Università degli Studi di
Napoli Federico II

Dipartimento di Ingegneria Elettrica e delle
Tecnologie dell'Informazione

Impianti di Elaborazione

`Elaborato`

Michele Pommella
Davide Trimaldi

Prof. Domenico Cotroneo

Anno 2018-2019

Indice

Elenco delle figure

Capitolo 1

Benchmark

Il primo elaborato consta di un *Linux I/O benchmark* per le operazioni di lettura e scrittura su di un file binario. Le dimensioni di file valutate sono:

- $10MB$
- $100MB$
- $1GB$

Le operazioni di I/O avvengono a blocchi di dimensione:

- $1KB$
- $10KB$
- $100KB$
- $1MB$

Il sistema utilizzato è composto da:

- processore `Intel Celeron N3050` $1.60GHz$
- memoria RAM DDR3 $4GB$
- disco `HGST HTS545050A7` $500GB$

Sono stati effettuati 30 esperimenti per ogni configurazione attraverso uno script *bash*. Prima di ogni esperimento sono stati utilizzati i comandi *sync* ed *echo* $1 > /proc/sys/vm/drop\_caches$ per ottenere l'indipendenza tra essi (purge della cache).

Si è sfruttata la direttiva `O_DIRECT` all'apertura dei file, per minimizzare l'effetto della cache nelle operazioni di I/O da e verso il file. Ciò ha richiesto

l'utilizzo di blocchi allineati, multipli di 512 byte, la cui creazione è stata demandata alla funzione *posix_memalign*. Il calcolo del tempo è effettuato con *gettimeofday*, ed è attuato in microsecondi.

1.1 Lettura

Per effettuare le operazioni di lettura, si è sfruttato un unico file *prova.bin* di $1GB$, creato in precedenza. Si sfruttano le direttive della primitiva *open* che consentono la lettura del file e la lettura diretta dallo storage.

```
#define _GNU_SOURCE
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <unistd.h>
#include <sys/time.h>
#include <stdlib.h>
#include <time.h>
#include <stdio.h>
#include <string.h>

void main(int argc, char* argv[]){

        int fd = open("prova.bin", O_RDONLY|O_DIRECT);

        if(fd != -1){
                size_t FILESIZE = strtoul(argv[1], NULL, 0);
                size_t BLOCKSIZE = strtoul(argv[2], NULL, 0);
                BLOCKSIZE -= (BLOCKSIZE%512);

                void* buffer;
                stato = posix_memalign(&buffer, 512, BLOCKSIZE);

                if(!stato){
                        size_t count = FILESIZE/BLOCKSIZE;
                        struct timeval inizio, fine;
                        gettimeofday(&inizio, NULL);
                        for(; count > 0; count--){
                                read(fd, buffer, BLOCKSIZE);
                        }
                        gettimeofday(&fine, NULL);
```

```
                        FILE* readtime;
                        char nome[50] = "readtime";
                        strcat(nome, argv[1]);
                        strcat(nome, "x");
                        strcat(nome, argv[2]);

                        readtime = fopen(nome, "a");
                        if(readtime){
                                fprintf(readtime, "%ld\n",
                                (long)fine.tv_sec*1000000+
                                (long)fine.tv_usec-
                                (long)inizio.tv_sec*1000000-
                                (long)inizio.tv_usec);
                                fclose(readtime);
                        }
                        else
                                perror(
                                "Salvataggio risultati non riuscito");
                        free(buffer);
                }
                else
                        perror("Errore nell'allocazione del buffer");
                close(fd);
        }
        else
                perror("Errore nell'apertura del file");
}
```

1.2 Scrittura

Le operazioni di scrittura sono effettuate sul file *test.bin*, che viene creato, se già esistente, o altrimenti sovrascritto. Ciò è definito dalle direttive della primitiva *open*, che inoltre indicano l'apertura in scrittura del file, da attuare direttamente verso lo storage. Nel caso di creazione del file, si indicano anche i permessi che esso dovrà avere. L'operazione utilizza un blocco allineato, inizializzato con numeri psudocasuali.

```
#define _GNU_SOURCE
#include <sys/types.h>
```

```
#include <sys/stat.h>
#include <fcntl.h>
#include <unistd.h>
#include <sys/time.h>
#include <stdlib.h>
#include <time.h>
#include <stdio.h>
#include <string.h>

void main(int argc, char* argv[]){

        int fd = open("test.bin", O_CREAT|O_TRUNC|O_WRONLY|
        O_DIRECT,S_IRWXU);

        if(fd != -1){
                size_t FILESIZE = strtoul(argv[1], NULL, 0);
                size_t BLOCKSIZE = strtoul(argv[2], NULL, 0);
                BLOCKSIZE -= (BLOCKSIZE%512);

                void* buffer;
                int* temp;
                int stato;
                stato = posix_memalign(&buffer, 512, BLOCKSIZE);

                if(!stato){
                        temp = buffer;
                        srand(time(NULL));
                        for(int i = 0; i<BLOCKSIZE/sizeof(int);
                        i++){
                                temp[i] = rand();
                        }

                        size_t count = FILESIZE/BLOCKSIZE;
                        struct timeval inizio, fine;
                        temp = buffer;
                        gettimeofday(&inizio, NULL);
                        for(; count > 0; count--){
                                write(fd, temp, BLOCKSIZE);
                        }
                        gettimeofday(&fine, NULL);
```

```
                        FILE* writetime;
                        char nome[50] = "writetime";
                        strcat(nome, argv[1]);
                        strcat(nome, "x");
                        strcat(nome, argv[2]);

                        writetime = fopen(nome, "a");
                        if(writetime){
                                fprintf(writetime, "%ld\n",
                                (long)fine.tv_sec*1000000+
                                (long)fine.tv_usec-
                                (long)inizio.tv_sec*1000000-
                                (long)inizio.tv_usec);
                                fclose(writetime);
                        }
                        else
                                perror(
                                "Salvataggio risultati non riuscito");
                        free(buffer);
                }
                else
                        perror("Errore nell'allocazione del buffer");
                close(fd);
        }
        else
                perror("Errore nell'apertura del file");
}
```

1.3 Caratterizzazione dei dati misurati

I dati sono stati riuniti in tabelle relative alla stessa dimensione di file ed analizzati tramite *JMP*. Si è studiata la distribuzione degli esperimenti di una stessa configurazione, tramite istogramma, e si è valutato il modo migliore per sintetizzare i dati. In particolare, per distribuzioni poco *skewed*, è stata usata la media, la deviazione standard, e l'intervallo di confidenza della media al 95%.

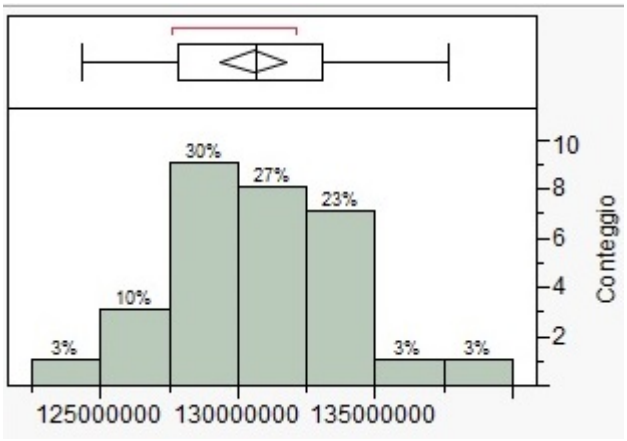

Figura 1.1: Scrittura di un file di $100MB$ con blocchi di $100KB$

In caso di distribuzioni skewed ed eventuali *outlier*, si è optato per la mediana ed il SIQR.

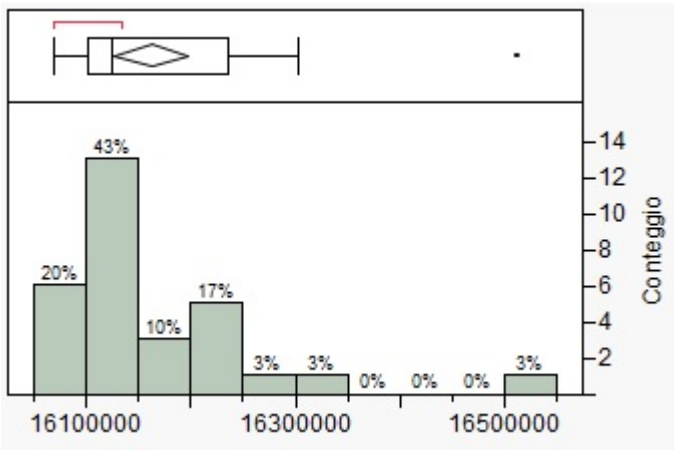

Figura 1.2: Lettura di un file di $1GB$ con blocchi di $1MB$

Sono stati quindi prodotti due data set, contenenti i dati delle operazioni di lettura e scrittura.

Operazione	File Size	Block Size	Media	Deviazione standard	Media superiore al 95%	Media inferiore al 95%	Mediana	SIQR
Lettura	10MB	1KB	934855,066666667	36649,5954608462	948540,250521448	921169,882811885	•	•
Lettura	10MB	10KB	•	•	•	•	205432	27438,125
Lettura	10MB	100KB	•	•	•	•	205094	3147,5
Lettura	10MB	1MB	•	•	•	•	194971,5	16620,75
Lettura	100MB	1KB	9465446,56666667	373670,936688463	9604977,58755428	9325915,54577905	•	•
Lettura	100MB	10KB	1612751,4	35485,9144368676	1626002,0582192	1599500,7417808	•	•
Lettura	100MB	100KB	•	•	•	•	1583719,5	11232,375
Lettura	100MB	1MB	1594768,06666667	17111,3834421927	1601157,5622524	1588378,57108093	•	•
Lettura	1GB	1KB	92558755,2	2361359,89626367	93440501,4763593	91677008,9236407	•	•
Lettura	1GB	10KB	16235173,5333333	171879,635371872	16299354,4439649	16170992,6227017	•	•
Lettura	1GB	100KB	•	•	•	•	16145116,5	97572
Lettura	1GB	1MB	•	•	•	•	16124616,5	66523,5

Figura 1.3: Dati delle letture

Operazione	File Size	Block Size	Media	Deviazione standard	Media superiore al 95%	Media inferiore al 95%	Mediana	SIQR
Scrittura	10MB	1KB	•	•	•	•	12311506,5	1148854,625
Scrittura	10MB	10KB	•	•	•	•	576896	64254,375
Scrittura	10MB	100KB	245443,56667	36271,4124	258987,53465	231899,59869	•	•
Scrittura	10MB	1MB	206389,96667	25445,214196	215891,3658	196888,56753	•	•
Scrittura	100MB	1KB	130562634,63	3173853,907	131747771,16	129377498,11	•	•
Scrittura	100MB	10KB	4389713,4667	406343,81457	4541444,7407	4237982,1927	•	•
Scrittura	100MB	100KB	1683369,5	68459,584308	1708932,7289	1657806,2711	•	•
Scrittura	100MB	1MB	1680462,5667	67880,361252	1705809,5101	1655115,6232	•	•
Scrittura	1GB	1KB	•	•	•	•	1307259333,5	11987577,125
Scrittura	1GB	10KB	•	•	•	•	43754224,5	1070926
Scrittura	1GB	100KB	20388083,133	1039264,2147	20776150,769	20000015,498	•	•
Scrittura	1GB	1MB	•	•	•	•	15432516,5	244812,375

Figura 1.4: Dati delle scritture

Si possono, inoltre, sfruttare i risultati ricavati per determinare la dimensione del campione necessaria per ottenere con accuratezza desiderata la media delle osservazioni con un certo livello di confidenza. Prendiamo in considerazione la scrittura di un file di $100MB$. I tempi medi stimati per effettuare l'operazione sono circa: $2.10min$ per blocchi di $1KB$, $4.4s$ per blocchi di $10KB$, $1.7s$ per blocchi di $100KB$, $1.7s$ per blocchi di $1MB$. Si può voler calcolare in questi casi un tempo accurato di $r = 0.5\%$ con livello

di confidenza del 95%. In questi casi, si ricaverebbe una media dei tempi accurata rispettivamente per al più di $6.5ds$, $22ms$, $8.4ms$, $8.4ms$. Si applica dunque:

$$n = \left(\frac{100zs}{r\bar{x}}\right)^2 \tag{1.1}$$

con $z = 1.96$, s deviazione standard, $\bar{x}$ media. Si ottiene, quindi, il numero di punti necessari: $n = 90$, $n = 1316$, $n = 254$, $n = 250$.

Capitolo 2

The Second Chapter