521451 KBytes/sec), através da medição da ferramenta **iostat**. Tal não se revelou verdade novamente nos valores

apresentados pela ferramenta **iozone**.

Existem portanto outras ferramentas como dstat que irão aprofundar o nosso conhecimento sobre o "modus operandi do iozone".

Façamos uma análise mais profunda das systems calls que a ferramenta **iozone** realiza para o cálculo dos resultados.

1.3 Take 2 – uma análise com truss

Analisemos o tipo de system calls realizada pela ferramenta recorrendo ao seguinte comando **truss**:

```
truss -o truss_report_write.txt -df /opt/csw/bin/iozone -+u -R -i 0 -S 20480 -s 39g -b /export/home/a57816/ESC_ACTIVE_BENCHMARKING_HOME/teste_write.xls -l 1 -u 1
```

Da análise do ficheiro truss_report_write.txt reparamos que o ficheiro de escrita do nosso interesse é o textbfiozone.DUMMY.0 tal como confirmado pelo seguinte excerto:

```
...
...
9990:- 0.0986-open("iozone.DUMMY.0", O_RDWR|O_CREAT, 0640)-= 5
9989:- 0.0993-pollsys(0xFFFF80FFBFFFF8A0, 0, 0xFFFF80FFBFFFF920, 0x00000000) = 0
9989:- 0.0994-getpid()----------= 9989 [9988]
9990:- 0.1000-pollsys(0xFFFF80FFBFFFEE80, 0, 0xFFFF80FFBFFFEF00, 0x00000000) = 0
9990:- 0.1002-getrusage(0xFFFF80FFBFFFEEA0)-----= 0
9990:- 0.1004-write(5, " y y y y y y y y\0\0\0\0"..., 4096)-= 4096
9990:- 0.1007-write(5, " y y y y y y y y\0\0\0\0"..., 4096)-= 4096
9990:- 0.1010-write(5, " y y y y y y y y\0\0\0\0"..., 4096)-= 4096
9990:- 0.1012-write(5, " y y y y y y y y\0\0\0\0"..., 4096)-= 4096
9990:- 0.1015-write(5, " y y y y y y y y\0\0\0\0"..., 4096)-= 4096
9990:- 0.1018-write(5, " y y y y y y y y\0\0\0\0"..., 4096)-= 4096
9990:- 0.1020-write(5, " y y y y y y y y\0\0\0\0"..., 4096)-= 4096
9990:- 0.1023-write(5, " y y y y y y y y\0\0\0\0"..., 4096)-= 4096
9990:- 0.1025-write(5, " y y y y y y y y\0\0\0\0"..., 4096)-= 4096
...
...
```

Podemos retirar dois aspetos importantes do seguinte excerto, sendo estes o textbfnome do ficheiro – iozone.DUMMY.0 , o seu respetivo **descritor de ficheiro – 5** e **tamanho de record utilizado - 4096 bytes** .

Podemos ainda analisar o momento do término da operação de initial write. Antes de se proceder ao fecho do descritor de ficheiro é garantida a sincronização dos dados com disco (conferir linhas 10223828 e 10223829 do próximo excerto) através da syscall **fdsync**.

Reparamos ainda em outro detalhe de interesse. Apenas quando corremos a ferramenta iozone com a opção **-+u** é verifica a ocorrência da medição de valores de utilização de cpu via via **getrusage**[1](conferir linha 10223827 do próximo excerto). Inicialmente pensávamos que esta syscall era utilizada para cálculo do tempo total da operação de escrita, contudo, correndo a ferramenta sem a opção **-+u** tal syscall não é encontrada.

```
...
...
10223819 9990:-380.0682--write(5, " y y y y y y y y\0\0\0\0"..., 4096)-= 4096
10223820 9990:-380.0683--write(5, " y y y y y y y y\0\0\0\0"..., 4096)-= 4096
10223821 9990:-380.0683--write(5, " y y y y y y y y\0\0\0\0"..., 4096)-= 4096
10223822 9990:-380.0683--write(5, " y y y y y y y y\0\0\0\0"..., 4096)-= 4096
10223823 9990:-380.0684--write(5, " y y y y y y y y\0\0\0\0"..., 4096)-= 4096
```

[1] http://linux.die.net/man/2/getrusage

```
10223824 9990:-380.0684--write(5, "yyyyyyyy\0\0\0\0"..., 4096)--= ↩ 4096
10223825 9990:-380.0684--write(5, "yyyyyyyy\0\0\0\0"..., 4096)--= ↩ 4096
10223826 9990:-380.0685--write(5, "yyyyyyyy\0\0\0\0"..., 4096)--= ↩ 4096
10223827 9990:-380.0685--getrusage(0xFFFF80FFBFFFEEA0)------= 0
10223828 9990:-380.6926--fdsync(5, FSYNC)---------= 0
10223829 9990:-380.6928--close(5)----------= 0
10223830 9990:-380.6930--_exit(0)
...
...
```

Continuando a análise do ficheiro truss_report_write.txt reparamos que o ficheiro de escrita do nosso interesse continua a ser o **iozone.DUMMY.0**.

```
...
...
10223853 9992:-383.3986--open("iozone.DUMMY.0", O_RDWR)-------= 5
10223854 9989:-383.3994--pollsys(0xFFFF80FFBFFFF8A0, 0, 0xFFFF80FFBFFFF920,↩ 0x00000000) = 0
10223855 9989:-383.3996--getpid()----------= 9989 [9988]
10223856 9992:-383.4000--pollsys(0xFFFF80FFBFFFEE90, 0, 0xFFFF80FFBFFFEF10,↩ 0x00000000) = 0
10223857 9992:-383.4002--getrusage(0xFFFF80FFBFFFEEB0)------= 0
10223858 9992:-383.4004--getrusage(0xFFFF80FFBFFFEEB0)------= 0
10223859 9992:-383.4007--write(5, "yyyyyyyy\0\0\0\0"..., 4096)--= ↩ 4096
10223860 9992:-383.4010--write(5, "yyyyyyyy\0\0\0\0"..., 4096)--= ↩ 4096
10223861 9992:-383.4013--write(5, "yyyyyyyy\0\0\0\0"..., 4096)--= ↩ 4096
10223862 9992:-383.4016--write(5, "yyyyyyyy\0\0\0\0"..., 4096)--= ↩ 4096
10223863 9992:-383.4018--write(5, "yyyyyyyy\0\0\0\0"..., 4096)--= ↩ 4096
10223864 9992:-383.4021--write(5, "yyyyyyyy\0\0\0\0"..., 4096)--= ↩ 4096
10223865 9992:-383.4023--write(5, "yyyyyyyy\0\0\0\0"..., 4096)--= ↩ 4096
10223866 9992:-383.4026--write(5, "yyyyyyyy\0\0\0\0"..., 4096)--= ↩ 4096
10223867 9992:-383.4028--write(5, "yyyyyyyy\0\0\0\0"..., 4096)--= ↩ 4096
10223868 9992:-383.4031--write(5, "yyyyyyyy\0\0\0\0"..., 4096)--= ↩ 4096
10223869 9992:-383.4034--write(5, "yyyyyyyy\0\0\0\0"..., 4096)--= ↩ 4096
10223870 9992:-383.4036--write(5, "yyyyyyyy\0\0\0\0"..., 4096)--= ↩ 4096
10223871 9992:-383.4039--write(5, "yyyyyyyy\0\0\0\0"..., 4096)--= ↩ 4096
10223872 9992:-383.4042--write(5, "yyyyyyyy\0\0\0\0"..., 4096)--= ↩ 4096
10223873 9992:-383.4044--write(5, "yyyyyyyy\0\0\0\0"..., 4096)--= ↩ 4096
...
...
```

1.4 Take 3 – uma análise com dtrace

Deste padrão de open() , write(), e read() podemos criar um primeiro ficheiro dtrace que calcule o tempo entre as syscalls open() e close() com o descritor de ficheiro = 5, e calcular o total de bytes escrito pelas syscalls write() entre open() e close(). Por recorrermos ao provider **fsinfo** estamos ainda a analisar todo o sistema. Considere portanto este script D-Trace como um introdutório para a validação dos pressupostos até ao momento assumidos, e da precisão dos tempos de medição. Em seções futuras do relatório analisaremos outras formas de calcular tanto os tempos como os valores de escrita e leitura em disco.

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

#pragma D option quiet

syscall::open*:entry
{
  self->pathname = copyinstr(arg1);
}

syscall::open*:return
/strstr(self->pathname,"iozone.DUMMY.0") != NULL /
{
  self->start = timestamp;
  total_size = 0;
}

fsinfo:::write
/strstr(args[0]->fi_pathname,"iozone.DUMMY.0") != NULL /
{
  kb = arg1 / 1024;
  total_size = total_size + kb ;
}

syscall::close*:return
/total_size > 0 && self->start /
{
  self->stop_t = timestamp;
  printf("\n\n################################################\n");
  printf("total time: %d\n", self->stop_t - self->start );
  printf("total size: %d\n", total_size);
  printf("################################################\n\n");
  total_size = 0;
}
```

Ora, a execução do script dtrace e ferramenta iozone resultou no seguinte output:

```
 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: Tue Apr 19 03:18:52 2016

 CPU utilization Resolution = 0.000 seconds.
```

```
 CPU utilization Excel chart enabled
 Excel chart generation enabled
 File size set to 40894464 kB
 Command line used: /opt/csw/bin/iozone -+u -R -i 0 -S 20480 -s 39g -b /↩
     export/home/a57816/ESC_ACTIVE_BENCHMARKING_HOME/teste_write.xls -l 1 -↩
     u 1
 Output is in kBytes/sec
 Time Resolution = 0.000001 seconds.
 Processor cache size set to 20480 kBytes.
 Processor cache line size set to 32 bytes.
 File stride size set to 17 * record size.
 Min process = 1
 Max process = 1
 Throughput test with 1 process
 Each process writes a 40894464 kByte file in 4 kByte records

 Children see throughput for  1 initial writers  =  326724.00 kB/sec
 Parent sees throughput for  1 initial writers  =  309091.96 kB/sec
 Min throughput per process    =  326724.00 kB/sec
 Max throughput per process    =  326724.00 kB/sec
 Avg throughput per process    =  326724.00 kB/sec
 Min xfer      = 40894464.00 kB
 CPU Utilization: Wall time  125.165    CPU time  125.165    CPU ↩
     utilization 100.00 %



#################################################
total time: 132303544264
total size: 40894464
#################################################


 Children see throughput for  1 rewriters  =  338724.72 kB/sec
 Parent sees throughput for  1 rewriters  =  320792.40 kB/sec
 Min throughput per process    =  338724.72 kB/sec
 Max throughput per process    =  338724.72 kB/sec
 Avg throughput per process    =  338724.72 kB/sec
 Min xfer      = 40894464.00 kB
 CPU utilization: Wall time  120.731    CPU time  120.730    CPU ↩
     utilization 100.00 %



#################################################
total time: 127477939460
total size: 40894464
#################################################




"Throughput report Y-axis is type of test X-axis is number of processes"
"Record size = 4 kBytes "
"Output is in kBytes/sec"

"  Initial write "  326724.00

"        Rewrite "  338724.72


"CPU utilization report Y-axis is type of test X-axis is number of ↩
    processes"
```

```
"Record size = 4 kBytes "                                                    77
"Output is in CPU%"                                                          78
                                                                             79
"  Initial write "      100.00                                               80
                                                                             81
"        Rewrite "      100.00                                               82
                                                                             83
                                                                             84
iozone test complete.                                                        85
```

Para o caso de initial write o script dtrace calculou 132303544264 nano-segundos como o tempo necessário à escrita – equivalente a 132.30 segundos. Encontramos já neste valor uma pequena discrepância versus os 125.165 segundos de Wall time apresentados pela ferramenta iozone. Analisemos o throughput médio calculado: 40894464 KB / 132.30 seg = 309104.036 KB/sec, também este diferente dos 326724.00 KB/sec calculados pela ferramenta iozone.

Procedendo de igual forma para re-write temos como tempo calculado pelo script dtrace 127477939460 nano-segundos como o tempo necessário à re-escrita – equivalente a 127.477 segundos. Encontramos novamente uma pequena discrepância versus os 120.731 segundos de Wall time apresentados pela ferramenta iozone. Analisemos o throughput médio calculado: 40894464 KB / 127.477 seg = 320798.763 KB/sec , também este diferente dos 338724.72 KB/sec calculados pela ferramenta iozone.

De um modo constante a ferramenta iozone foi mais "otimista"nos valores medidos, e acreditamos que a **diferença de valores obtidos se prende com a necessidade de aumentar a granularidade dos testes**.

1.5 Take 4 – uma análise com dtrace via provider syscall

1.5.1 Operações de Escrita

À anterior análise das syscalls open(), close(), write(), e read(), foram adicionadas ações de análise às probes que analisam as syscalls lseek e fdsync. Denote que tal com enunciado anteriormente era necessária aumentar a precisão e certeza dos testes. Para isso foi utilizado o provider **syscall** em detrimento do provider **fsinfo**. Desta forma, estaremos preparados para analisar os testes de read, re-read, write, re-write, random-read, e random-write.

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

#pragma D option quiet

dtrace:::BEGIN{
  opp_num = 1;

}

syscall::openat*:entry{
  self->time_in = timestamp;
  self->file = arg1;
}

syscall::openat*:return
/self->file /
{
self->pathname = copyinstr(self->file);
   flag = (strstr(self->pathname,"iozone.DUMMY.0") != NULL ) ?  1 : 0;
   self->time_out = timestamp;
  self->total_t = (strstr(self->pathname,"iozone.DUMMY.0") != NULL ) ? ( ↩
      self->time_out - self->time_in ) / 1000 : 0 ;
  @total_time[opp_num] = sum  (self->total_t);
  @nofdsync_time[opp_num] = sum (self->total_t);
}

syscall::read*:entry
/flag == 1/
{
  self->time_in = timestamp;
  self->r_size = arg2/1024;
  @total_r_size[opp_num] =  sum (self->r_size);
}

syscall::read*:return
/flag == 1 &&  self->r_size/
{
   self->time_out = timestamp;
  self->total_t = ( self->time_out - self->time_in ) / 1000;
  @total_time[opp_num] = sum  (self->total_t);
  @nofdsync_time[opp_num] = sum (self->total_t);
  self->r_size = 0;
}

syscall::fdsync*:entry
/flag == 1/
{
  self->time_in = timestamp;
  @total_fdsync[opp_num] =  count();
}

syscall::fdsync*:return
/flag == 1 /
{
```

```
  self->time_out = timestamp;
  self->total_t = ( self->time_out - self->time_in )/ 1000;
  @total_time[opp_num] = sum  (self->total_t);

}


syscall::write*:entry
/flag == 1/
{
  self->time_in = timestamp;
  self->w_size = arg2 / 1024;
  @total_w_size[opp_num] =  sum (self->w_size);
}

syscall::write*:return
/flag == 1 && self->w_size/
{
  self->time_out = timestamp;
  self->total_t = ( self->time_out - self->time_in )/ 1000;
  @total_time[opp_num] = sum  (self->total_t);
  @nofdsync_time[opp_num] = sum (self->total_t);
  self->w_size = 0;
}

syscall::lseek*:entry
/flag == 1 /
{
  self->time_in = timestamp;
  @lseek_offset[opp_num] = quantize (arg1);
  @lseek_count[opp_num] = count();
}

syscall::lseek*:return
/flag == 1 /
{
  self->time_out = timestamp;
  self->total_t = ( self->time_out - self->time_in ) / 1000;
  @total_time[opp_num] = sum  (self->total_t);
  @nofdsync_time[opp_num] = sum (self->total_t);
}

syscall::close*:entry
/flag == 1 /
{
  self->time_in = timestamp;
}


syscall::close*:return
/flag == 1 /
{
  self->time_out = timestamp;
  self->total_t = ( self->time_out - self->time_in ) / 1000;
  @total_time[opp_num] = sum  (self->total_t);
  @nofdsync_time[opp_num] = sum (self->total_t);
  opp_num = opp_num + 1;
  flag = 0;
}

dtrace:::END{
  printf("%-5s\t%20s\t%20s\t%15s\t%15s\t%10s\t%10s\n","#opp", "Total T. ↩
      elapsed", "No fdsynk T. elapsed", "T. Wr. KB", "T. Rd. KB", "#fdsync"↩
```

```
      , "#lseek");
  printa("%-5d\t%@20d\t%@20d\t%@15d\t%@15d\t%@10d\t%@10d\n"        , ↩
      @total_time        , @nofdsync_time       , @total_w_size , ↩
      @total_r_size , @total_fdsync , @lseek_count );
  printf("\n\n Lseek offset analysis:\n\n\n\n\n\n");
  printa( @lseek_offset );

  clear ( @lseek_offset );
  clear ( @total_time );
  clear ( @nofdsync_time );
  clear ( @total_w_size );
  clear ( @total_r_size );
  clear ( @total_fdsync );
  clear ( @lseek_offset );

}
```

Denote que foi adicionada à script D-Trace a funcionalidade de impressão da distribuição da posição(offset) do ficheiro de leitura/escrita via a syscall **lseek**.

Ora, a execução do script dtrace de escrita e re-rescrita, e da ferramenta iozone com as opções:

```
/opt/csw/bin/iozone -+u -i 0 -S 20480 -s 1g -l 1 -u 1
```

resultou no seguinte output:

```
        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: Tue Apr 26 05:14:08 2016

        CPU utilization Resolution = 0.000 seconds.
        CPU utilization Excel chart enabled
        File size set to 1048576 kB
        Command line used: /opt/csw/bin/iozone -+u -i 0 -S 20480 -s 1g -l 1 -u 1
        Output is in kBytes/sec
        Time Resolution = 0.000001 seconds.
        Processor cache size set to 20480 kBytes.
        Processor cache line size set to 32 bytes.
        File stride size set to 17 * record size.
        Min process = 1
        Max process = 1
        Throughput test with 1 process
        Each process writes a 1048576 kByte file in 4 kByte records

        Children see throughput for  1 initial writers  =  371087.25 kB/sec
        Parent sees throughput for  1 initial writers   =  198815.72 kB/sec
        Min throughput per process                      =  371087.25 kB/sec
        Max throughput per process                      =  371087.25 kB/sec
        Avg throughput per process                      =  371087.25 kB/sec
        Min xfer                                        = 1048576.00 kB
        CPU Utilization: Wall time    2.826    CPU time    2.826    CPU utilization ↩
            100.00 %


        Children see throughput for  1 rewriters        =  402761.41 kB/sec
        Parent sees throughput for  1 rewriters         =  196457.28 kB/sec
        Min throughput per process                      =  402761.41 kB/sec
        Max throughput per process                      =  402761.41 kB/sec
        Avg throughput per process                      =  402761.41 kB/sec
        Min xfer                                        = 1048576.00 kB
        CPU utilization: Wall time    2.603    CPU time    2.603    CPU utilization ↩
            100.00 %
```

```
iozone test complete.
#opp   Total T. elapsed      No fdsync T. elapsed   WR KB   RD KB        #fdsync   #lseek
1          4721288                 2275330          1050982        80          1     0
2          4789564                 2059135          1050956        80          1     0


 Lseek offset analysis:
```

Dedique especial atenção ao final do ficheiro de output:

```
#opp Total T. elapsed  No fdsync T. elapsed WR KB     RD KB #fdsync   #lseek
1          4721288                 2275330          1050982        80          1     0
2          4789564                 2059135          1050956        80          1     0
```

Reparamos que para ficheiros de grande tamanho o tempo demonstrado como o total pela ferramenta iozone parecia desmedido versus o tempo total que a mesma levava para terminar corretamente a execução. Ora, como se pode verificar pelo script e pelo resultado de execução anterior, a ferramenta não está a calcular como tempo de escrita o tempo que perde com a syscall **fdsync**. A mesma tem um valor de tempo total 2.826 segundos para a primeira operação e 2.603 segundos para a segunda, versus 2.2753 segundos (2275330 micro-segundos) para a primeira operação e 2.059 segundos (2059135 micro-segundos) para a segunda. Ora, se a ferramenta considera-se os valores do tempo na syscall **fsync** os valores seriam mais aproximados de 4.7212 segundos (4721288 micro-segundos) para a primeira operação e 4.7896 segundos (4789564 micro-segundos) para a segunda.

Um pouco para nosso espanto a medição do tempo é terminada antes da garantia de sincronização dos dados com disco. Ou seja, este tipo de medição não será certamente o mais preciso uma vez que poderão ocorrer discrepâncias entre o tempo calculado pela ferramenta e o tempo real que a ferramenta demora a completar a operação de escrita e sincronização. **Encontramos a primeira imprecisão!**

Verifiquemos a qualidade da medição da ferramenta para operações de leitura.

1.5.2 Operações de Escrita

Ora, a execução do script dtrace de escrita e re-rescrita, e da ferramenta iozone com as opções:

```
/opt/csw/bin/iozone -+u -i 1 -S 20480 -s 1g -l 1 -u 1
```

resultou no seguinte output:

```
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: Tue Apr 26 05:56:06 2016

CPU utilization Resolution = 0.000 seconds.
CPU utilization Excel chart enabled
File size set to 1048576 kB
Command line used: /opt/csw/bin/iozone -+u -i 0 -i 1 -S 20480 -s 1g -l 1 -u 1
Output is in kBytes/sec
Time Resolution = 0.000001 seconds.
Processor cache size set to 20480 kBytes.
Processor cache line size set to 32 bytes.
File stride size set to 17 * record size.
Min process = 1
Max process = 1
Throughput test with 1 process
Each process writes a 1048576 kByte file in 4 kByte records

Children see throughput for  1 initial writers  =  378709.97 kB/sec
Parent sees throughput for  1 initial writers  =  175685.84 kB/sec
Min throughput per process    =  378709.97 kB/sec
Max throughput per process    =  378709.97 kB/sec
```

```
 Avg throughput per process    =  378709.97 kB/sec
 Min xfer      = 1048576.00 kB
 CPU Utilization: Wall time    2.769    CPU time    2.769    CPU utilization 100.00 %


 Children see throughput for  1 rewriters  =  387702.94 kB/sec
 Parent sees throughput for  1 rewriters  =  190278.17 kB/sec
 Min throughput per process    =  387702.94 kB/sec
 Max throughput per process    =  387702.94 kB/sec
 Avg throughput per process    =  387702.94 kB/sec
 Min xfer      = 1048576.00 kB
 CPU utilization: Wall time    2.705    CPU time    2.705    CPU utilization 100.00 %


 Children see throughput for  1 readers   = 1027197.94 kB/sec
 Parent sees throughput for  1 readers   = 1024951.81 kB/sec
 Min throughput per process    = 1027197.94 kB/sec
 Max throughput per process    = 1027197.94 kB/sec
 Avg throughput per process    = 1027197.94 kB/sec
 Min xfer      = 1048576.00 kB
 CPU utilization: Wall time    1.021    CPU time    1.021    CPU utilization 100.00 %


 Children see throughput for 1 re-readers  =  763717.62 kB/sec
 Parent sees throughput for 1 re-readers  =  762507.54 kB/sec
 Min throughput per process    =  763717.62 kB/sec
 Max throughput per process    =  763717.62 kB/sec
 Avg throughput per process    =  763717.62 kB/sec
 Min xfer      = 1048576.00 kB
 CPU utilization: Wall time    1.373    CPU time    1.373    CPU utilization 100.00 %




iozone test complete.
#opp      Total T. elapsed No fdsynk T. elapsed       T. Wr. KB       T. Rd. KB    #↩
    fdsync     #lseek
3                  996517               994517               0          1048600 ↩
             1          0
4                  838613               838581               0          1048592 ↩
             1          0
2                 4945363              2141110         1050854               72 ↩
             1          0
1                 5399966              2202638         1050964               88 ↩
             1          0


 Lseek offset analysis:
```

Dedique especial atenção ao final do ficheiro de output:

```
#opp  Total T. elapsed   No fdsync T. elapsed WR KB     RD KB #fdsync   #lseek
3         996517              994517 0              1048600       1       0
4         838613              838581 0              1048592       1       0
2         4945363              2141110 1050854           72       1       0
1         5399966              2202638 1050964           88       1       0
```

Denote que para os casos específicos de leitura e re-leitura, a diferença entre tempos com ou sem a inclusão no cálculo da syscall **fdsync** não tão significância. Uma vez mais acreditamos que estes valores provam a não inclusão da syscall **fdsync** no cálculo dos tempos totais.

1.5.3 Operações de Escrita e Leitura com inclusão de posições aleatórias no ficheiro

Verifiquemos a qualidade da medição da ferramenta para operações de leitura e escrita, agora com a inclusão de aleatoriedade na posição de leitura/escrita do ficheiro. Nestes casos, não nos interessa apenas que o tempo total medido via benchmarking activo seja aproximado da medição passiva, mas também verificar que existe uma distribuição ponderada por todo o ficheiro no acesso aleatório.

Atente no ficheiro de output resultante:

```
 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: Tue Apr 26 06:18:01 2016

 CPU utilization Resolution = 0.000 seconds.
 CPU utilization Excel chart enabled
 File size set to 1048576 kB
 Command line used: /opt/csw/bin/iozone -+u -i 0 -i 2 -S 20480 -s 1g -l 1 -u 1
 Output is in kBytes/sec
 Time Resolution = 0.000001 seconds.
 Processor cache size set to 20480 kBytes.
 Processor cache line size set to 32 bytes.
 File stride size set to 17 * record size.
 Min process = 1
 Max process = 1
 Throughput test with 1 process
 Each process writes a 1048576 kByte file in 4 kByte records

 Children see throughput for  1 initial writers  =  384791.19 kB/sec
 Parent sees throughput for  1 initial writers  =  178575.15 kB/sec
 Min throughput per process    =  384791.19 kB/sec
 Max throughput per process    =  384791.19 kB/sec
 Avg throughput per process    =  384791.19 kB/sec
 Min xfer      = 1048576.00 kB
 CPU Utilization: Wall time    2.725    CPU time    2.725    CPU utilization  99.99 %


 Children see throughput for  1 rewriters  =  381352.62 kB/sec
 Parent sees throughput for  1 rewriters  =  198996.56 kB/sec
 Min throughput per process    =  381352.62 kB/sec
 Max throughput per process    =  381352.62 kB/sec
 Avg throughput per process    =  381352.62 kB/sec
 Min xfer      = 1048576.00 kB
 CPU utilization: Wall time    2.750    CPU time    2.750    CPU utilization 100.00 %


 Children see throughput for 1 random readers  =  483253.06 kB/sec
 Parent sees throughput for 1 random readers  =  482545.45 kB/sec
 Min throughput per process    =  483253.06 kB/sec
 Max throughput per process    =  483253.06 kB/sec
 Avg throughput per process    =  483253.06 kB/sec
 Min xfer      = 1048576.00 kB
 CPU utilization: Wall time    2.170    CPU time    2.170    CPU utilization 100.00 %


 Children see throughput for 1 random writers  =  293867.94 kB/sec
 Parent sees throughput for 1 random writers  =  127562.14 kB/sec
 Min throughput per process    =  293867.94 kB/sec
 Max throughput per process    =  293867.94 kB/sec
 Avg throughput per process    =  293867.94 kB/sec
 Min xfer      = 1048576.00 kB
 CPU utilization: Wall time    3.568    CPU time    3.568    CPU utilization 100.00 %




iozone test complete.
#opp      Total T. elapsed No fdsynk T. elapsed       T. Wr. KB       T. Rd. KB    #↩
    fdsync     #lseek
3                  1111365              1111333               0          1048600 ↩
             1     262144
2                  4697017              2179381         1050956              80 ↩
             1           0
1                  5285556              2141299         1051653              88 ↩
             1           0
4                  7105555              2457011         1051410             120 ↩
             1     262144


 Lseek offset analysis:




       3
          value  ------------- Distribution ------------- count
             -1 |                                        0
              0 |                                        1
```

```
                1 |                                                    0 |
                2 |                                                    0 |
                4 |                                                    0 |
                8 |                                                    0 |
               16 |                                                    0 |
               32 |                                                    0 |
               64 |                                                    0 |
              128 |                                                    0 |
              256 |                                                    0 |
              512 |                                                    0 |
             1024 |                                                    0 |
             2048 |                                                    0 |
             4096 |                                                    1 |
             8192 |                                                    2 |
            16384 |                                                    4 |
            32768 |                                                    8 |
            65536 |                                                   16 |
           131072 |                                                   32 |
           262144 |                                                   64 |
           524288 |                                                  128 |
          1048576 |                                                  256 |
          2097152 |                                                  512 |
          4194304 |                                                 1024 |
          8388608 |                                                 2048 |
         16777216 |@                                                4096 |
         33554432 |@                                                8192 |
         67108864 |@@@                                             16384 |
        134217728 |@@@@@                                           32768 |
        268435456 |@@@@@@@@@@                                      65536 |
        536870912 |@@@@@@@@@@@@@@@@@@@@                           131072 |
       1073741824 |                                                    0 |

  4

      value  ------------- Distribution ------------- count
         -1 |                                                    0
          0 |                                                    1
          1 |                                                    0
          2 |                                                    0
          4 |                                                    0
          8 |                                                    0
         16 |                                                    0
         32 |                                                    0
         64 |                                                    0
        128 |                                                    0
        256 |                                                    0
        512 |                                                    0
       1024 |                                                    0
       2048 |                                                    0
       4096 |                                                    1
       8192 |                                                    2
      16384 |                                                    4
      32768 |                                                    8
      65536 |                                                   16
     131072 |                                                   32
     262144 |                                                   64
     524288 |                                                  128
    1048576 |                                                  256
    2097152 |                                                  512
    4194304 |                                                 1024
    8388608 |                                                 2048
   16777216 |@                                                4096
   33554432 |@                                                8192
   67108864 |@@@                                             16384
  134217728 |@@@@@                                           32768
  268435456 |@@@@@@@@@@                                      65536
  536870912 |@@@@@@@@@@@@@@@@@@@@                           131072
 1073741824 |                                                    0
```

Dedique especial atenção ao final do ficheiro de output:

```
#opp Total T. elapsed  No fdsync T. elapsed WR KB    RD KB #fdsync  #lseek
3       1111365       1111333 0               1048600       1       262144
2       4697017        2179381 1050956            80         1       0
1       5285556        2141299 1051653            88         1       0
4       7105555       2457011 1051410       1048592         1       262144
```

Tal como se verificou para os casos específicos de leitura e re-leitura, a diferença entre tempos com ou sem a inclusão no cálculo da syscall **fdsync** não tem a significância que possui para os casos de escrita e re-escrita. Um aspecto interessante a observar prende-se com a distribuição da posição de leitura/escrita aleatória. Como podemos constatar pelo output do programa a leitura e escrita aleatórias apresentam o mesmo número de chamadas da syscall **lseek**. Podemos ainda verificar que

a distribuição está normalizada, uma que quanto maior é o intervalo de offset à posição inicial do ficheiro, maior é o numero de chamadas da syscall **lseek**, dado o tamanho do intervalo aumentar na mesma proporção.

Resta-nos analisar se o comportamento analisado nesta seção (e consequentemente filesystem ZFS) se verifica para os outros filesystems que temos disponíveis no sistema de computação.

1.6 Take 5 – uma análise com dtrace aos filesystems NFS e UFS

1.6.1 filesystem NFS

Até ao momento apenas trabalhamos com o filesystem ZFS (dado estarmos a trabalhar na home). Ora, será interessante analisar o comportamento da mesma benchmark para filesystems distintos. Analisemos portanto o comportamento da ferramenta iozone no filesystem NFS com as seguintes opções:

```
/opt/csw/bin/iozone -+u -i 0 -S 20480 -s 1g -l 1 -u 1 -F /share/jade/a57816/iozone.DUMMY.0
```

```
 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: Tue Apr 26 06:53:15 2016

 CPU utilization Resolution = 0.000 seconds.
 CPU utilization Excel chart enabled
 File size set to 1048576 kB
 Command line used: /opt/csw/bin/iozone -+u -i 0 -S 20480 -s 1g -l 1 -u 1 -F /share/jade/a57816/iozone.DUMMY.0
 Output is in kBytes/sec
 Time Resolution = 0.000001 seconds.
 Processor cache size set to 20480 kBytes.
 Processor cache line size set to 32 bytes.
 File stride size set to 17 * record size.
 Min process = 1
 Max process = 1
 Throughput test with 1 process
 Each process writes a 1048576 kByte file in 4 kByte records

 Children see throughput for  1 initial writers  =  113067.48 kB/sec
 Parent sees throughput for  1 initial writers  =   97872.50 kB/sec
 Min throughput per process    =  113067.48 kB/sec
 Max throughput per process    =  113067.48 kB/sec
 Avg throughput per process    =  113067.48 kB/sec
 Min xfer      = 1048576.00 kB
 CPU Utilization: Wall time    9.274    CPU time    3.210    CPU utilization  34.61 %


 Children see throughput for  1 rewriters  =  113052.21 kB/sec
 Parent sees throughput for  1 rewriters  =   96989.31 kB/sec
 Min throughput per process    =  113052.21 kB/sec
 Max throughput per process    =  113052.21 kB/sec
 Avg throughput per process    =  113052.21 kB/sec
 Min xfer      = 1048576.00 kB
 CPU utilization: Wall time    9.275    CPU time    3.114    CPU utilization  33.58 %




iozone test complete.
#opp      Total T. elapsed No fdsynk T. elapsed       T. Wr. KB       T. Rd. KB    #↩
    fdsync     #lseek
2                 9917653             84572075          1056978              32 ↩
             0          0
1                10835306             94914415          1056978              40 ↩
             0          0
```

```
Lseek offset analysis:
```

Dedique especial atenção ao final do ficheiro de output:

```
#opp       Total T. elapsed No fdsynk T. elapsed       T. Wr. KB       T. Rd. KB     #↩
    fdsync      #lseek
2                  9917653             84572075          1056978              32 ↩
              0           0
1                 10835306             94914415          1056978              40 ↩
              0           0
```

Ora, para o caso do filesystem UFS podemos considerar a precisão da ferramenta vs ZFS é superior. Tal será certamente devido à complexidade acrescida e otimização presentes no filesystem ZFS.

1.6.2 filesystem UFS

Realizemos o mesmo teste agora para o filesystem UFS. Denote que, este último filesystem lida com uma tecnologia de armazenamento física diferente das até agora utilizadas. Iremos realizar o teste para o path /diskHitachi/a57816/, montado num disco HDD.

Analisemos portanto o comportamento da ferramenta iozone no filesystem UFS com as seguintes opções:

```
/opt/csw/bin/iozone -+u -i 0 -S 20480 -s 1g -l 1 -u 1 -F /diskHitachi/↩
    a57816/iozone.DUMMY.0
```

```
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: Tue Apr 26 07:01:27 2016

CPU utilization Resolution = 0.000 seconds.
CPU utilization Excel chart enabled
File size set to 1048576 kB
Command line used: /opt/csw/bin/iozone -+u -i 0 -S 20480 -s 1g -l 1 -u 1 -F /↩
    diskHitachi/a57816/iozone.DUMMY.0
Output is in kBytes/sec
Time Resolution = 0.000001 seconds.
Processor cache size set to 20480 kBytes.
Processor cache line size set to 32 bytes.
File stride size set to 17 * record size.
Min process = 1
Max process = 1
Throughput test with 1 process
Each process writes a 1048576 kByte file in 4 kByte records

Children see throughput for  1 initial writers  =   69354.38 kB/sec
Parent sees throughput for  1 initial writers  =   68516.99 kB/sec
Min throughput per process     =   69354.38 kB/sec
Max throughput per process     =   69354.38 kB/sec
Avg throughput per process     =   69354.38 kB/sec
Min xfer      = 1048576.00 kB
CPU Utilization: Wall time   15.119     CPU time    5.233     CPU utilization   34.61 %


Children see throughput for  1 rewriters  =   73200.20 kB/sec
Parent sees throughput for  1 rewriters  =   72431.44 kB/sec
Min throughput per process     =   73200.20 kB/sec
Max throughput per process     =   73200.20 kB/sec
Avg throughput per process     =   73200.20 kB/sec
Min xfer      = 1048576.00 kB
CPU utilization: Wall time   14.325     CPU time    3.488     CPU utilization   24.35 %
```

```
                                                                                50
                                                                                51
iozone test complete.                                                           52
#opp       Total T. elapsed No fdsynk T. elapsed       T. Wr. KB       T. Rd. KB    #↩  53
    fdsync      #lseek
2                   15811178             15575780          1056649             236 ↩   54
              1           0
1                   14608933             14426632          1057084             224 ↩   55
              1           0
                                                                                56
                                                                                57
 Lseek offset analysis:                                                         58
```

Dedique especial atenção ao final do ficheiro de output:

```
#opp       Total T. elapsed No fdsynk T. elapsed       T. Wr. KB       T. Rd. KB    #↩  1
    fdsync      #lseek
2                   15811178             15575780          1056649             236 ↩   2
              1           0
1                   14608933             14426632          1057084             224 ↩   3
              1           0
```

Tal como se verificou para o sistema de ficheiros NFS, também a ferramenta iozone para o sistema de ficheiros UFS apresenta valores de benchmarking passivos muito próximos dos obtidos via benchmarking activa. Daqui concluímos que quanto mais otimizado e complexo o sistema de ficheiros, maior será a dificuldade em inferir testes de benchmarking que reflitam situações de interesse para o utilizador.

Apesar de não ser o propósito do trabalho prático, constatamos que a tecnologia SSD, tal como esperado, permite obter ordens de grandeza superiores no que toca a capacidade máxima de escrita/leitura.

2 Conclusão e Trabalho Futuro

> Running a benchmark is the easy part. Understanding a benchmark can take much longer.
>
> `http://www.brendangregg.com/ActiveBenchmarking/bonnie++.html`

Tal como mencionado em trabalhos anteriores a ferramenta DTrace mostra-se bastante útil e única em termos de funcionalidades quando necessitamos de agregar informação de vários processos/threads,etc. Contudo, esta representa uma pequena porção de todo o trabalho de análise e tratamento de dados necessário para a validação de valores apresentados por ferramentas de benchmarking como o caso da iozone. Sem a análise anterior via **iostat** e **truss** seria impossível tirar qualquer tipo de conclusão.

Ora, o trabalho futuro prender-se-ia com o teste de outras funcionalidades da ferramenta iozone corroborando/desmentindo os valores nela anunciados. Por vezes, durante o processo de compreensão da ferramenta iozone foi necessário analisar instrução-a-instrução resultante, com algumas descobertas interessantes.

Retratamos sobretudo a capacidade analisar funcionalidades disponibilizadas e a sua correta aplicação na resolução de problemas de computação tendo sempre em conta o mínimo de alteração possível na medição de performance do sistema analisado.