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%.

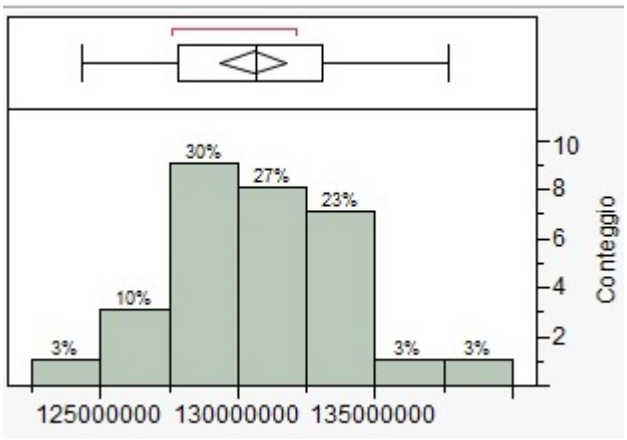

Figura 1.1: Scrittura di un file di $100MB$ con blocchi di $100KB$

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

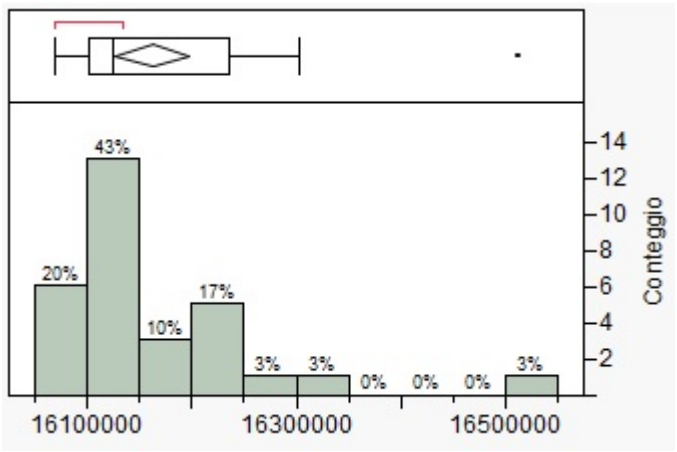

Figura 1.2: Lettura di un file di $1GB$ con blocchi di $1MB$

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

Dataset Reduction

2.1 Descrizione del problema

Lo scopo di questo elaborato è quello di analizzare un set di dati e ridurlo, in modo che sia comunque rappresentativo della popolazione da cui è stato estratto il campione.
Il dataset in questione contiene informazioni riguardanti parametri e valori di prestazioni di un file system di Unix, raccolte in 3000 istanze (righe) descritte da 24 feature (colonne).

L'obiettivo, quindi, sarà quello di selezionare un numero esiguo di righe, pur mantenendo la maggior percentuale di varianza e quindi di informazione possibile.
Per fare questo, dopo aver manipolato preliminarmente i dati, si sono utilizzate due tecniche: la **PCA** e il **Clustering**. Come software invece sono stati utilizzati JMP e Matlab.

2.2 Trattamento preliminare dei dati

Prima di utilizzare le tecniche sopracitate, è stata calcolata la varianza di ogni colonna con JMP e il risultato è stato il seguente:

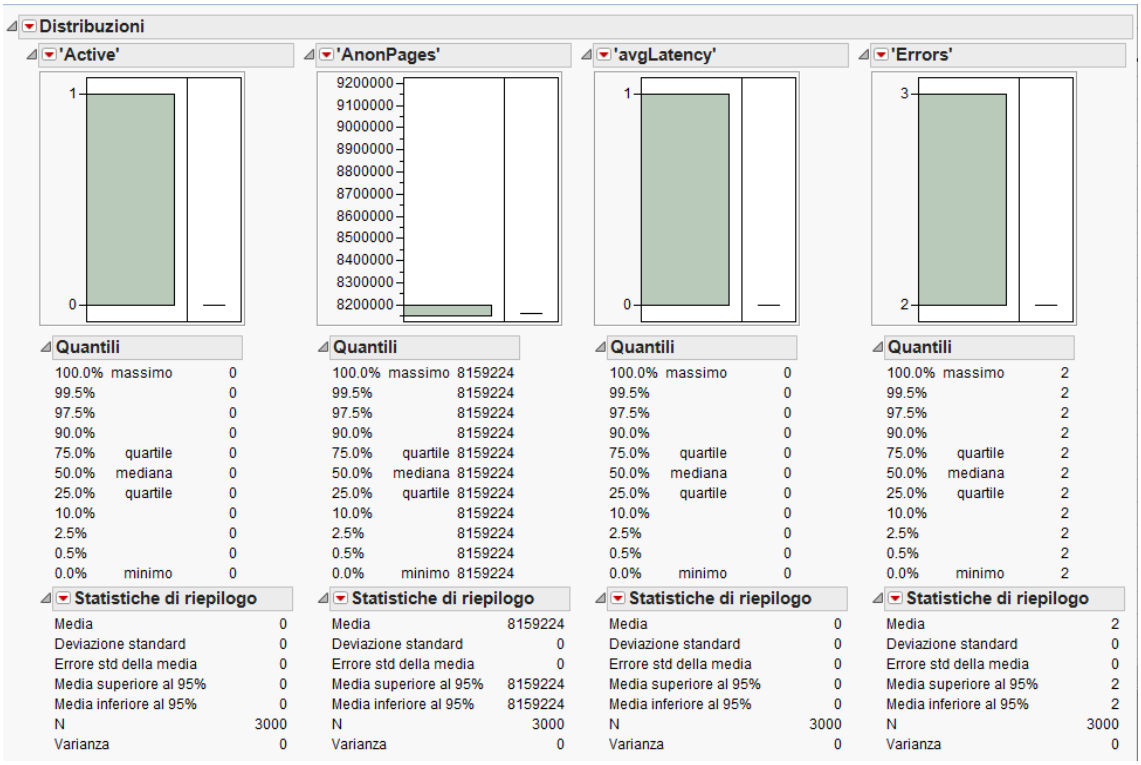

Figura 2.1: Distribuzione colonne costanti

Come si nota sia dagli istogrammi e dal valore in basso, la varianza di queste colonne è nulla, quindi vengono eliminate in quanto non apportano contenuto informativo all'analisi dei dati.

2.3 PCA

Si è utilizzata la Principal Component Analysis per compiere una trasformazione lineare delle variabili originarie (gli attributi del dataset), proiettandole in un nuovo sistema cartesiano, in modo che le nuove variabili ottenute, dette **componenti principali** spieghino la maggior parte della varianza di quelle originarie.

Autovalori

Numero	Autovalore	Percentuale	20 40 60 80	Percentuale cumulativa
1	10,4082	52,041		52,041
2	4,0947	20,474		72,514
3	2,4797	12,398		84,913
4	0,7810	3,905		88,818
5	0,7070	3,535		92,353
6	0,6253	3,126		95,479
7	0,3396	1,698		97,177
8	0,2657	1,329		98,506
9	0,1618	0,809		99,315
10	0,0906	0,453		99,767
11	0,0178	0,089		99,856
12	0,0114	0,057		99,913
13	0,0079	0,039		99,953
14	0,0059	0,029		99,982
15	0,0018	0,009		99,991
16	0,0011	0,006		99,997
17	0,0007	0,003		100,000
18	0,0000	0,000		100,000
19	0,0000	0,000		100,000
20	0,0000	0,000		100,000

Figura 2.2: Componenti principali e varianza conservata

Come si nota dalla figura esse sono ordinate in base a quanta varianza spiegano, quindi prendendo 6 componenti principali riesco a spiegare circa il 95% della varianza.

2.4 Clustering

Una volta ridotto il numero di feature a 6 attraverso la PCA, si è ridotto il numero di istanze attraverso la tecnica del clustering. Quello che si fa è, una volta definita una metrica di distanza, di unire in gruppi i dati che hanno distanza minima, ovvero quelli che sono più simili.
Come metodo di clustering è stato scelto quello di **Ward**, che è una tecnica gerarchica agglomerativa che consiste nel formare cluster unendo ad ogni iterazione una coppia di cluster con l'obiettivo di minimizzare la varianza intra-cluster e massimizzare quella inter-cluster.
Il processo di clustering porta alla realizzazione della seguente struttura gerarchica detta **dendrogramma**:

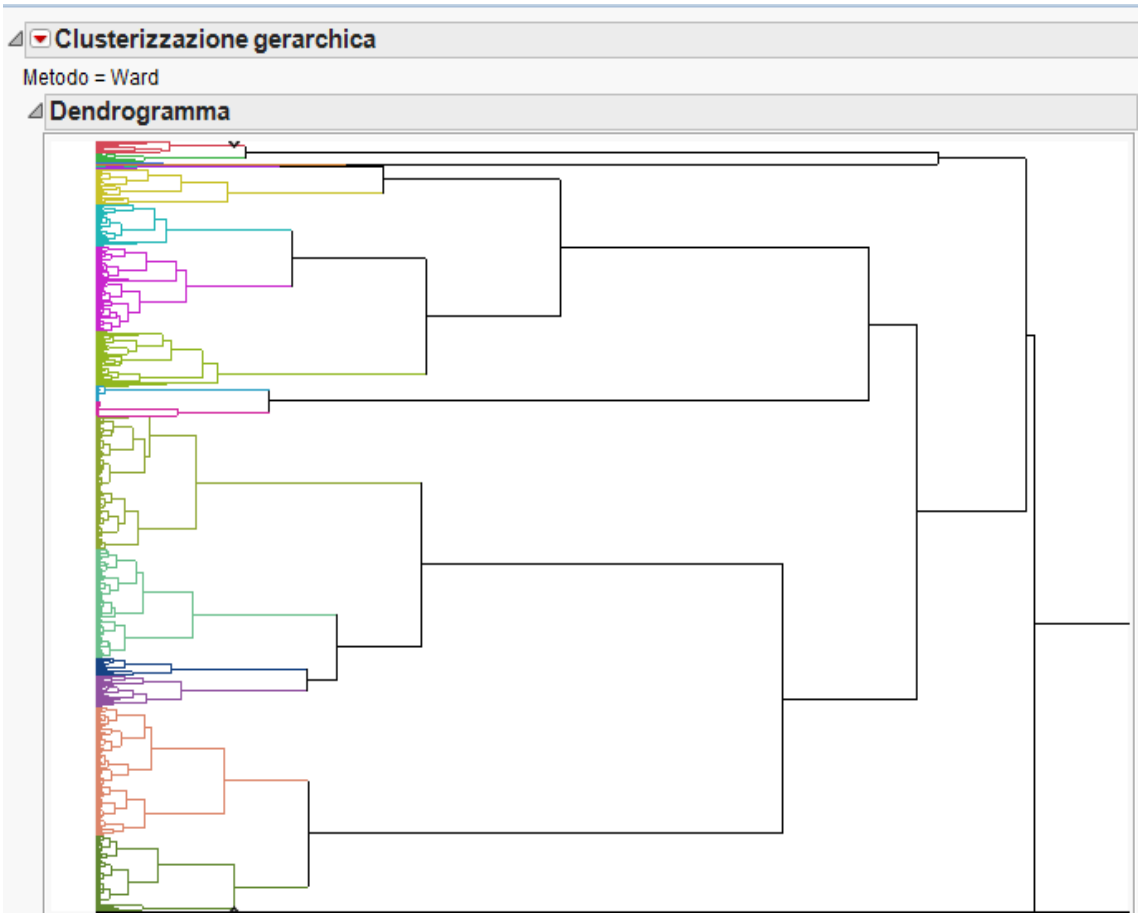

Figura 2.3: Dendrogramma

Come si nota alla radice sono raggruppati tutti i dati in un unico cluster mentre, scendendo verso il basso i dati vengono divisi in più cluster fino a giungere alle foglie, che non sono altro che cluster formate da un solo elemento. Più salgo verso la radice e più perdo varianza, più scendo verso le foglie e più la conservo, quindi bisogna trovare un trade-off tra la varianza persa e la riduzione del dataset.
La seguente immagine mostra la varianza persa a seconda del numero di cluster considerati:

Cronologia di clusterizzazione

Numero di cluster	Distanza	Leader	Subordinato
22	7,06859782	1873	2266
21	7,44782248	109	262
20	7,64403551	2737	2795
19	7,99161995	112	113
18	8,63159437	1	7
17	10,05365037	2885	2939
16	10,70073124	92	2844
15	11,35073498	1862	1931
14	12,22471456	193	1863
13	12,28100290	109	112
12	13,96390211	102	193
11	14,51362527	84	91
10	16,65821826	92	2737
9	18,83770694	96	102
8	19,14155949	1862	1873
7	26,93497818	92	1862
6	39,86801276	96	109
5	44,85223587	92	2885
4	47,69483696	92	96
3	48,86540849	1	84
2	53,98486129	1	92
1	54,51219296	1	90

Figura 2.4: Cluster con relativa varianza persa

In realtà si usa la devianza come metrica di distanza anzichè la varianza, in quanto vale la seguente relazione:

$$devianza_{totale} = devianza_{intracluster} + devianza_{intercluster}$$

Quindi nella tabella precedente, ogni distanza indica la devianza persa considerando quel dato numero di cluster e calcolando il rapporto $\frac{devianza_{intracluster}}{devianza_{totale}}$, sono in grado di conoscere la percentuale di devianza e quindi di varianza (a meno di una costante) persa a seguito dell'operazione di clustering.

Si è scelto 19 come numero di cluster, poiché, superata tale soglia, si va a perdere troppa devianza cosi come si nota dal seguente grafico, dove sull'asse delle ascisse è stato messo il numero di cluster e sull'asse delle ordinate la distanza.

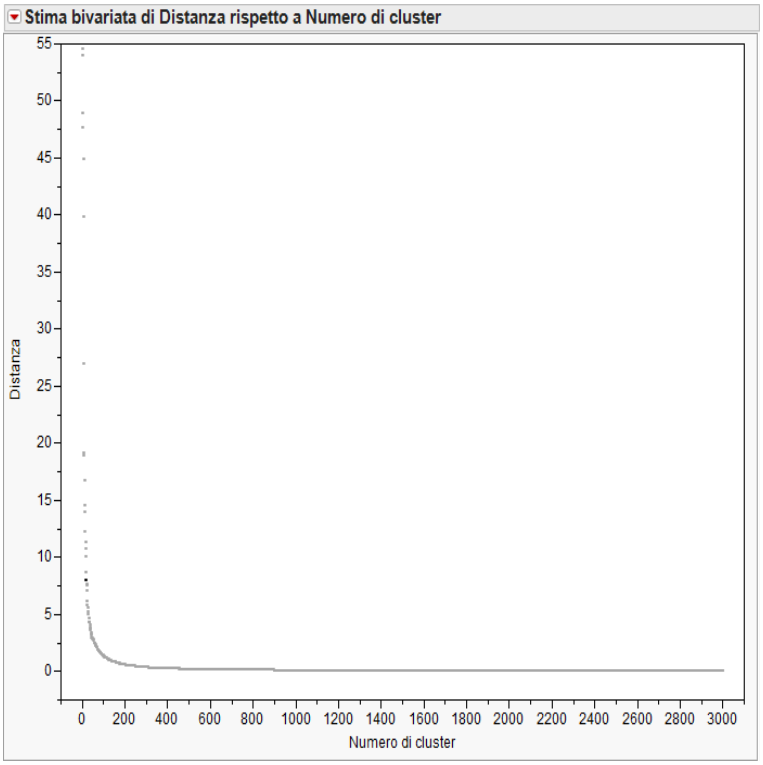

Figura 2.5: Grafico devianza persa al variare del numero di cluster

Scegliendo 19 cluster, quindi, perdiamo circa il 18,5% della varianza del dataset originale e ne conservo l'81,5%.

2.5 Conclusioni

In conclusione riportiamo il numero di dati che ogni cluster contiene con la relativa percentuale di copertura del dataset originario:

	Cluster	N. elem.	%
1	1	39	1,3
2	2	44	1,466666667
3	3	6	0,2
4	4	1	0,033333333
5	5	4	0,133333333
6	6	11	0,366666667
7	7	137	4,566666667
8	8	161	5,366666667
9	9	336	11,2
10	10	214	7,133333333
11	11	54	1,8
12	12	62	2,066666667
13	13	522	17,4
14	14	417	13,9
15	15	70	2,333333333
16	16	118	3,933333333
17	17	506	16,86666667
18	18	297	9,9
19	19	1	0,033333333

Figura 2.6: Numero di elementi per ogni cluster

Come si nota dalla tabella ci sono alcuni cluster che contengono pochi elementi, ad esempio il cluster 19 ne contiene solo 1 e questo corrisponde all'istanza che presenta l'unico valore diverso da zero dell'attributo *Slab*. Probabilmente in questi casi si tratta di **outlier**, cioè valori anomali, ma nonostante questo non sono stati eliminati in quanto potrebbero rappresentare un comportamento specifico del sistema preso in esame e quindi non avendo informazioni sulla loro significatività si è ritenuto opportuno mantenerli nel dataset.

Infine riportiamo i dati scelti in maniera casuale da ogni cluster (un elemento per ognuno), che sono rappresentativi del dataset originario.

	Principale1	Principale2	Principale3	Principale4	Principale5	Principale6	Cluster
1	-16,04303918	5,493733667	-4,693897776	-3,313022012	0,217077938	-0,279088379	1
2	-12,67665877	8,338050389	-2,004014509	0,271434213	0,490764516	0,577191122	2
3	-6,289476549	8,727554328	15,07971312	3,332650152	-6,212036061	-0,497157878	3
4	5,448282012	11,44660476	37,85694216	4,934142768	-14,53425854	-3,436110297	4
5	-1,824797051	3,356908876	11,11480145	1,921636674	-4,202721532	-1,663355562	5
6	8,026787137	5,025269291	0,691887655	1,766173222	-0,635533574	0,323947204	6
7	4,672870044	2,00448032	-0,417809686	1,236366608	0,573585979	1,89203745	7
8	0,273814086	-0,645675264	0,027288736	-0,848881994	-0,662951979	1,046405411	8
9	1,165552598	-0,134662731	-0,118516444	-0,567280251	-0,410479263	0,781090546	9
10	0,329047058	-0,580817249	-0,21301818	-1,854125224	-1,057808012	-0,00916855	10
11	6,111852851	2,978406231	-0,989535802	0,239979856	0,664729992	-2,406381038	11
12	5,766383687	2,710976449	-0,750252006	1,41693522	1,323999823	-2,148838839	12
13	-1,747440583	0,676807385	3,550064544	1,045887409	-1,069319404	-0,223383416	13
14	-2,157072757	-0,41909445	0,729120578	0,580443554	0,081865717	0,366655683	14
15	-2,281868835	-0,858390547	0,389214758	1,812819638	0,992021467	0,520360799	15
16	0,097334657	-0,890352506	0,241293401	0,201946114	-0,125710409	1,772805922	16
17	-1,919506059	-0,366314413	-0,111373652	-0,528029851	-0,259108322	-0,165437672	17
18	-2,036686644	-0,494480324	-0,078168859	-0,33288777	-0,031416297	-0,911123161	18
19	4,243816085	15,79861231	53,36914855	-20,98513048	28,89095084	3,032427145	19

Figura 2.7: Dati finali

Capitolo 3

Case study and experimental setup

L'elaborato è stato condotto sfruttando due sistemi differenti, aventi ruolo di *client* e *server*.

Il client ha un processore `Intel Celeron N3050` 1.60GHz, memoria di 4GB ed un sistema operativo `Ubuntu 18.04.1 LTS`.

Il server è costituito da una macchina virtuale con sistema operativo `Trisquel-mini 8.0`, distribuzione di *GNU* con kernel *Linux-libre*, memoria di 512GB, risiedente su un sistema con processore `Intel Core i3` 2.27GHz.

Il web server utilizzato è *Apache Web Server* versione 2, il load generator *Apache JMeter 5.0*. Esso consente l'invio di richieste HTTP al server con tasso impostabile. La scelta delle pagine su cui incentrare l'esperimento è stata dettata da una ricerca sul web delle dimensioni di quelle, a nostro parere, maggiormente rappresentative: social network, e-commerce, blog, siti aziendali, wiki. Infine abbiamo sfruttato per la scelta il report di *HTTP Archive* sullo stato del web. Esso, infatti, evidenzia come le pagine web, in contesto desktop e mobile, stiano aumentando di dimensioni, passando dalle centinaia di KB, al MB.

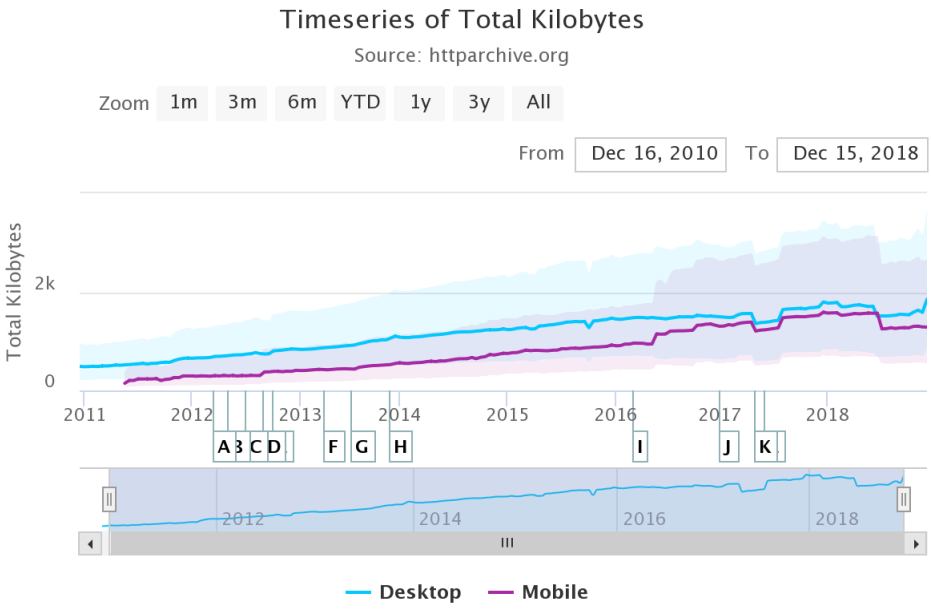

Figura 3.1: Stato del Web: Kilobyte totali

3.1 Workload Characterization

Mediante Jmeter, sono state inviate al server $60 req/s$ di 5 diversi tipi di pagine. Il tasso di richieste è stato impostato con il *Constant Throughput Timer*, selezionando $3600 req/min$, svolte dai thread complessivamente. Le richieste sono infatti eseguite da 30 thread per 5 minuti. Ogni richiesta può essere casualmente di uno dei 5 tipi:

- *Instagramlogin.html* di $32MB$
- *FrancoCFA.html* di $112KB$
- *Amazon.html* di $460KB$
- *Facebook.html* di $1.4MB$
- *Sample-jpg-image-5mb.jpg* di $1MB$

I dati di application level sono stati raccolti con un *Simple Data Writer*. Lato server i dati system level sono stati collezionati per 6 minuti, tramite il comando `vmstat -n -a 1 360`. Alcune istanze, quindi, tra questi sono precedenti e successive all'esperimento.

Tabella 3.1: Capacity Test per tipo di richiesta

Tipo	Usable Capacity	Knee Capacity
Random	120	50
Piccola	600	200
Media	220	88
Grande	80	20

3.2 Capacity Test

Il Capacity Test è stato eseguito con le pagine Amazon.html, Facebook.html, Sample-jpg-image-5mb.jpg, rappresentanti tipologie di pagine piccole, medie e grandi. Il test è stato attuato prima considerando tutte le pagine (*Random Controller*), poi ciascuna singolarmente.

Throughput e response time sono stati analizzati al crescere delle richieste al minuto. Diverse osservazioni sono state collezionate per una singola condizione di carico, caratterizzate, poi, dalla media, poiché c'è interesse nell'andamento globale.

Il tasso di richieste desiderato è stato ottenuto stabilendo il tasso per ciascun thread ed incrementando ogni volta il loro numero.

I risultati ottenuti sono riportati in **Tabella 3.2**. Di seguito l'andamento di throughput (1/min) e response time (ms), all'aumentare del carico nei 4 casi considerati.

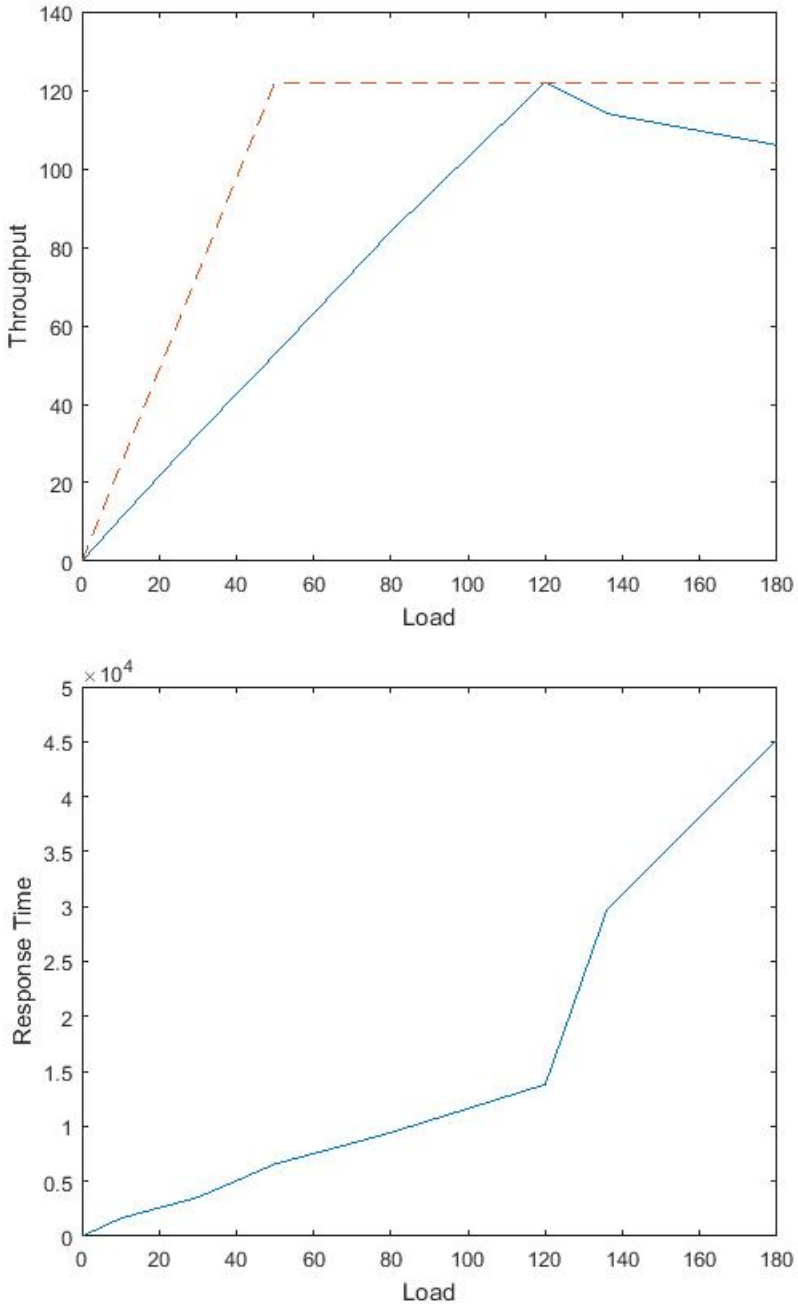

Figura 3.2: Pagine Random

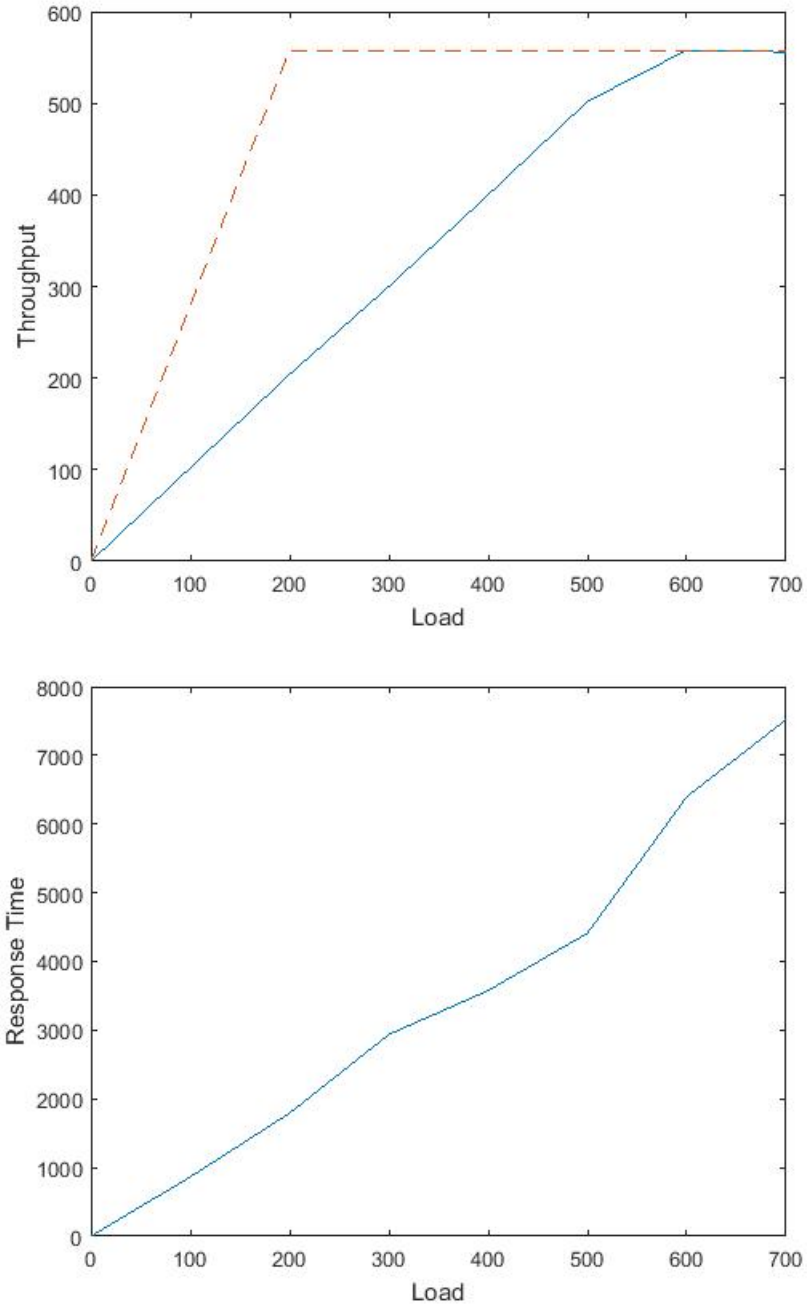

Figura 3.3: Pagine Piccole

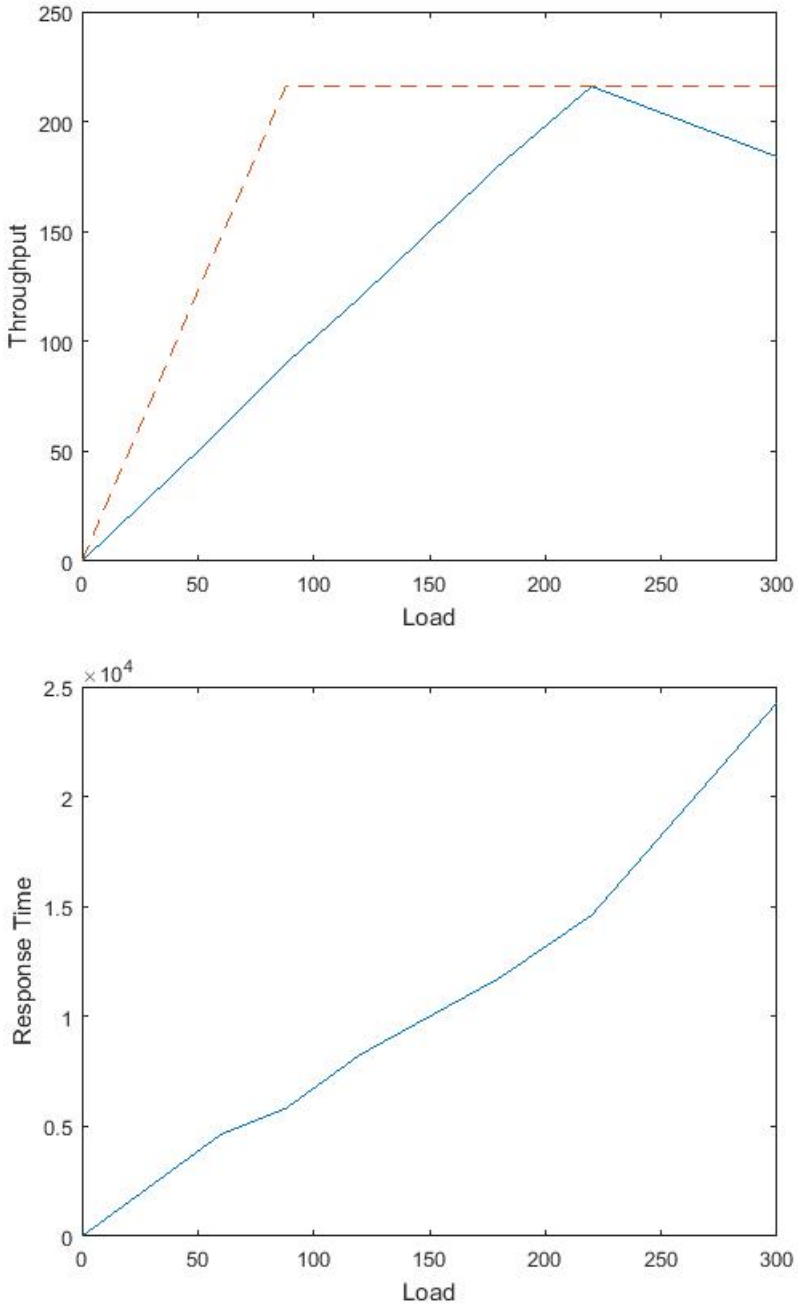

Figura 3.4: Pagine Medie

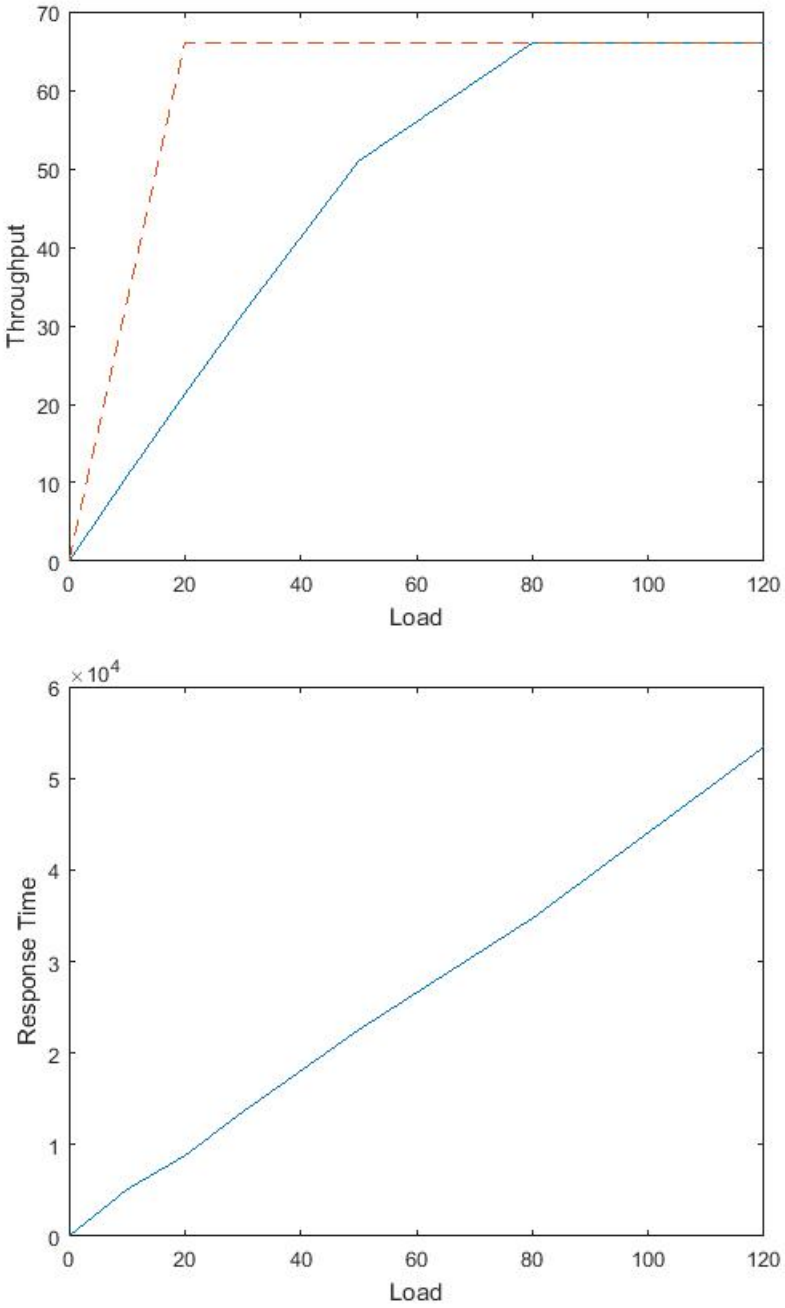

Figura 3.5: Pagine Grandi

Infine possiamo ricavare i valori per il caso medio ed il caso peggiore delle 3 tipologie di pagine (**Tabella 3.2**). Osserviamo come il caso medio presenti valori superiori agli altri, poiché molto influenzato dal peso delle pagine

Tabella 3.2: Capacity Test

Case	Usable Capacity	Knee Capacity
Random	120	50
Average	300	102.7
Worst	80	20

piccole. Esse rappresentano la maggioranza delle pagine web di dimensione di centinaia di KB, escluse quelle relative ai social network, e non riescono a saturare velocemente il server.

3.3 Experimental Design and Analysis

Per studiare l'impatto del tasso di richiesta e del tipo di pagina sul tempo di risposta medio, si prosegue con la tecnica del *Design of Experiment*. I fattori sono stati categorizzati. Si sono considerati 2 tipi di pagina: Piccola (Amazon.html), Grande (Sample-jpg-image-5mb.jpg). Si sono poi determinati 4 livelli per il tasso di richiesta, corrispondenti al 20%, 40%, 60%, 80% della usable capacity media delle pagine: Low, Low-Medium, High-Medium, High. I trattamenti sono stati ripetuti per 10 volte ed in ordine casuale, con durata di 1 minuto ciascuno. Tramite JMP si è ricavata la stima del modello.

Riepilogo della stima

R-quadro	0,863545
R-quadro corretto	0,856267
Scarto quadratico medio	8835,492
Media della risposta	23374,54
Osservazioni (o somma pesata)	80

Analisi della varianza

Origine	DF	Somma dei quadrati	Media quadratica	Rapporto F
Modello	4	3,7053e+10	9,2631e+9	118,6578
Errore	75	5854943448	78065913	Prob > F
C. totale	79	4,2907e+10		<,0001*

Mancata stima

Origine	DF	Somma dei quadrati	Media quadratica	Rapporto F
Mancata stima	3	4576072211	1,5254e+9	85,8771
Errore puro	72	1278871238	17762101	Prob > F
Errore totale	75	5854943448		<,0001*

R-quadro max.
0,9702

Test degli effetti

Origine	N param	DF	Somma dei quadrati	Rapporto F	Prob > F
Page Size	1	1	3,0075e+10	385,2456	<,0001*
Intensity	3	3	6977964343	29,7952	<,0001*

Figura 3.6: Analisi della varianza

Dal rapporto della somma dei quadrati dei fattori rispetto a quella totale, si evince la loro importanza: 70% della variazione totale è attribuito a Page Size, 16.3% ad Intensity, il resto all'errore. In realtà parte dell'importanza dell'errore è dovuta all'interazione tra i fattori trascurata, che rappresenta il 10.7% della variazione totale. Il modello, quindi, spiega circa 86.3% di SST, come testimonia anche R^2.

Per la scelta del tipo di analisi, si provano le assunzioni di normalità ed omoschedasticità. Per la normalità si effetua il *Quantile-Quantile plot* dei residui. Si osserva che la loro distribuzione è asimmetrica. Ciò può essere anche confermato dal test di *Shapiro-Wilk*, che restituisce $p < 0.05$, rigettando l'ipotesi di normalità.

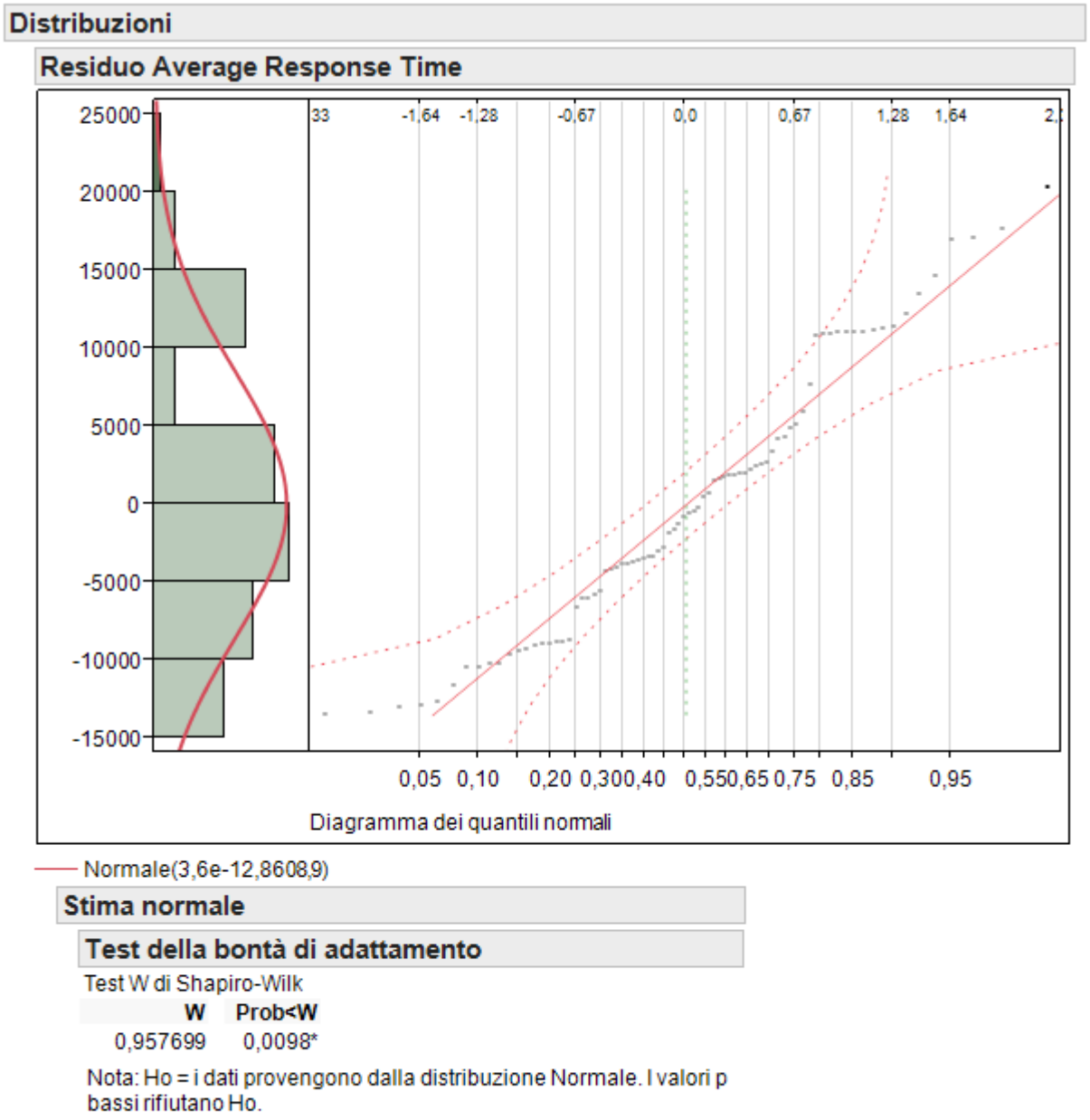

Figura 3.7: Q-Q plot e test di Shapiro-Wilk

Necessitiamo, quindi, di un'analisi non parametrica. Ciò può farci già propendere per il test di *Wilcoxon/Kruskal-Wallis*, anche senza valutare

l'omoschedasticità (che da test visuale risulta non verificata).

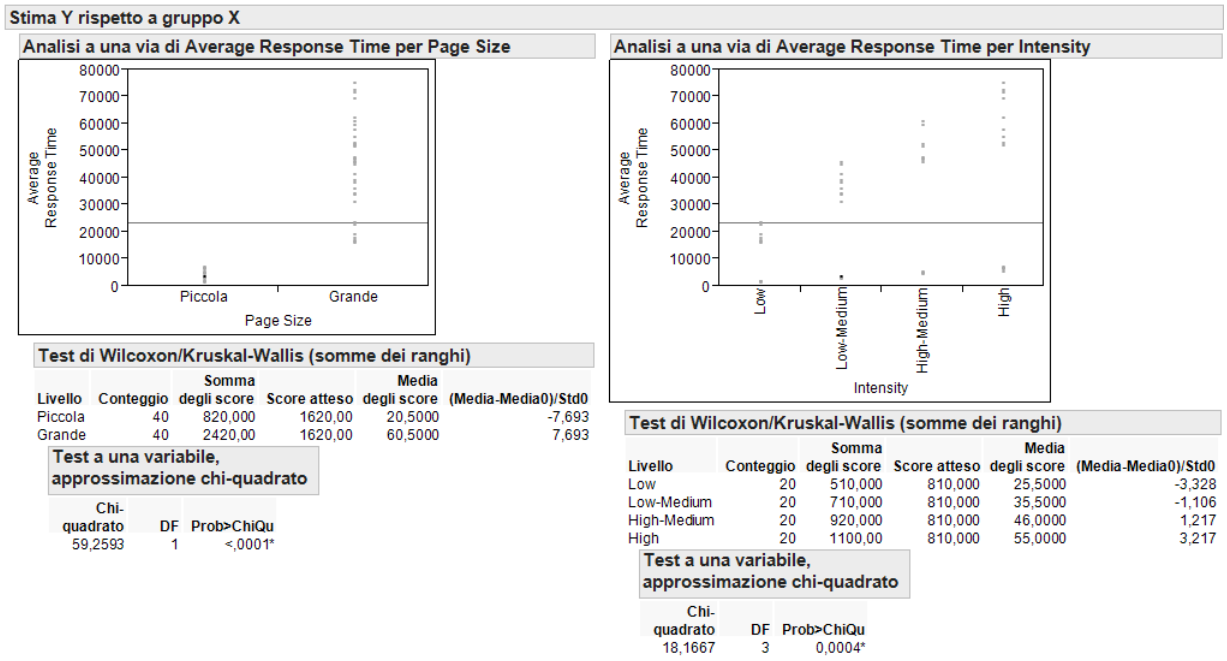

Figura 3.8: Test di Wilcoxon/Kruskal-Wallis

Il test di Kruskal-Wallis rigetta l'ipotesi nulla (campioni dalla stessa popolazione) per entrambi i fattori. Troviamo, infatti, $p < 0.05$, quindi entrambi gli effetti sono statisticamente significativi con livello di significatività 0.05.