Benchmark Ativo com IOzone
- Pré-Relatório TPC4 -

Carlos Sá - A59905
carlos.sa01@gmail.com

April 18, 2016

Abstract

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

Sendo este trabalho um trabalho de Benchmark Ativo, pretende-se no final do mesmo, confrontar o resultado do benchmark feito pelo IOzone com a tentativa de replicação desse benchmark recorrendo às restantes ferramentas anteriormente descritas e atestar a veracidade dos resultados devolvidos pelo IOzone. Para esta fase preliminar do trabalho apenas se pretende explorar a ferramenta IOZone e as restantes ferramentas de com as diferentes flags disponíveis por forma a perceber que tipo de profilling de input/output poderá ser realizado.

1. Introdução ao Benchmark Ativo

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

- Benchmarking Ativo;
- Benchmarking Passivo.

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

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

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

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

Neste estudo utilizarei a ferramenta IOZone como ferramenta principal de benchmark, e tentarei fazer benchmark ativo utilizando ferramentas auxiliares como **dtrace**, **strace**, **truss** etc.

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

2. A ferramenta IOzone

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

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

E através do comando:

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

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

```

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

(....)

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

A ferramenta IOZone será utilizada como ferramenta principal de benchmarking e disponibiliza ao utilizador um vasto conjunto de testes que podem ser passados ao programa de teste através de flags de acordo com os testes que se pretendem estudar. Todos os testes estão relacionados sobretudo com testes de performance de operações de input/output tais como leituras e escritas de ficheiros de/em disco. Assim, a primeira coisa que fiz foi estudar o conjunto de testes disponíveis pelo IOZone e escolher aqueles

que irei utilizar no meu estudo. Através da consulta da *man page* do IOzone de entre o vasto conjunto de testes disponíveis, irei concentrar o meu estudo da ferramenta nos seguintes:

Flags/Testes	Descrição do teste
-a	Utiliza um modo automático e produz resultados que cobre todos os testes para tamanhos de record de 4K a 16M para ficheiros de 64K a 512M
-i0	Para realizar testes sobre a performance de escrita (write) em ficheiro e de re-escrita em ficheiros que já existam. Nota para o facto que uma escrita em ficheiro não contempla apenas a escrita dos dados do ficheiro em si mas também do seu meta-dados (metadata) para ser possível localizar o ficheiro no disco.
-i1	Para realizar testes sobre a performance de leitura (read) de ficheiros já existentes e de ficheiros lidos recentemente (re-read).
-i2	Semelhante aos testes read/write, mas agora com as leituras e escritas dos dados do ficheiro a feitas de/para localizações aleatórias do disco. Neste teste vários factores são tidos em conta e que influenciam os resultados finais: número de discos, latências de acesso a disco, tamanho da cache etc. Ao mesmo tempo permite atenuar os resultados de leituras e escritas de dados de ficheiros que estejam em posições consecutivas no disco.
-i3	Utilizada para medir a performance das leituras feitas em sentido inverso (backwards read).
-B	Faz utilização de ficheiros mmap() que faz com que sejam acedidos apenas os ficheiros de loads e stores que provocam I/O
-G	Forçar a escrita de ficheiros mmap no disco de forma síncrona.
-D	Forçar a escrita de ficheiros mmap no disco de forma asincrona.
-I	Garante que são utilizadas apenas operações de I/O diretas no disco (sem utilizar operações de I/O de buffers ou cache).

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

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

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

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

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

4. Testes com IOzone e experimentação

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

4.1. Teste iozone automatico

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

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

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

4.2. Testes de Escrita: write e re-write

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

Para realizar o teste de escrita executei:

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

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

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

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

4.3. Testes de Leitura: Read e re-read

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

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

kB	reclen	read	reread	(....)
64	4	1734015	1828508	
64	8	2467108	2561267	
64	16	2662899	2772930	
64	32	2801873	2923952	
64	64	3363612	3363612	
128	4	3359523	4135958	
128	8	2058509	2608629	
128	16	4717434	6406138	
128	32	4889281	6727225	
128	64	2843510	3297617	
128	128	3199360	3199360	
256	4	3236056	4181144	
256	8	4054829	5810112	
256	16	2266207	2812272	
256	32	2533571	3123106	
256	64	2613746	3326279	
256	128	2754556	3236056	
256	256	1445521	1479379	
512	4	1667552	1933266	
512	8	4228947	5509113	
512	16	2178404	2782414	
512	32	4871723	6736026^C	

5. Testes com truss

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

A titulo de exemplo, ao executar na shell:

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

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

Assim, podemos destacar algum output relevante tal como:

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

(.....)

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

(...)

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

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

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

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

6. Notas finais sobre a experimentação realizada

Ao longo de todo este trabalho inicial foi feito uma tentativa de abordar os diferentes testes existentes do IOzone. Ao longo do meu estudo com a ferramenta várias flags foram exploradas juntamente com diferentes configurações com o intuito de perceber a potencialidade do IOzone.

Claro que a nível de exposição dos resultados apenas incluí neste relatório preliminar, testes pequenos e relativamente simples de documentar apenas para explicar os conceitos e não tornar o relatório demasiado exaustivo. Tentei explorar os testes com a maior parte das flags existentes no documento de leitura recomendada sobre IOZone [1] e decidi que irei focar o trabalho nos testes que englobam *reads* e *writes*. E, desta forma, tentarei criar scripts dtrace que façam igualmente monitorização deste tipo de operações. Na componente experimental, depois de criadas as scripts dtrace para fazer benchmarking activo espero poder testar com ficheiros de dimensões maiores (entre 8 e 16GB).

7. Trabalho Futuro

Ao longo deste pré-relatório foi feita uma pequena abordagem sobre como irei explorar benchmarking ativo e replicar os resultados obtidos pelo IOzone com ferramentas auxiliares como **dtrace**, e **truss**. Nesta fase do trabalho a principal preocupação foi familiarizar-me com as ferramentas, e perceber como irei conduzir o estudo que irei realizar. Todas as ferramentas foram exploradas e os resultados avaliados com diferentes flags por forma a realizar diferentes testes. De um modo geral percebi que terei que utilizar a ferramenta **dtrace** para criar um conjunto de scripts que permitam replicar os testes que o IOZone realiza e comparar os resultados. Percebi que a ferramenta **truss** e **strace** podem dar um excelente contributo nesta tarefa de replicação pois permitem saber as system calls invocadas quando um determinado teste é feito o que certamente ajudará a produzir uma réplica do respectivo teste com scripts em **dtrace**. Igualmente a ferramenta **ltrace** poderá ser utilizada para perceber quais as **linked libraries** que são invocadas durante a execução de um teste, mas aínda tenho que perceber melhor como tirar partido dessa conveniência.

De uma forma geral, eu resumiria este primeiro relatório à documentação de uma fase de exploração das ferramentas de trabalho por forma a ser possível numa fase posterior do trabalho, executar a réplica dos testes feitos com o IOzone.

Numa fase posterior deste trabalho o objectivo será criar scripts **dtrace** que permitam replicar os testes feitos pelo IOZone e tentar encontrar divergências nos resultados. A esta tarefa de contra-análise ativa de resultados obtidos por uma ferramenta de benchmark com recurso a uma segunda ferramenta de benchmark chamamos **benchmark activo** e que será explorado com mais detalhe no próximo trabalho.

References

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

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