Università degli Studi di
Napoli Federico II

Dipartimento di Ingegneria Elettrica e delle
Tecnologie dell'Informazione

Impianti di Elaborazione

`Elaborato`

Michele Pommella M63/790
Davide Trimaldi M63/799

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%.

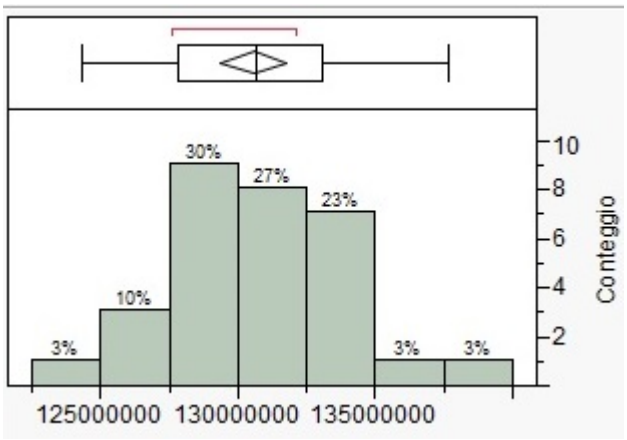

Figura 1.1: Scrittura di un file di $100MB$ con blocchi di $100KB$

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

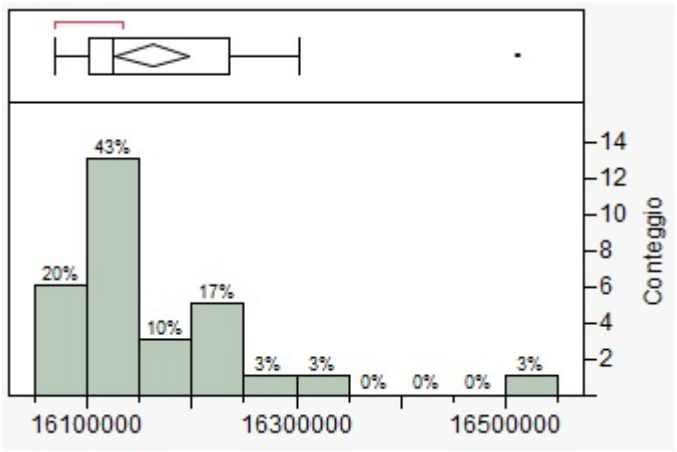

Figura 1.2: Lettura di un file di $1GB$ con blocchi di $1MB$

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:

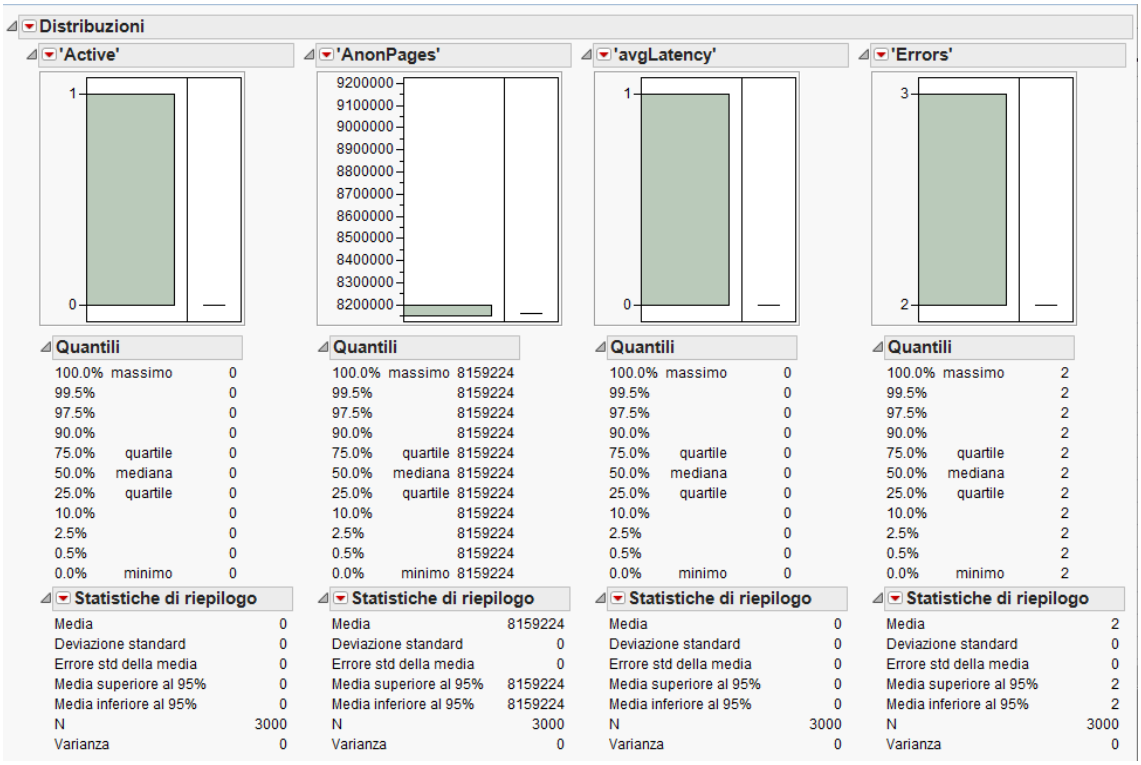

Figura 2.1: Distribuzione colonne costanti

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**:

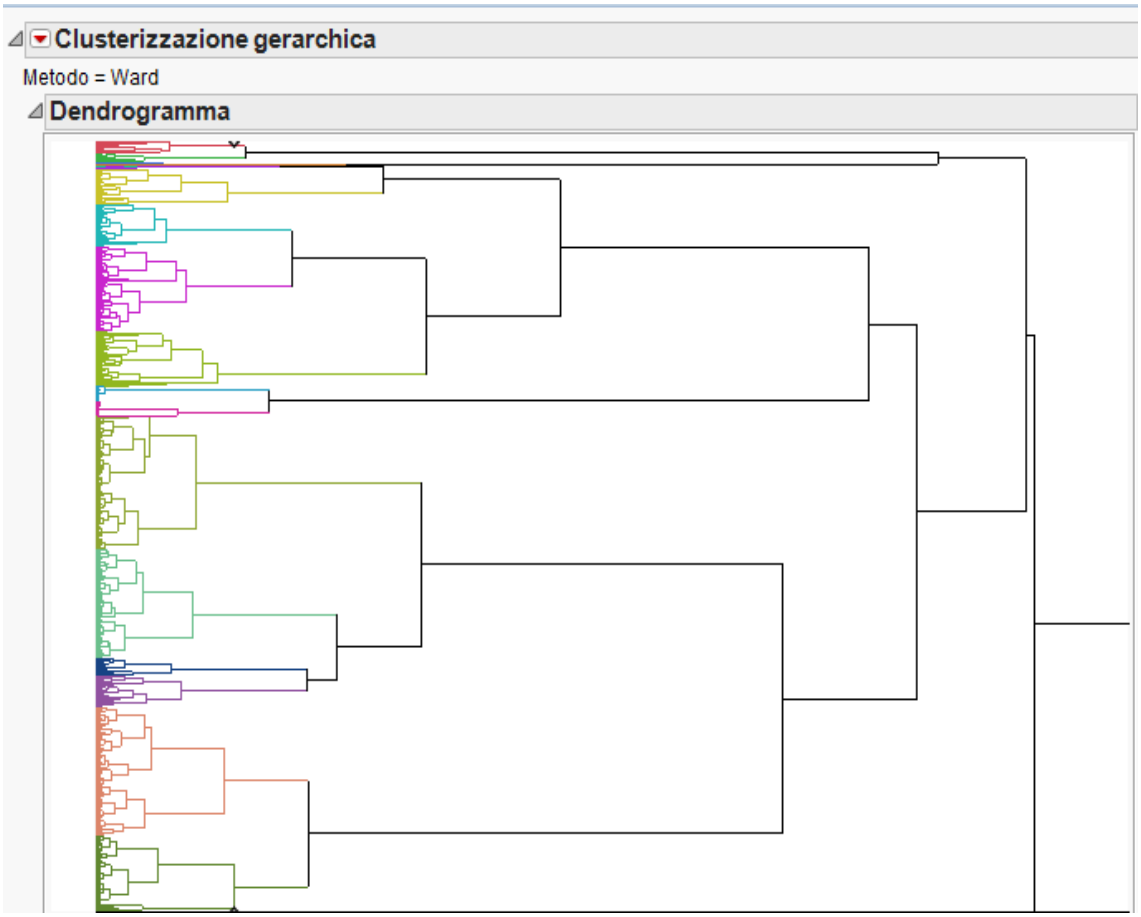

Figura 2.3: Dendrogramma

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.

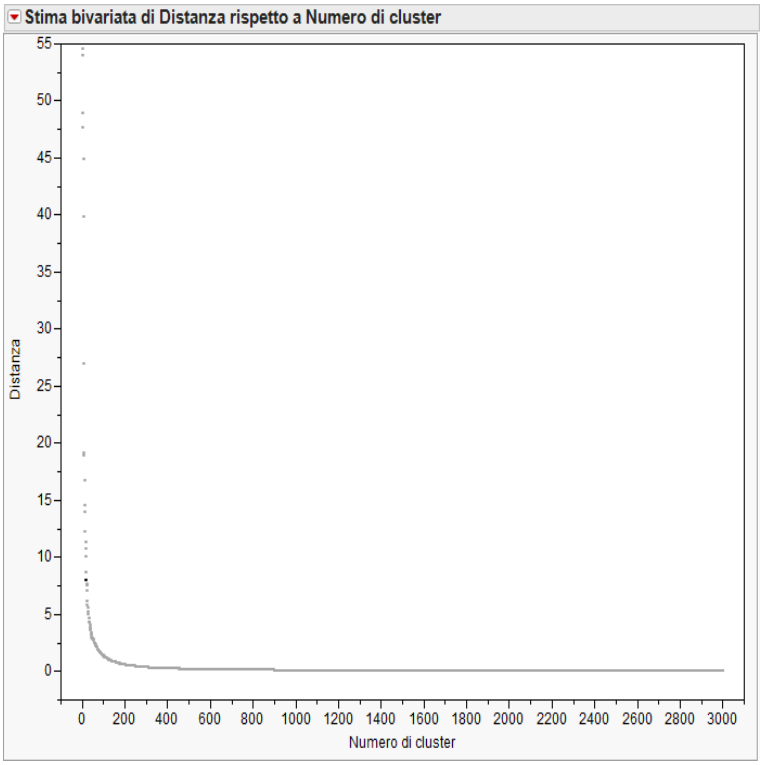

Figura 2.5: Grafico devianza persa al variare del numero di cluster

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.

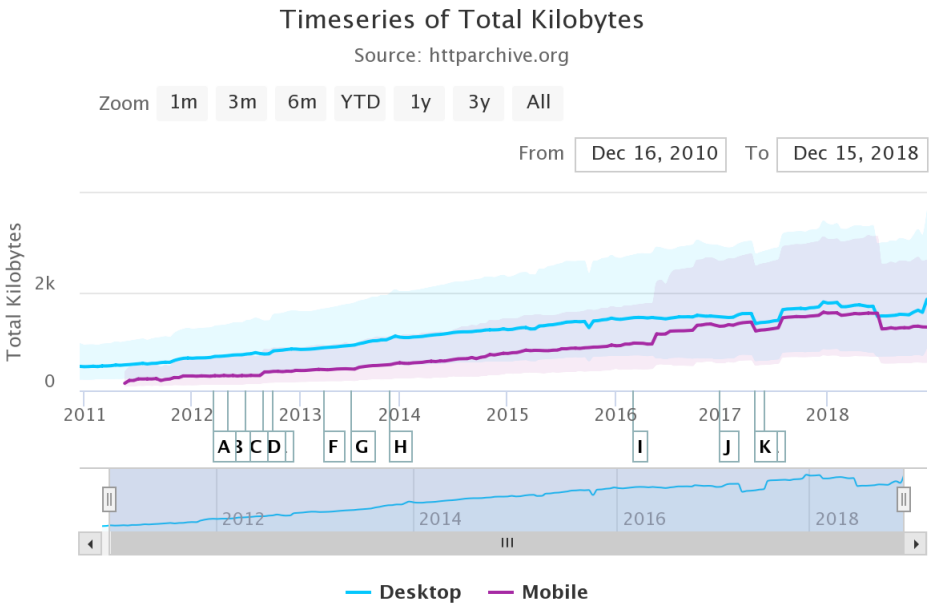

Figura 3.1: Stato del Web: Kilobyte totali

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 $29KB$
- *FrancoCFA.html* di $112KB$
- *Amazon.html* di $460KB$
- *Facebook.html* di $1.4MB$
- *Sample-jpg-image-5mb.jpg* di $5MB$

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.

3.1.1 WL Dati Application Level

Nei 5 minuti di test sono state prelevate 515 istanze (richieste inviate al web server) descritte da 17 parametri app-level. Tra questi, quelli maggiormente significativi sono:

- *Elapsed*: tempo tra la richiesta e l'ultima risposta
- *Label*: tipologia di pagina in base alla sua dimensione
- *ThreadName*: thread a cui è stata assegnata la richiesta
- *Success*: esito della richiesta
- *Bytes*: dimensione della pagina richiesta
- *Latency*: tempo tra la richiesta e la prima risposta
- *Connect*: tempo necessario ad instaurare la connessione

Notando l'andamento delle distribuzioni di questi parametri, si è concluso che ha senso caratterizzare statisticamente solo tre di questi, ovvero *Elapsed*, *Latency* e *Connect*.

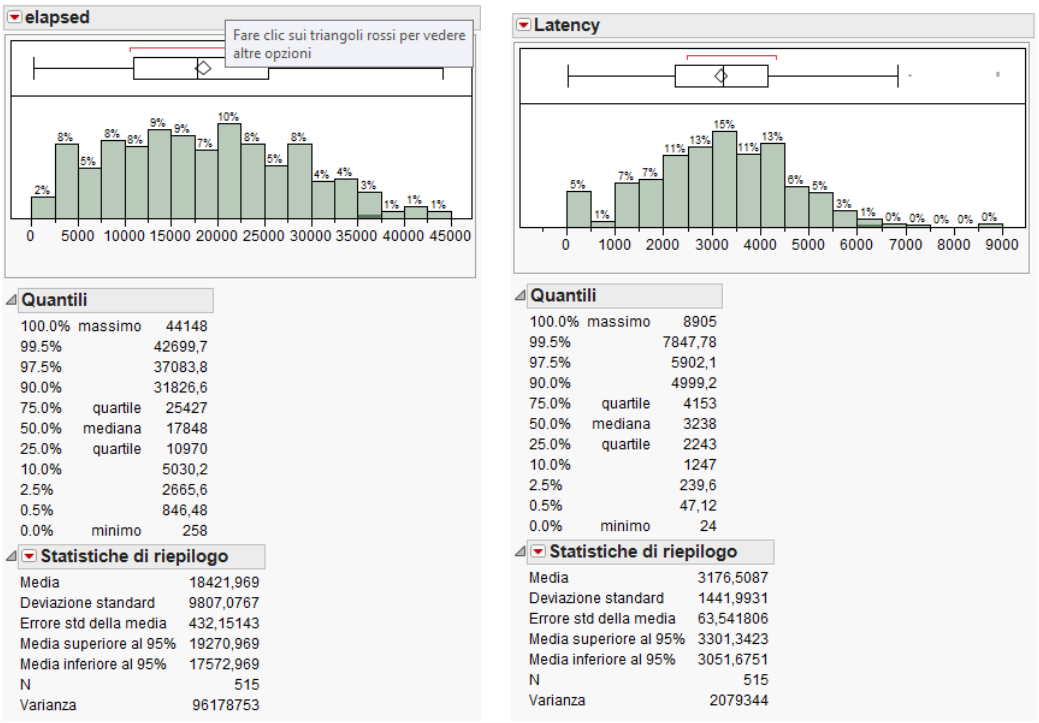

Figura 3.2: Distribuzione Elapsed Time e Latency

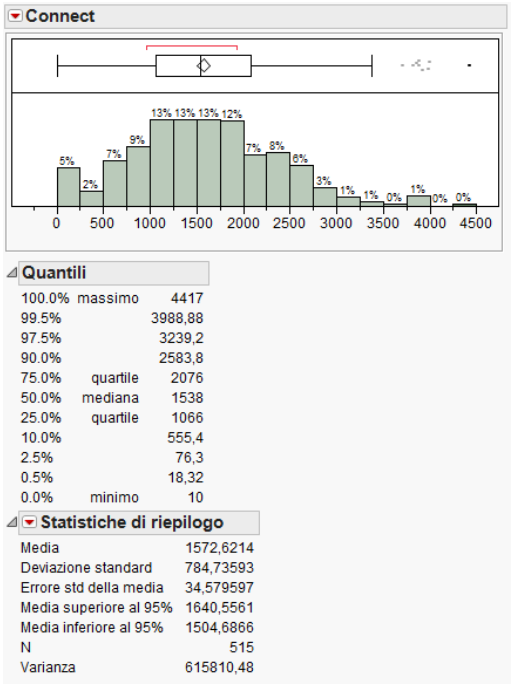

Figura 3.3: Distribuzione Connect

Osservando le distribuzioni di questi tre parametri, si nota che sono molto poco skewed, quindi si è concluso che la media è un buon indice statistico in grado di descriverle.
Consideriamo ora la distribuzione del parametro *Success*:

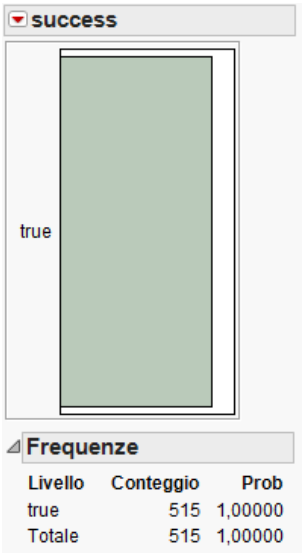

Figura 3.4: Distribuzione Success

Come si nota, tutte le richieste sono andate a buon fine, poiché la terminazione dell'esperimento è stata demandata allo *Scheduler*, che ne ha atteso la conclusione.

Osserviamo ora la distribuzione dei parametri *Label* e *ThreadName*:

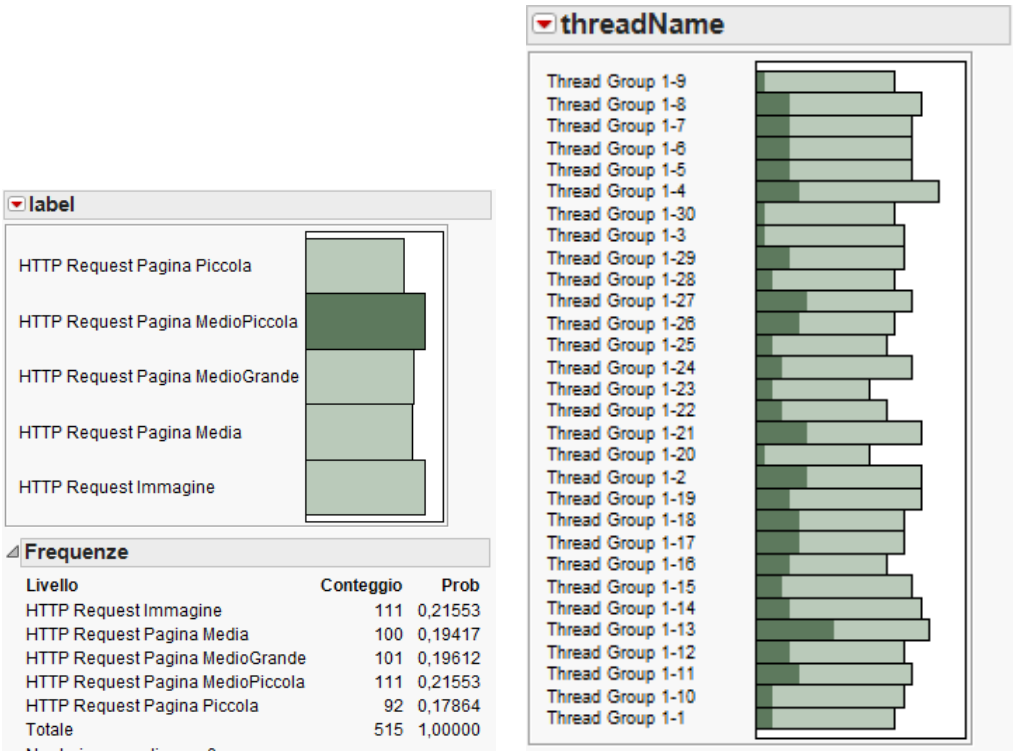

Figura 3.5: Distribuzione Label e ThreadName

Come si nota ogni tipologia di pagina è distribuita abbastanza equamente tra i thread, per una distribuzione del carico alquanto regolare.

Infine si è deciso di verificare come variano i parametri *Elapsed*, *Latency* e *Connect* al variare della dimensione delle pagine richieste.

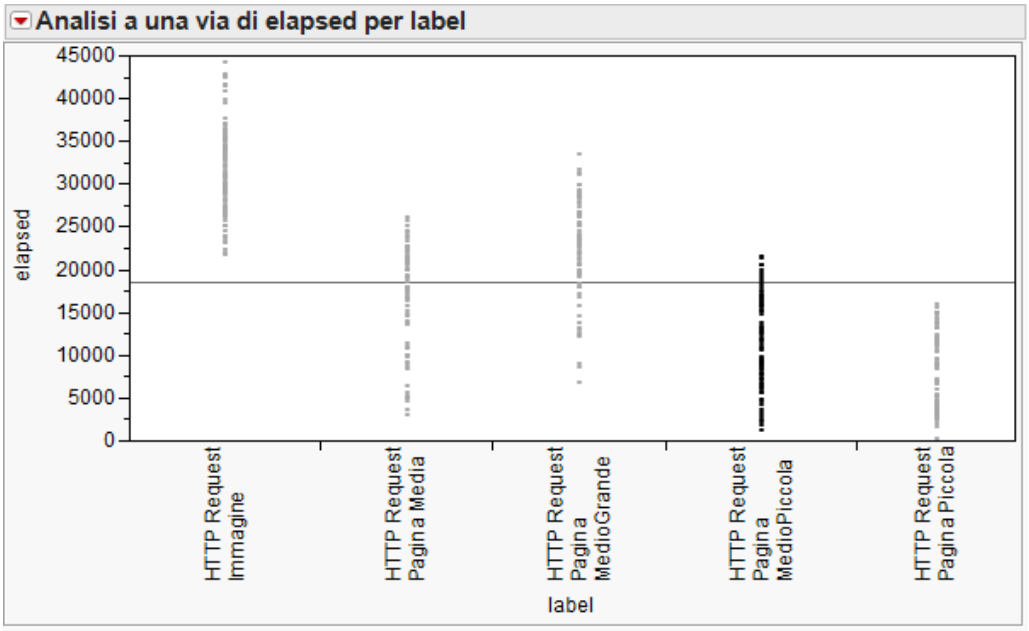

Figura 3.6: Grafico variazione Elapsed rispetto alla tipologia di pagina

Come si nota dalla **Figura 3.6**, l'elapsed time cresce al crescere della dimensione delle pagine (misurata in bytes).
Di seguito invece gli altri due grafici mostrano come latency e tempo di connessione non dipendano dalla grandezza delle pagine.

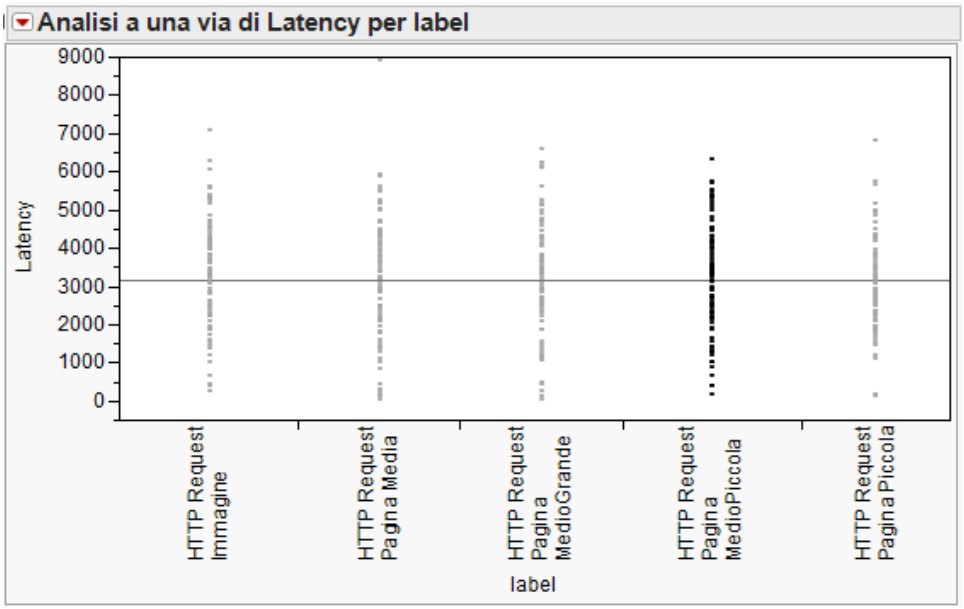

Figura 3.7: Grafico variazione Latency rispetto alla tipologia di pagina

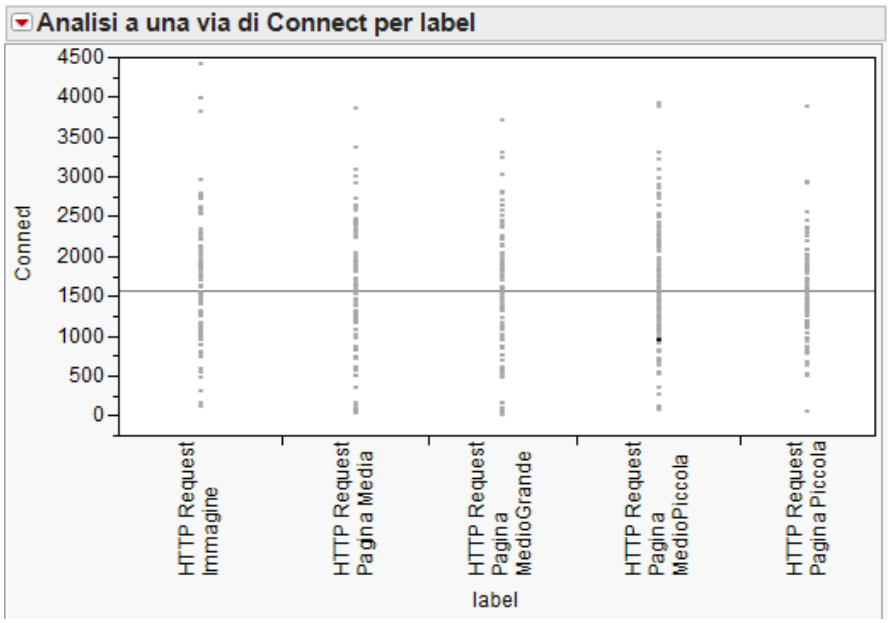

Figura 3.8: Grafico variazione Connect rispetto alla tipologia di pagina

Tutto quello appena detto si può verificare anche calcolando la correlazione tra i parametri elapsed, latency, bytes e connect. Calcoliamo quindi la matrice di correlazione:

Multivariato

Correlazioni

	elapsed	bytes	Latency	Connect
elapsed	1,0000	0,7873	0,4289	0,4236
bytes	0,7873	1,0000	0,0470	0,0740
Latency	0,4289	0,0470	1,0000	0,9013
Connect	0,4236	0,0740	0,9013	1,0000

Figura 3.9: Matrice di correlazione

Si può vedere come elapsed e bytes siano molto correlati a differenza degli altri parametri, questo perché più è grande la pagina richiesta in termini di byte, più è grande il tempo di risposta.

3.1.2 WL Dati System Level

Passiamo all'analisi dei dati di basso livello del sistema. Nei 5 minuti in cui è stato eseguito il test, sono stati prelevate 360 istanze (una al secondo) descritte da 17 parametri. Di queste sono state eliminate le prime tre e le ultime 26 perché precedenti e successive all'esperimento così come si nota dal corrispondente basso numero di interruzioni e dall'alta percentuale di tempo in cui il sistema è idle.
Per prima cosa osservando le distribuzioni dei parametri, sono state eliminate le colonne a varianza nulla che sono 6. Sulle colonne rimaste applichiamo la tecnica della PCA per trovare le componenti principali.

Autovalori

Numero	Autovalore	Percentuale	20 40 60 80	Percentuale cumulativa
1	3,1115	28,287		28,287
2	1,4151	12,864		41,151
3	1,2918	11,743		52,894
4	1,0477	9,524		62,418
5	0,9591	8,719		71,137
6	0,9325	8,477		79,614
7	0,8272	7,520		87,134
8	0,6983	6,348		93,482
9	0,4022	3,657		97,139
10	0,3144	2,858		99,997
11	0,0003	0,003		100,000

Figura 3.10: Componenti principali e varianza conservata

Osservando gli autovalori, si è scelto di prendere 9 componenti principali che spiegano il 97% della varianza.
Dopo aver fatto questo, si adotta la tecnica di clustering di Ward sulle 9 componenti per ridurre il numero di istanze del dataset originario. Osservando il dendrogramma si è determinato che il salto si trovasse a 17 cluster, che determinano una perdita di varianza pari al 30%, permettendo quindi di conservarne il 70%.

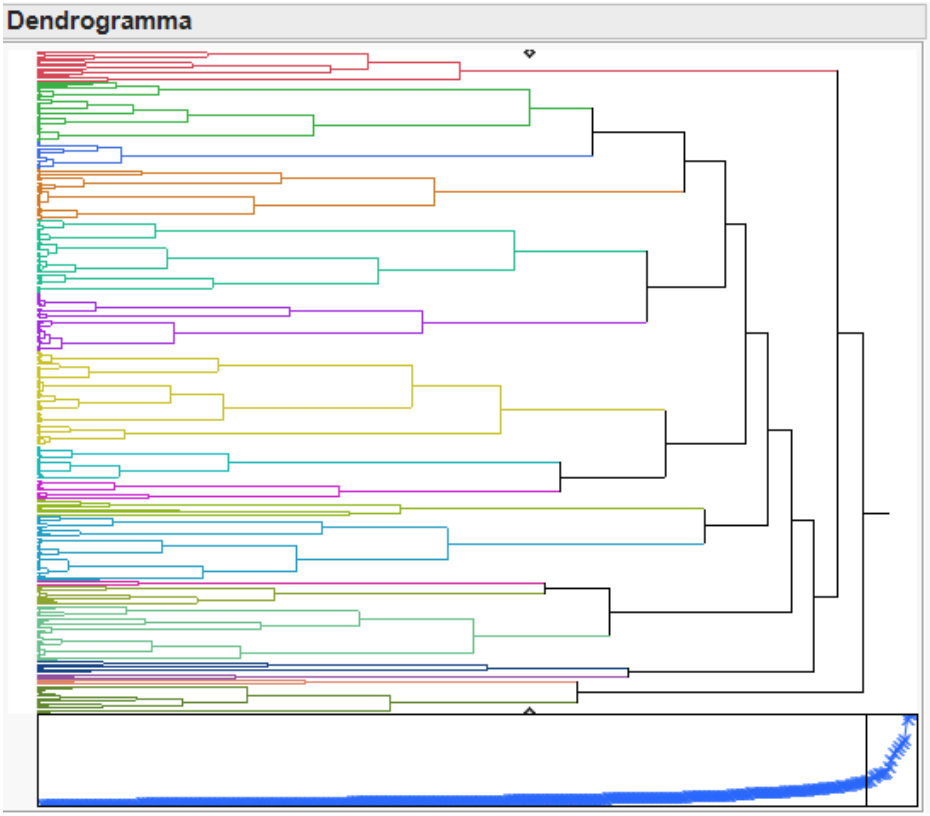

Figura 3.11: Dendrogramma

Infine riportiamo l'insieme di istanze rappresentative del dataset oroginario, selezionate in maniera casuale prendendone una per ogni cluster.

	Principale1	Principale2	Principale3	Principale4	Principale5	Principale6	Principale7	Principale8	Principale9	Cluster
1	-0,382425984	4,681873782	-0,652141403	-2,717575228	1,072573549	2,709172168	-0,072024913	-0,130463253	-0,154242786	1
2	2,603386883	3,215806454	0,242177488	-0,171072164	-1,448189654	0,791009703	1,096584987	-0,4514812	0,889241016	2
3	-1,218179892	0,839914332	-0,351164636	-0,150820337	-0,593736164	0,146846538	1,054037244	0,069111364	0,616686572	3
4	0,064241172	0,278573928	-0,181182025	0,032002814	-0,81948223	-0,437132991	-1,183338745	-0,607823301	1,507092827	4
5	1,396302174	0,016721492	-0,257738248	-0,299327031	-1,166883653	0,184760861	-0,545938484	-0,813654053	-0,497207133	5
6	1,873233635	0,068317848	-0,400209897	-0,382485887	-0,553144181	0,222267331	0,726824661	-0,33801226	-0,121478422	6
7	1,824539398	2,045689593	0,044314604	-0,159648103	-1,321331096	0,543830091	0,565638006	-0,605253361	-0,400566343	7
8	-1,148968958	-0,260456573	-0,239008743	-0,161246432	-1,488750945	-0,013265721	-1,557295502	-0,870743001	0,115374114	8
9	0,14659867	0,390892073	-0,078370145	0,493383722	-0,193269674	-0,948453851	-1,495794428	-0,807624905	-1,352397715	9
10	3,594360007	1,488046721	2,730159765	1,137473213	1,549790707	-1,621240719	-1,454679242	2,095408443	-0,441346246	10
11	-3,510023832	1,81933061	-0,086561781	0,778240612	0,131149671	-0,976489559	0,123913404	0,076916515	-0,444035302	11
12	0,486521247	0,754591869	4,365630206	-0,309915338	-1,57991628	1,404307067	-1,098287656	3,711227379	-0,454844847	12
13	-1,448263105	1,15546041	1,03513575	-0,026544524	-1,402714345	0,492795631	-0,48591593	0,538976025	-0,755821492	13
14	0,338849826	-1,103489905	3,637132707	-0,571163666	-0,343963613	0,773419502	-0,565672845	-2,720296233	-0,340443413	14
15	-0,445345932	-0,652099181	6,789191356	-0,444305062	1,635460523	0,882460943	1,013361798	-4,863064356	0,48820485	15
16	3,239671309	2,57059292	-0,346036962	3,683994683	0,666790924	2,554017345	0,122489311	-0,595832478	-0,388642904	16
17	-1,146304571	-0,78362763	0,150649793	2,986779519	-0,244135298	3,158698731	-1,339893741	0,49170078	0,542610261	17

Figura 3.12: Dati finali

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.

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

I risultati ottenuti sono riportati in **Tabella 3.1**. Di seguito l'andamento di throughput (1/min) e response time (ms), all'aumentare del carico nei 4 casi considerati.

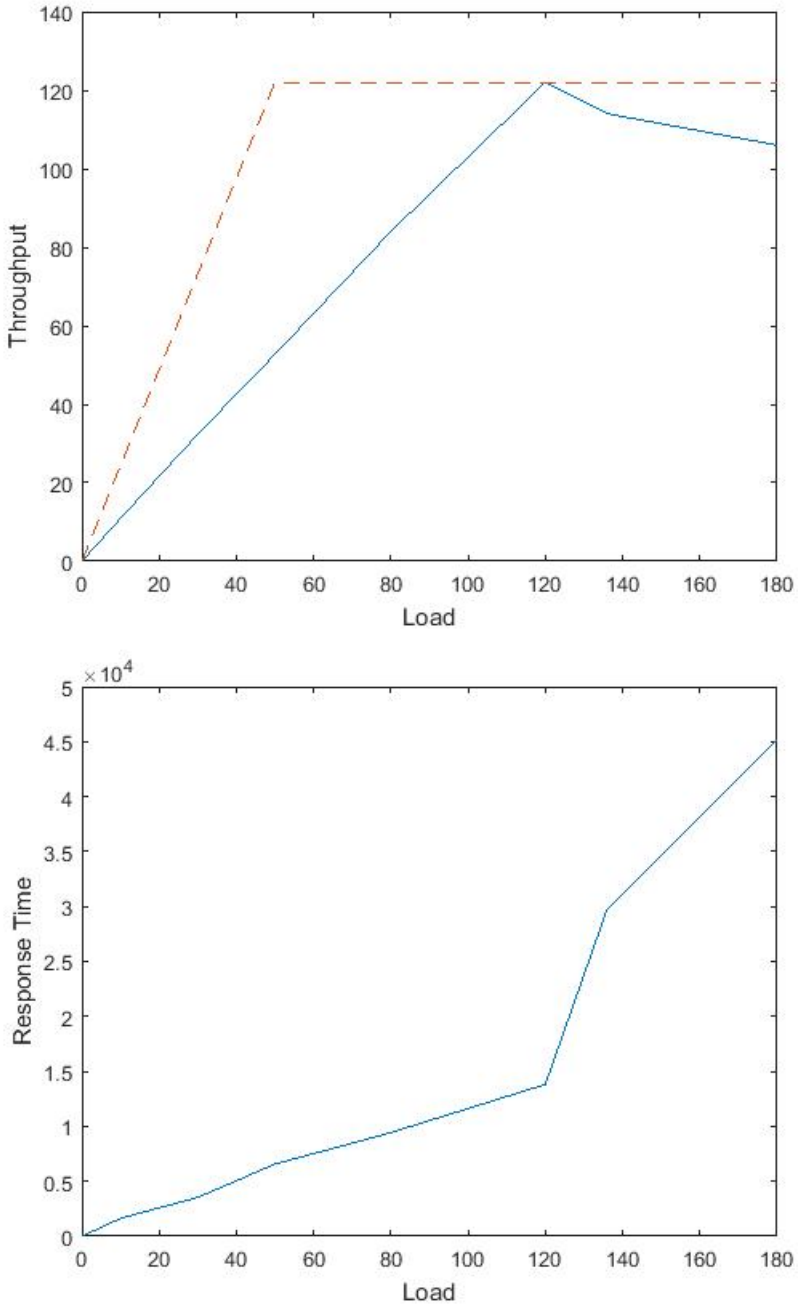

Figura 3.13: Pagine Random

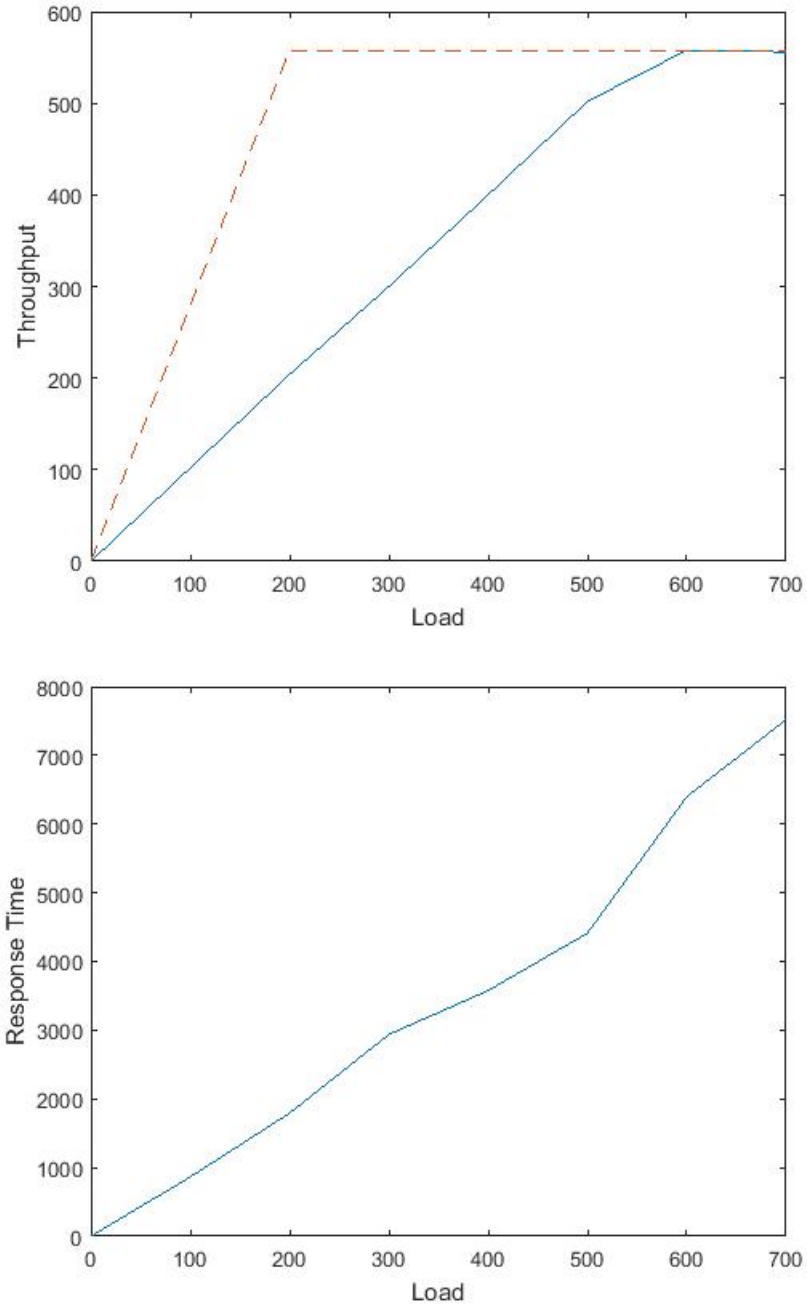

Figura 3.14: Pagine Piccole

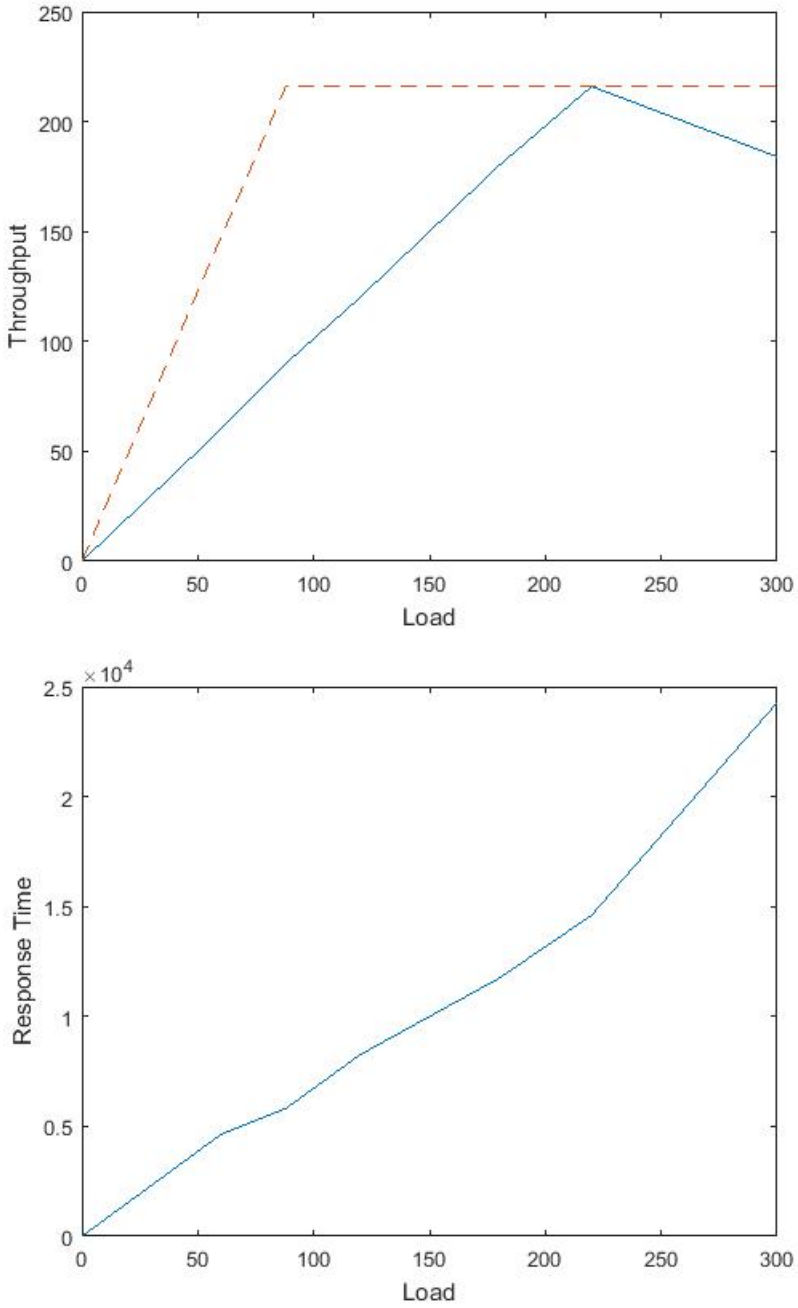

Figura 3.15: Pagine Medie

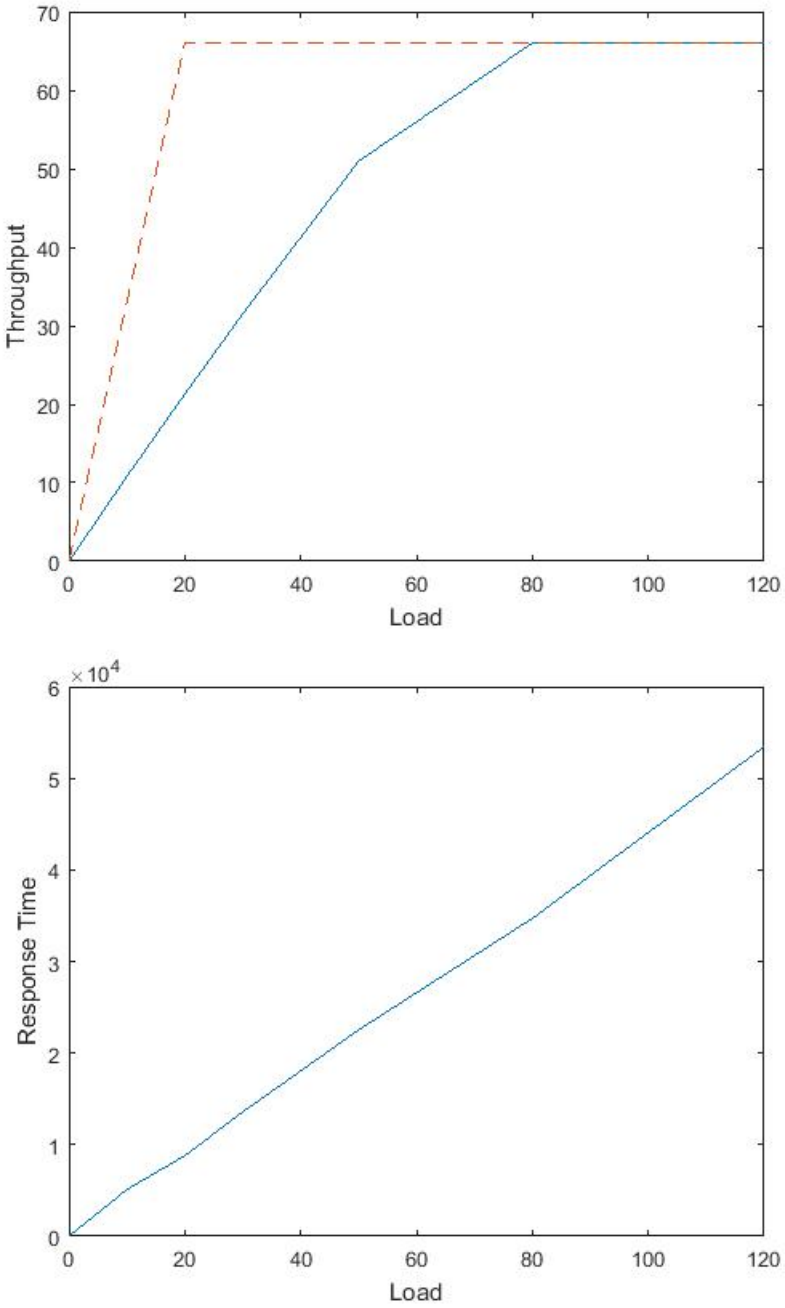

Figura 3.16: Pagine Grandi

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 3.17: 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.

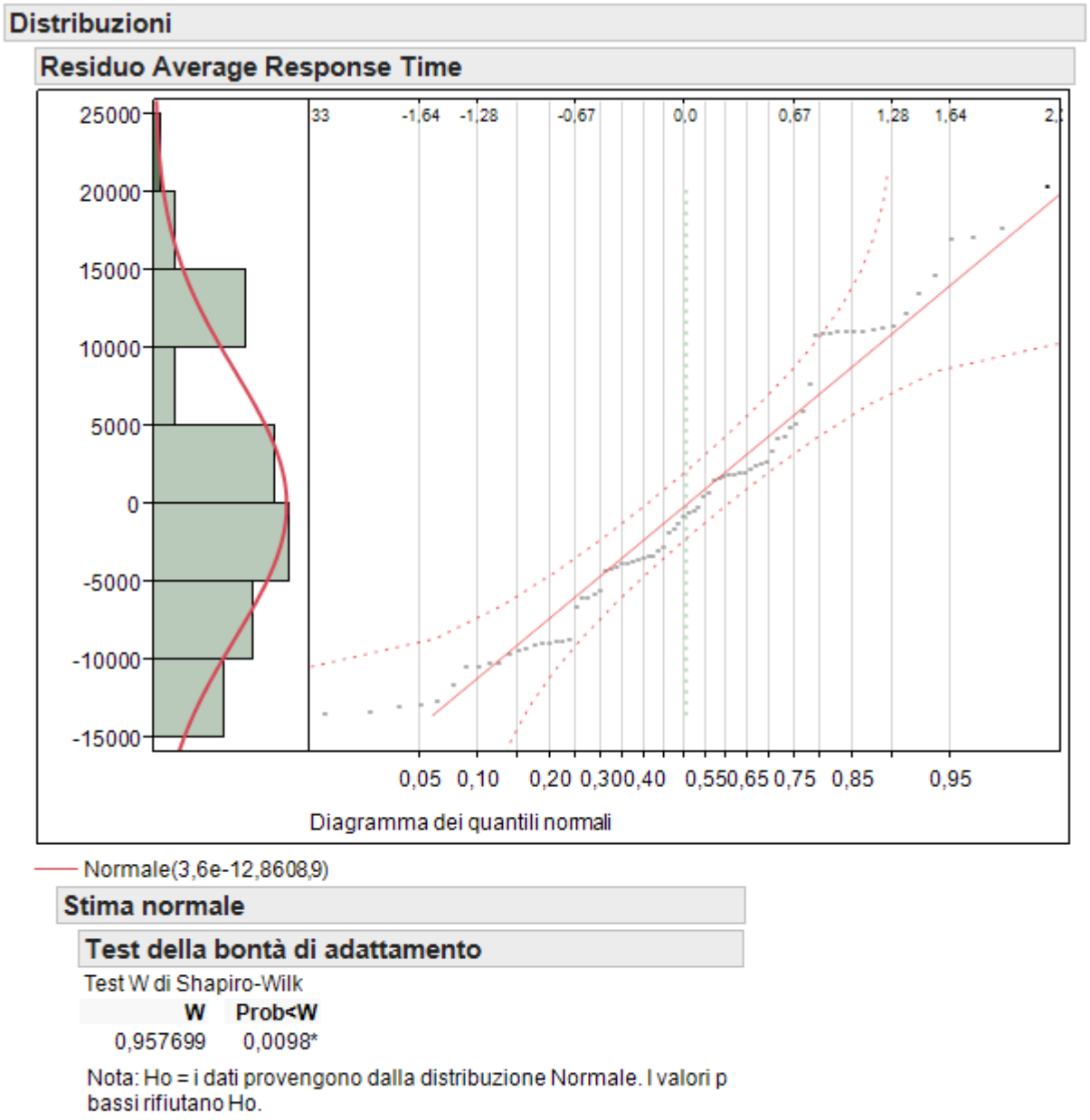

Figura 3.18: Q-Q plot e test di Shapiro-Wilk

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).

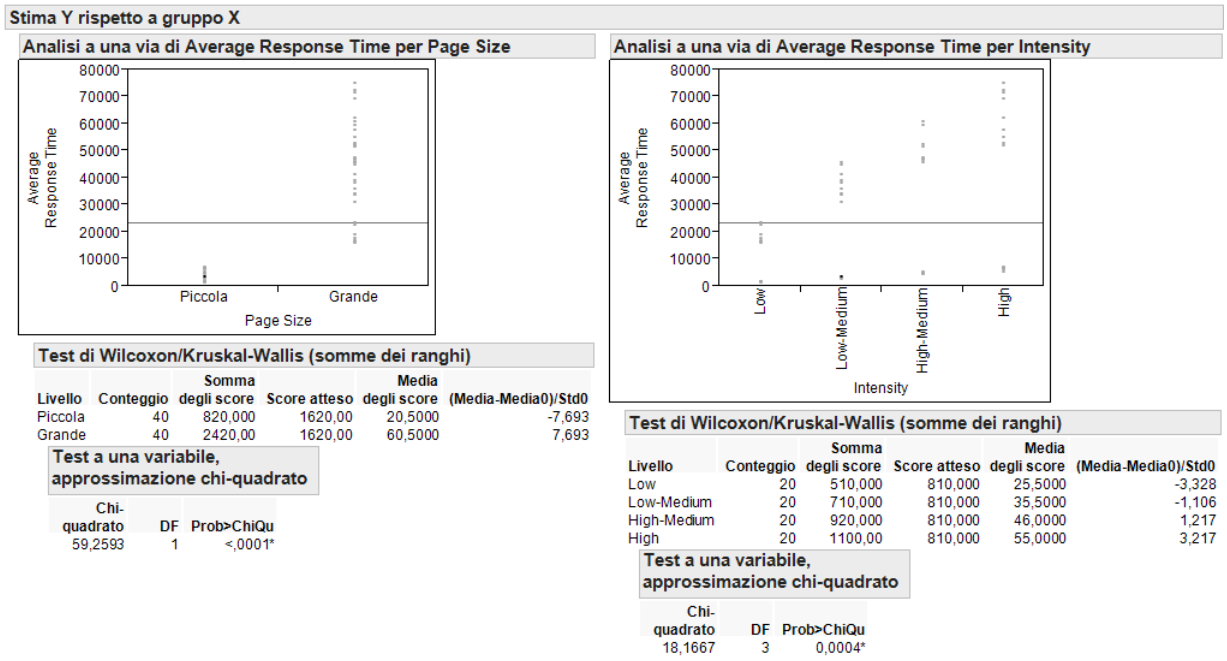

Figura 3.19: Test di Wilcoxon/Kruskal-Wallis

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.

Capitolo 4

Esercizi Reliability

4.1 Esercizio 1

Calcolare la reliability di R(t) e l'MTTF del sistema il cui diagramma di reliability è mostrato in figura. Nel calcolare l'MTTF assumere che tutti i componenti siano uguali e che falliscano randomicamente con failure rate pari a λ.

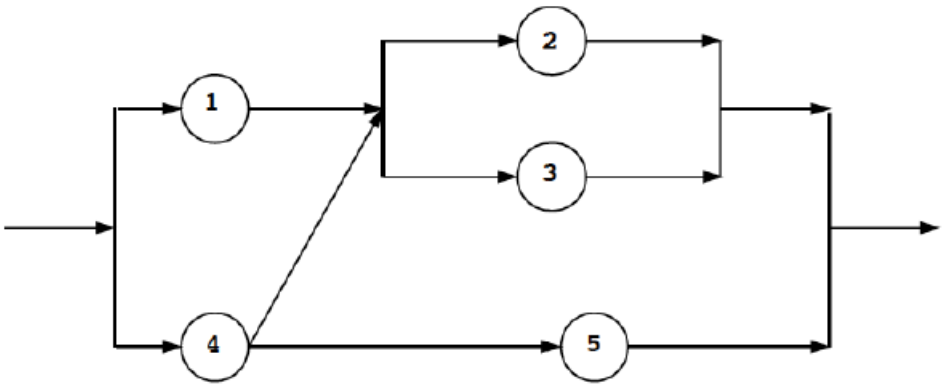

Figura 4.1: Diagramma di reliability del sistema

Tale sistema non è nella forma serie-parallelo, quindi lo riconduciamo ad un sistema del genere scegliendo un componente che viene considerato una volta come funzionante e una volta no, utilizzando poi la seguente relazione che viene dalla regola di Bayes.

$$R_{sys} = R_m P(System\ works|m\ works) + (1 - R_m) P(System\ works|m\ fails)$$

Il componente scelto è il numero 4, in quanto si ritiene che tale scelta possa semplificare maggiormente il diagramma. Considerando il componente 4 come funzionante, esso viene sostituito con un corto circuito, quindi si elimina anche il componente 1.
A questo punto si può già calcolare la reliability dei due componenti in parallelo 2 e 3:

$R_{23} = 1 - (1 - R_m)^2 = 1 - (1 + R_m^2 - 2R_m) = -R_m^2 + 2R_m =>$

$=> 1 - R_{23} = (1 - R_m)^2$

Calcolo la probabilità che il sistema funzioni dato il componente 4 funzionante:

$P(sys\ works|4\ works) = 1 - (1 - R_m)(1 - R_{23}) = 1 - (1 - R_m)^3 =$
$= R_m^3 + 3R_m - 3R_m^2$

Considero ora il componente 4 come non funzionante, quindi viene sostituito con un circuito aperto. Di conseguenza si elimina il componente 5.
Calcolo la probabilità che il sistema funzioni dato il componente 4 non funzionante:

$P(sys\ works|4\ fails) = R_m R_{23} = 2R_m^2 - R_m^3$

A questo punto posso calcolare la reliability del sistema:

$R_{sys} = R_m^4 + 3R_m^2 - 3R_m^3 + (1 - R_m)(2R_m^2 - R_m^3) = R_m^4 + 3R_m^2 - 3R_m^3 + 2R_m^2 -$
$+ R_m^3 - 2R_m^3 + R_m^4 = 2R_m^4 - 6R_m^3 + 5R_m^2$

Assumendo che l'andamento della reliability sia un esponenziale negativo, posso così calcolare l'MTTF facendone l'integrale tra 0 e ∞.

$MTTF = \int_0^\infty R_{sys}(t)\ dt = \int_0^\infty 2e^{-4\lambda t} - 6e^{-3\lambda t} + 5e^{-2\lambda t}\ dt =$
$= 2 * \frac{1}{4\lambda} - 6 * \frac{1}{3\lambda} + 5 * \frac{1}{2\lambda} = \frac{1}{\lambda} * (\frac{1}{2} - 2 + \frac{5}{2}) = \frac{1}{\lambda}$

In conclusione, quindi, possiamo dire che l'MTTF del sistema complessivo coincide con quello del singolo componente.

$MTTF_{sys} = MTTF_{simplex}$

4.2 Esercizio 2

Si vogliono mettere a confronto due schemi che cercano di aumentare la reliability di un sistema usando ridondanza. Supponiamo che il sistema abbia bisogno di s componenti identici in serie per le proprie operazioni. Supponiamo anche che siano dati $(m * s)$ componenti. Quale dei due schemi mostrati in figura fornirà una reliability maggiore? Considerando che la reliability di un singolo componente sia r, deriva le espressioni per le reliability delle due configurazioni. Confronta le due espressioni per m=3 e s=3.

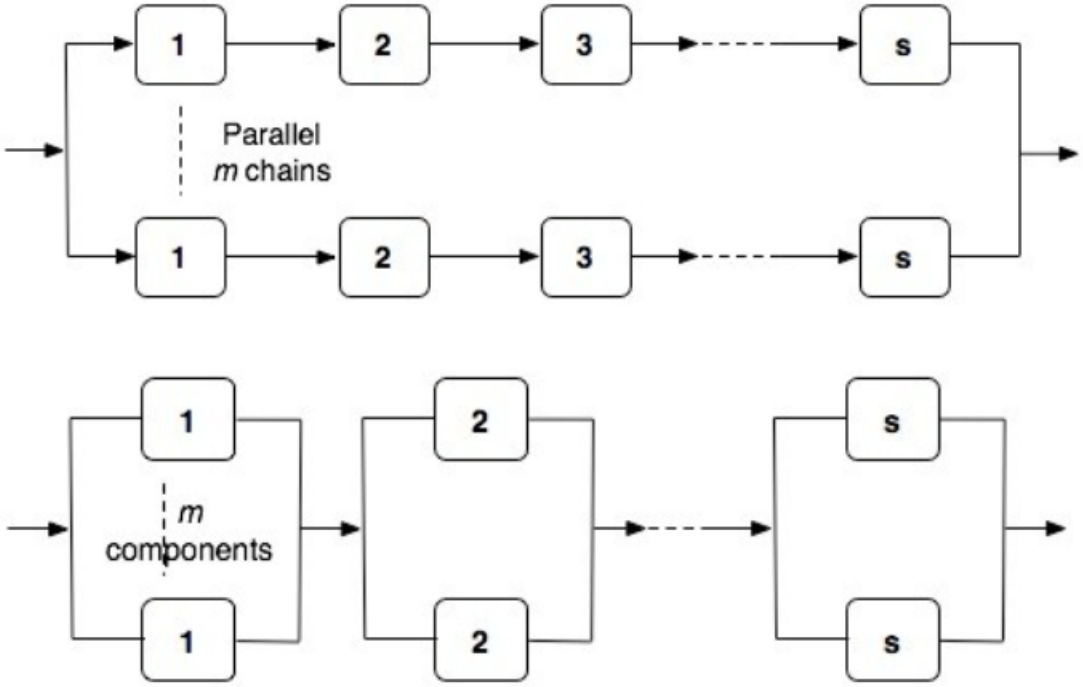

Figura 4.2: Due diversi schemi del sistema che adottano la ridondanza

A occhio si nota subito che il secondo schema è migliore del primo qualunque siano m ed s. Difatti, mentre nel primo caso deve funzionare per forza ogni componente di una delle m serie affinché il sistema funzioni, nel secondo invece per ognuno degli m componenti in parallelo, basta che ne funzioni uno affinché funzioni tutto il sistema. Quindi si può dire che ci sono più probabilità che il primo sistema fallisca.
Riportiamo ora le espressioni delle reliability dei due sistemi:

$R_{sys1} = 1 - \prod_{i=1}^{m}(1 - \prod_{j=1}^{s} r_{ij})$

$R_{sys2} = \prod_{j=1}^{s}[1 - \prod_{i=1}^{m}(1 - r_{ij})]$

Dato che i componenti sono tutti uguali, essi hanno le stesse reliability quindi posso scrivere le espressioni precedenti come segue:

$R_{sys1} = 1 - (1 - r^s)^m$

$R_{sys2} = (1 - (1 - r)^m)^s$

A questo punto bisogna confrontare le due espressioni per m=3 e s=3.

$R_{sys1} = 1 - (1 - r^3)^3$

$R_{sys2} = (1 - (1 - r)^3)^3$

Considerando la reliability come un esponenziale negativo e fissando un certo failure rate λ, si confrontano su un grafico le curve della reliability dei due sistemi.

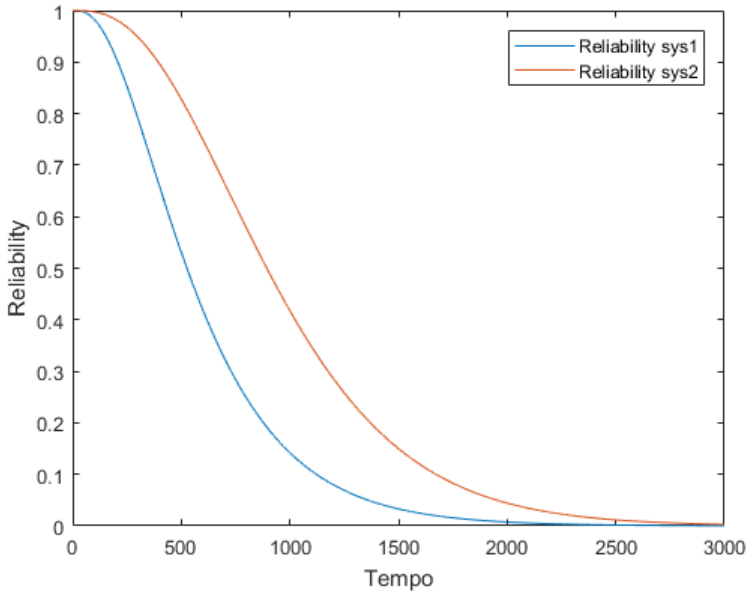

Figura 4.3: Grafico reliability dei sistemi al variare del tempo

Come si nota dal grafico, la reliability del secondo sistema risulta essere sempre maggiore di quella del primo.

4.3 Esercizio 3

L'architettura di una rete di computer in un sistema bancario è mostrata in figura. L'architettura è chiamata rete a skip-ring ed è progettata per permettere ai processori di comunicare anche dopo che si sono presentati dei fallimenti di alcuni nodi. Ad esempio, se fallisce il nodo 1, il nodo 8 può bypassare il nodo fallito istradando i dati su un link alternativo che va dal nodo 8 al nodo 2. Assumendo che i collegamenti siano perfetti e che ogni nodo abbia una reliability R_m, deriva un'espressione per la reliability della rete. Se R_m obbedisce alla legge di fallimento esponenziale e il failure rate di ogni nodo è 0.005 fallimenti all'ora, determina la reliability del sistema al termine di un periodo di 48 ore.

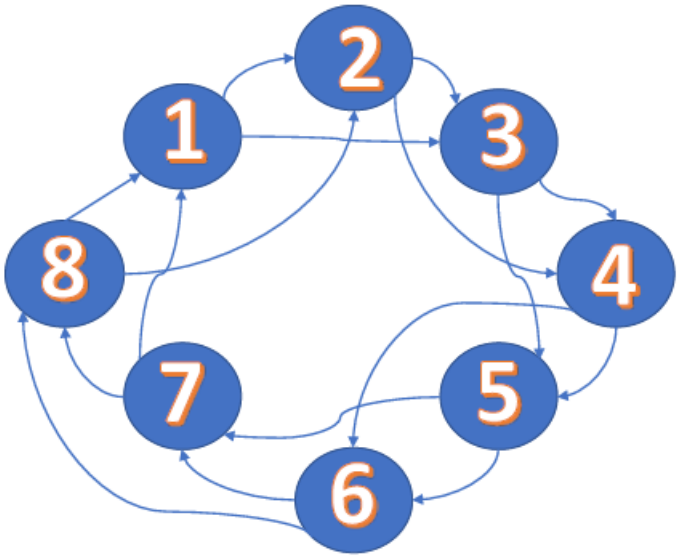

Figura 4.4: Architettura rete di computer

In un certo intervallo di tempo possono fallire da 0 a 8 nodi. Il fallimento del sistema si ha quando falliscono due nodi consecutivi. Quindi la reliability del sistema non è altro che la probabilità che il sistema funzioni in un intervallo di tempo. Tale probabilità si calcola come:
$P(sys\ works) = \sum_{k=0}^{8} P(sys\ works|k\ of\ 8\ nodes\ work)P(k\ of\ 8\ nodes\ work)$

Essendo tutti i nodi indipendenti ed identici con stessa reliability R_m, la probabilità che k nodi su 8 non falliscano si calcola come il prodotto della probabilità che k nodi non falliscano per la probabilità che 8-k nodi falliscano, il tutto moltiplicato per il numero di configurazioni in cui si verifica questo

evento, ottenibile tramite il coefficiente binomiale. Quindi si può scrivere:

$$P(k\ of\ 8\ nodes\ work) = \binom{8}{k}(R_m)^k(1-R_m)^{8-k}$$

Come detto in precedenza il sistema non funziona se ci sono almeno due nodi consecutivi non funzionanti, quindi la probabilità a posteriori dell'espressione precedente si calcola come il rapporto del numero di configurazioni in cui questo non accade (con k nodi funzionanti), fratto il numero di configurazioni in cui ci sono k nodi funzionanti su 8, ovvero il coefficiente binomiale nominato in precedenza.
Tali probabilità sono riportate nella seguente tabella.

Tabella 4.1: Probabilità a-posteriori

Numero nodi funzionanti	**P(sys works|k of 8 nodes work)**
0	0
1	0
2	0
3	0
4	$\frac{2}{70}$
5	$\frac{16}{56}$
6	$\frac{20}{28}$
7	1
8	1

La reliability del sistema sarà quindi:

$$R_{sys} = \sum_{k=0}^{8} P(sys\ works|k\ of\ 8\ nodes\ work) * \binom{8}{k}(R_m)^k(1-R_m)^{8-k} =$$

$$= \tfrac{2}{70} * 70 * (R_m^4)(1-R_m)^4 + \tfrac{16}{56} * 56 * (R_m^5)(1-R_m)^3 + \tfrac{20}{28} * 28 * (R_m^6)(1-R_m)^2 + 8 * (R_m^7)(1-R_m) + R_m^8$$

Valutiamo ora la reliability dopo un periodo di 48 ore, supponendo abbia una distribuzione esponenziale e il failure rate sia 0.005.

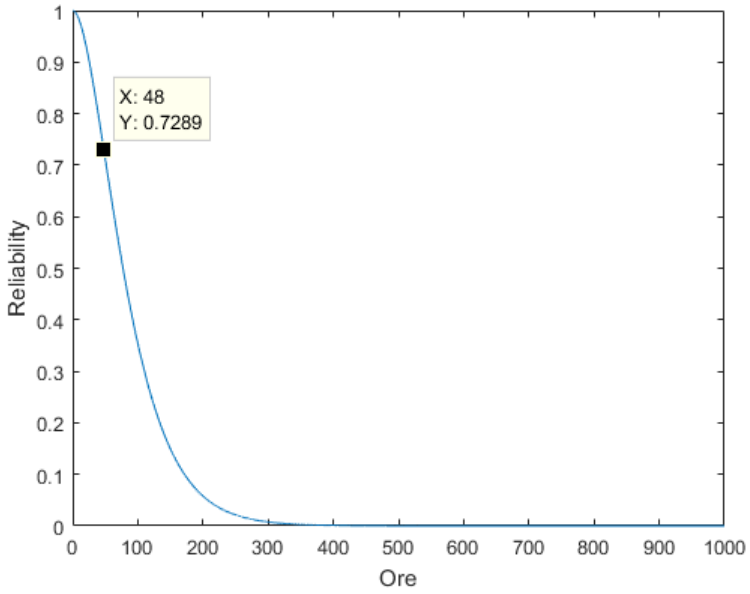

Figura 4.5: Grafico reliability del sistema al variare del tempo con $\lambda = 0.005$

Come si nota dal grafico, dopo 48 ore la reliability è 0.7289.

4.4 Esercizio 4

Un'applicazione richiede che almeno 3 processori di un sistema multiprocessore siano disponibili con più del 99% di probabilità. Il costo di un processore con l'80% di reliability è 1000$ e ogni incremento del 10% in reliability costa 500$. Determinare il numero di processori (n) e la reliability (p) di ogni processore (assumere che tutti i processori abbiano la stessa reliability) che minimizzi il costo totale del sistema.

Dato che si richiede debbano funzionare 3 processori su 3, questo sistema può essere visto come un 3-out-of-N system, la cui reliability sarà:

$R_{sys} = \sum_{i=0}^{N-3} \binom{N}{i} R_m^{N-i}(1 - R_m)^i$

Ora bisogna trovare il numero di processori N che minimizzi il costo e garantisca una reliability di almeno 2 nines.
Per i singoli processori è possibile avere solo 3 valori di reliability incremen-

tandola ogni volta del 10%, ovvero 0.8, 0.88 e 0.96, questo perché non posso avere un valore di reliability superiore a 1.
Usando l'espressione precedente, tramite Matlab, sono stati calcolati i valori di reliability per diverso numero di processori (enumerazione), fino a che non si è raggiunta una reliability di almeno 0.99.
Vediamo i risultati ottenuti, partendo ovviamente da un numero di processori pari a 3.

Tabella 4.2: Valori reliability sys al variare di N e della reliability dei processori

N	**Rm=0.8**	**Rm=0.88**	**Rm=0.96**
3	0.512	0.6815	0.8847
4	0.8192	0.9268	0.9909
5	0.9421	0.9857	
6	0.983	0.9975	
7	0.9953		

Come si nota, con processori aventi 0.8 di reliability, ne sono necessari 7 afinché il sistema abbia una reliability di 2 nines; nel caso di reliability del singolo processore pari a 0.88 ne servono 6 e infine nel caso di reliability pari a 0.96, ne servono solo 2.
Calcoliamo ora i costi in tutti e 3 i casi:

$Costo_{N=7,Rm=0.8} = 1000 * 7 = 7000\$$

$Costo_{N=6,Rm=0.88} = 1000 * 6 + 500 * 6 = 9000\$$

$Costo_{N=2,Rm=0.96} = 1000 * 2 + 500 * 2 + 500 * 2 = 4000\$$

In conclusione, per minimizzare il costo totale del sistema ed ottenere una reliability di 2 nines, abbiamo bisogno di 2 processori su cui si applicano due aumenti di reliability del 10%.

4.5 Esercizio 5

Il sistema descritto nella figura sottostante è un sistema di elaborazione per un elicottero. Il sistema ha processori dual-ridondanti e unità di interfaccia dual-ridondanti. Sono usati due bus nel sistema e entrambi sono dual-ridondanti. La parte interessante del sistema è l'equipaggiamento di navigazione. Il velivolo può essere completamente navigato usando l'Inertial Navigation System (INS). Se l'INS fallisce, il velivolo può essere navigato usando la combinazione del Doppler e dell'Altitude Heading and Reference System (AHRS). Il sistema contiene tre unità AHRS, di cui solo una è necessaria. Questo è un esempio di ridondanza funzionale dove i dati dall'AHRS e dal Doppler possono essere usati per rimpiazzare l'INS, se esso fallisce. A causa di altri sensori e strumenti, entrambi i bus sono necessari affinché il sistema funzioni correttamente indipendentemente dalla modalità di navigazione per cui esso è impiegato.

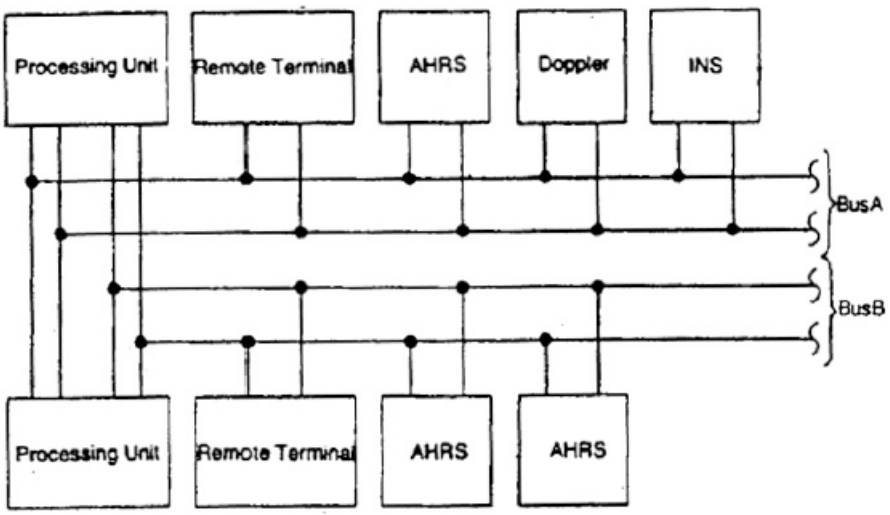

Figura 4.6: Sistema di elaborazione di un elicottero

Di seguito è mostrato il Reliability Block Diagram (RBD) del sistema:

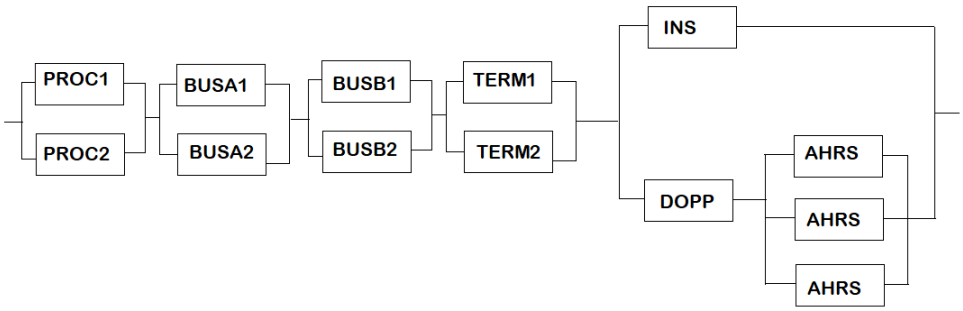

Figura 4.7: Reliability Block Diagram

Di seguito è mostrato il Fault Tree del sistema:

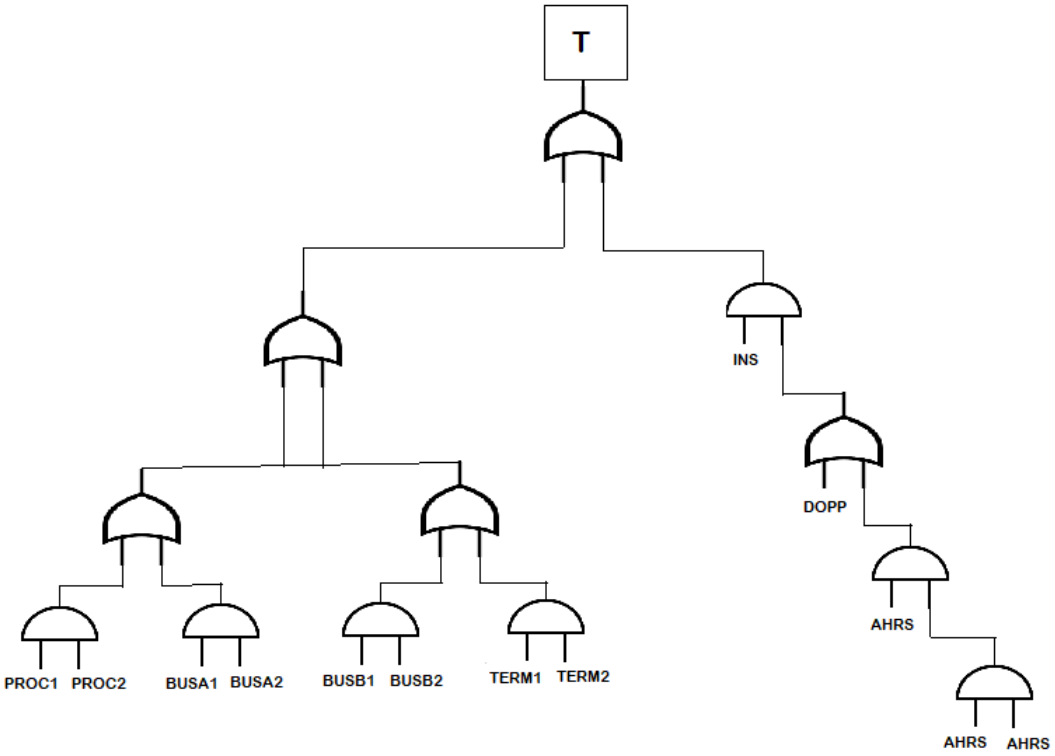

Figura 4.8: Fault Tree

Il minimal cut set sarà:

$$(PROC1 * PROC2) + (BUSA1 * BUSA2) + (BUSB1 * BUSB2) + (TERM1 *$$

$TERM2) + (((AHRS * AHRS * AHRS) + DOPP) * INS)$

Calcoliamo ora la reliability del sistema per un volo di un'ora tramite le formule per le serie e i paralleli:

$R_{ser} = \prod_i R_i(t)$
$R_{par} = 1 - \prod_i (1 - R_i(t))$

$R_{sys} = (1 - (1 - R_{PROC})^2) * (1 - (1 - R_{BUS})^2)^2 * (1 - (1 - R_{TERM})^2) * (1 - (1 - R_{INS})) * (R_{DOPP} * (1 - (1 - R_{AHRS})^3))$

Considerando la reliability come un'esponenziale negativo del tipo $e^{-\lambda t}$ e avendo a disposizione i seguenti valori di MTTF per ogni componente, è possibile calcolare il failure rate di ciascuno e quindi la reliability del sistema ($\lambda = \frac{1}{MTTF}$). Per farlo utilizziamo Matlab.

Equipment	MTTF (hr)
Processing Unit	5000
Remote Terminal	2500
AHRS	1000
INS	1000
Doppler	300
Bus	10000

Figura 4.9: MTTF

Otteniamo i seguenti valori di λ per ogni componente mostrati in ordine come nella tabella precedente:

```
lambda =

    0.0002
    0.0004
    0.0010
    0.0010
    0.0033
    0.0001
```

Figura 4.10: Failure Rate

Otteniamo così un valore di reliability pari a 0.9957, quindi di due nines.

Ripetiamo ora lo stesso procedimento, includendo però un fattore di coverage c per la fault detection e la riconfigurazione delle processing unit, mantenendo, però, una reliability pari a 0.99999.
La formula da considerare che include il fattore di coverage è:

$$R_{MOD} = R_1 + c * (1 - R_1) * R_2$$

c è la probabilità che, quando un fallimento si presenta, esso è rilevato correttamente. R_1 e R_2 nel nostro caso sono entrambe uguali a R_{PROC}, mentre R_{MOD} non è altro che R_{PROC} modificato ovvero il rapporto tra la reliability desiderata (0.99999) e la reliability del sistema senza quella relativa ai processori.
Detto questo calcolo c (tramite un apposito script Matlab), come:

$$c = \frac{R_{PROC\_MOD} - R_{PROC}}{(1 - R_{PROC}) * R_{PROC}} = 22.6711$$

Capitolo 5

Field Failure Data Analysis

L'analisi di dati di fallimento è relativa ai log di errore dei sistemi *Mercury* e *BlueGene/L*. I dati erano già stati sottoposti alla fase di *Filtering*. Si è potuto, quindi, procedere con la *Manipulation*, utilizzando il metodo della *Coalescenza temporale*, ed effettuare, infine, la *Reliability Analysis*.

Per determinare la finestra temporale è stata effettuata un'analisi di sensibilità del numero di tuple rispetto alla dimensione della finestra: sono state conteggiate le tuple al variare della finestra ed è stato trovato il valore ottimo di quest'ultima, alla destra del ginocchio della curva ottenuta. Per il sistema Mercury si è scelta una finestra temporale di 400s, mentre per BlueGene di 200s.

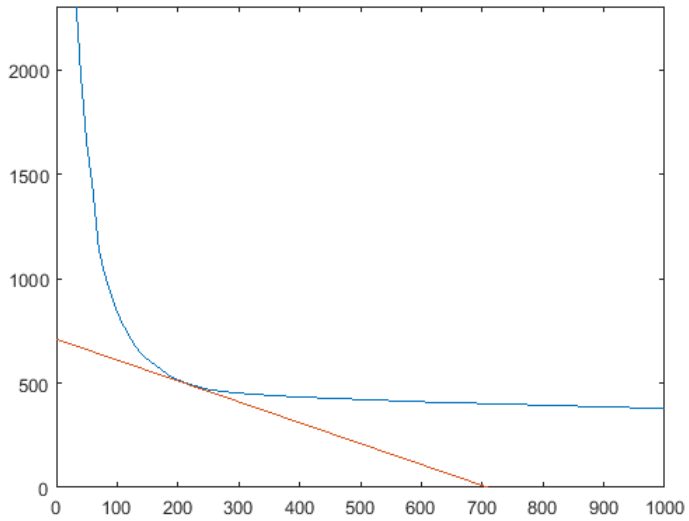

Figura 5.1: Sensitivity Analysis di Mercury

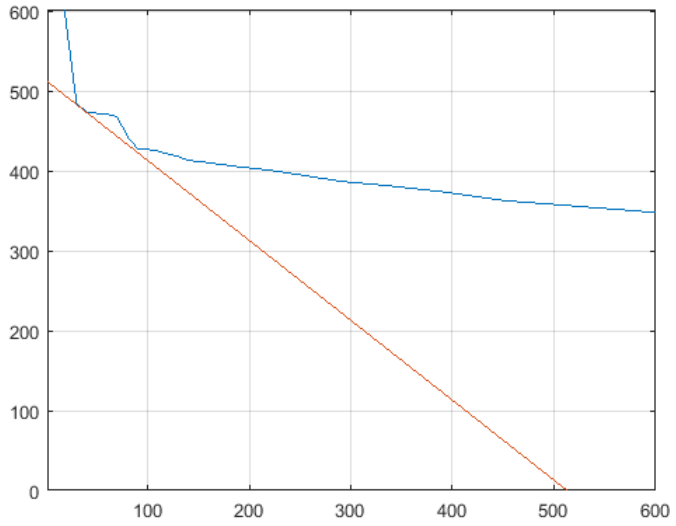

Figura 5.2: Sensitivity Analysis di BlueGene/L

Effettuanto il tupling, si ottengono 432 tuple per Mercury, 403 tuple per BlueGene. Sono presenti troncamenti e collisioni, che abbiamo provato ad individuare analizzando i file *interarrivals* e *lenghts*. Una tupla molto lunga può presentare collisioni al suo interno. Analogamente, tempi troppo brevi tra tuple possono essere un indizio di troncamento. In tal modo in Mercury si è individuato un troncamento tra le tuple 3 e 4, distanti $457s$, di un errore DEV nel nodo tg-c238, ed una collisione nella tupla 264 di errori dei nodi tg-c401 e tg-c027. In BlueGene osserviamo una collisione nella tupla 289 di errori di rack diversi.

Mediante il file interarrivals, sono state calcolate le distribuzioni empiriche di TTF e Reliability dei due sistemi. Possiamo osservare come la curva della Reliability di BlueGene abbia un andamento più dolce, garantendo maggiore affidabilità.

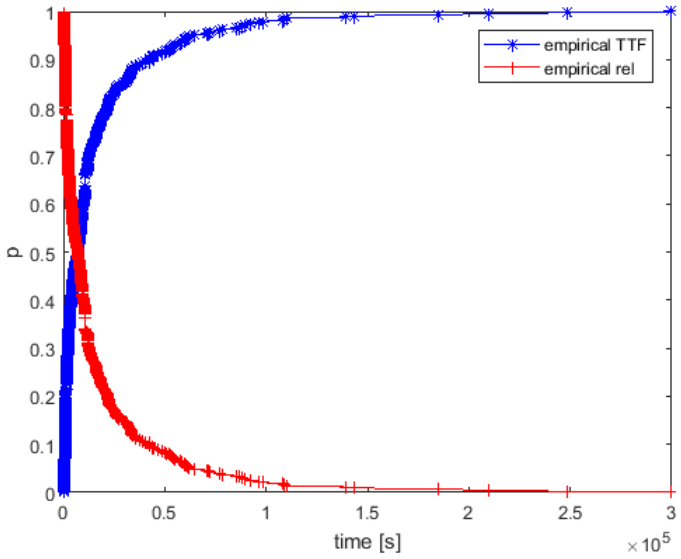

Figura 5.3: Reliability empirica e TTF empirica di Mercury

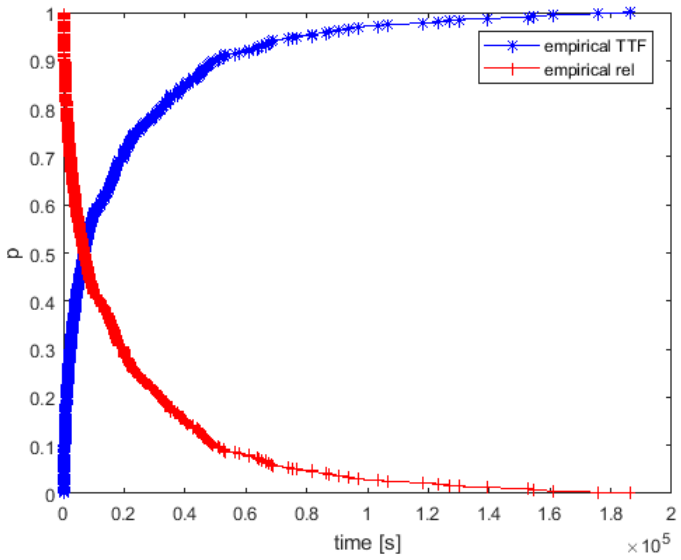

Figura 5.4: Reliability empirica e TTF empirica di BlueGene/L

Le curve empiriche sono state utilizzate per stimare i parametri delle distribuzioni teoriche. Sono state provate le distribuzioni esponenziale e di Weibull ed, in entrambi i casi, la seconda è risultata più aderente ai dati.

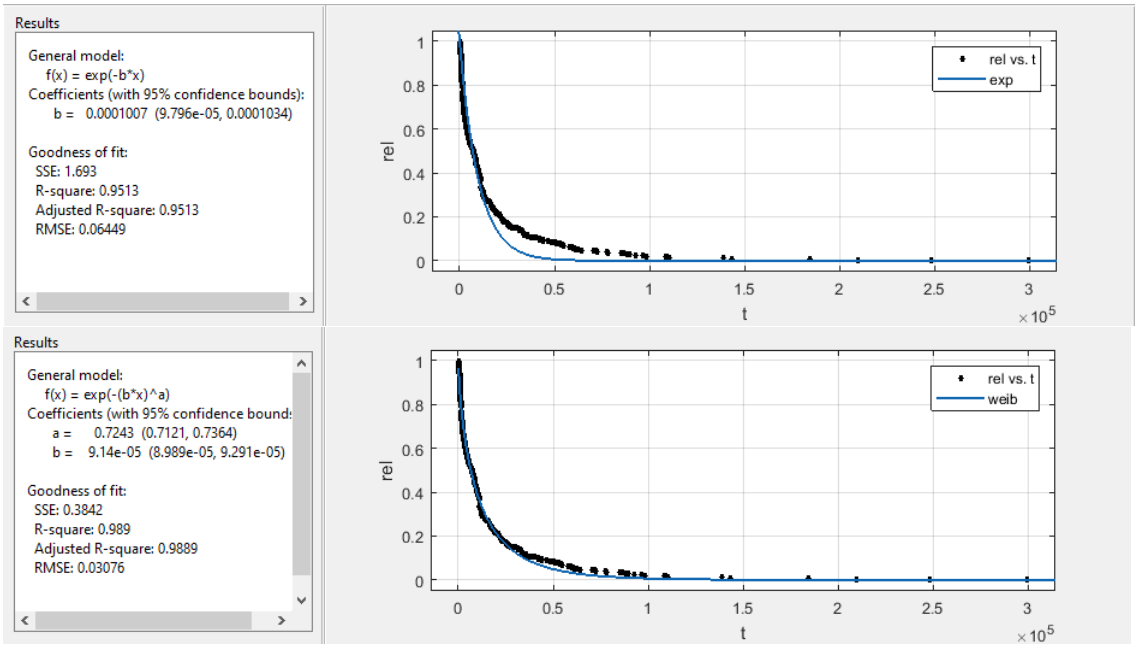

Figura 5.5: Fitting della Reliability empirica in Mercury

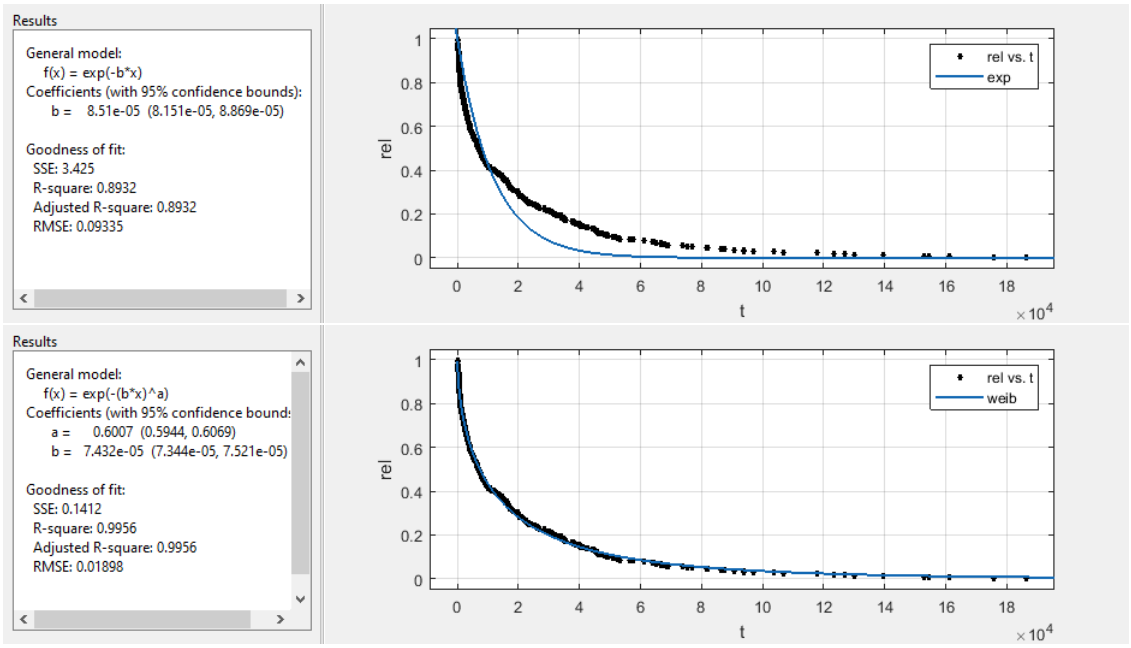

Figura 5.6: Fitting della Reliability empirica in BlueGene

La bontà del fitting è stata valutata con un test *GOF*, in particolare il

Tabella 5.1: Kolmogorov-Smirnov in Mercury

Distribuzione	Ipotesi	P-value	K-statistic
Esponenziale	Rigettata	0.0081	0.1152
Weibull	Accettata	0.0657	0.0907

Tabella 5.2: Kolmogorov-Smirnov in BlueGene

Distribuzione	Ipotesi	P-value	K-statistic
Esponenziale	Rigettata	0.00060403	0.1421
Weibull	Accettata	0.2631	0.0711

test di *Kolmogorov-Smirnov* con livello di significatività pari a 0.05. L'ipotesi che i dati seguano la distribuzione specifica è rigettata se $p < 0.05$.

Assumendo, quindi, distribuzione di Weibull per entrambi i sistemi, possiamo calcolare MTTF (**Tabella 5.3**).

5.1 Analisi a livello di nodo

Prendiamo in analisi i primi 5 nodi con più occorrenze nei log dei due sistemi. Nel caso del sistema Mercury essi sono:

- `tg-c401` con 62340 occorrenze
- `tg-master` con 4098 occorrenze
- `tg-c572` con 4030 occorrenze
- `tg-s044` con 3224 occorrenze
- `tg-c238` con 1273 occorrenze

Si filtrano i dati di log ad essi relativi per effettuare l'analisi di sensibilità e della reliability. Il procedimento è del tutto analogo a quello precedentemente seguito. Si riportano i risultati di seguito.

Tabella 5.3: MTTF

Sistema	MTTF [h]
Mercury	3.7281
BlueGene/L	5.6154

Tabella 5.4: Coalescence Window dei top-5 entries-prone nodes di Mercury

Nodo	Window Size [s]	Tuple Count
tg-c401	300	40
tg-master	150	63
tg-c572	10	2
tg-s044	10	13
tg-c238	150	24

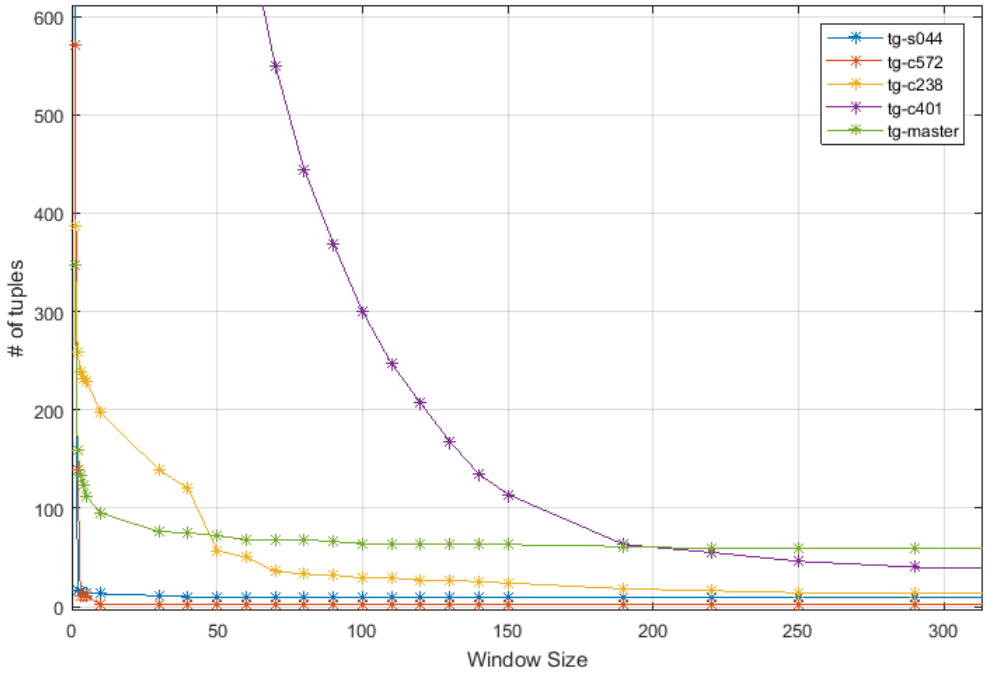

Figura 5.7: Sensitivity Analysis dei top-5 entries-prone nodes di Mercury

La dimensione delle finestre varia a seconda del nodo. Il nodo che funge da *dependability-bottleneck* è tg-master, che ha ruolo di gestore nel sistema. Analizziamo i nodi con maggior numero di tuple.

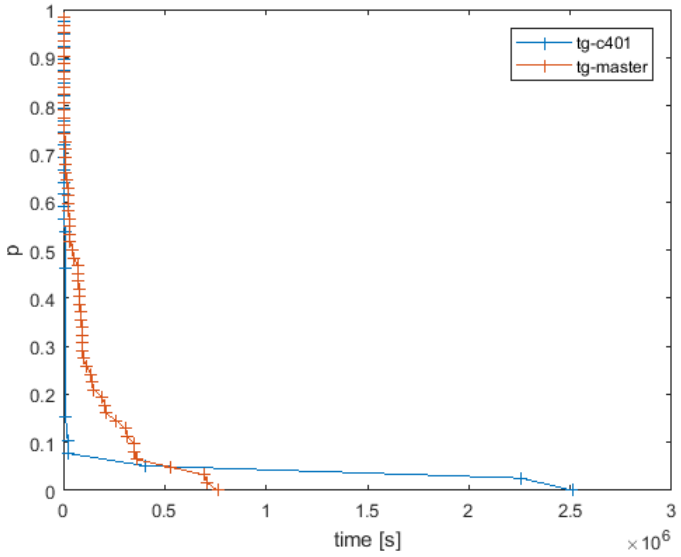

Figura 5.8: Empirical Reliability degli entries-prone nodes di Mercury

La Reliability di tg-master è superiore fino a $0.5 * 10^6 s$, dopo il quale presenta una discesa molto più ripida di tg-c401, nodo di calcolo. Si ipotizza che il gestore debba avere una grande affidabilità, essendo un *Single Point of Failure*, ma che sia soggetto ad una maggiore attenzione relativa alla manutenzione. Sembra quindi ragionevole che il tempo entro cui l'affidabilità è elevata sia il suo *mission time*.

Nel caso del sistema BlueGene/L i 5 nodi con più occorrenze nel log sono:

- `R71-M0-N4` con 1716 occorrenze
- `R12-M0-N0` con 1563 occorrenze
- `R63-M0-N2` con 976 occorrenze
- `R03-M1-NF` con 960 occorrenze
- `R63-M0-N0` con 791 occorrenze

Tabella 5.5: Coalescence Window dei top-5 entries-prone nodes di BlueGene/l

Nodo	Window Size [s]	Tuple Count
R71-M0-N4	10	100
R12-M0-N0	10	116
R63-M0-N2	10	7
R03-M1-NF	10	8
R63-M0-N0	10	151

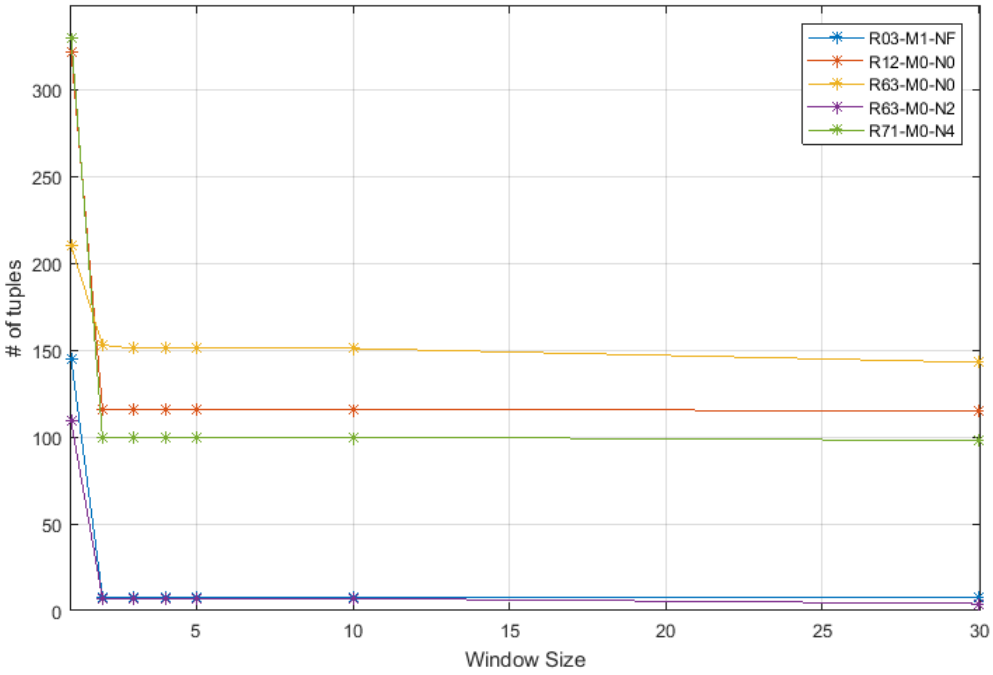

Figura 5.9: Sensitivity Analysis dei top-5 entries-prone nodes di BlueGene/L

In BlueGene la finestra temporale è la stessa per tutti i nodi (**Tabella 5.5**). Il numero di tuple, infatti, è molto sensibile fino ad una dimensione di circa 3s della finestra temporale, dopo la quale si stabilizza. Sono anche presenti dei *dependability-bottleneck* con un grande numero di tuple: R71-M0-N4, R12-M0-N0, R63-M0-N0. Da notare che essi sono tutti nodi con la Compute Card di IO aggiuntiva. Presentano un andamento della Reliability simile, anche se risultano comunque essere più affidabili i rack con minori tuple (**Figura 5.10**).

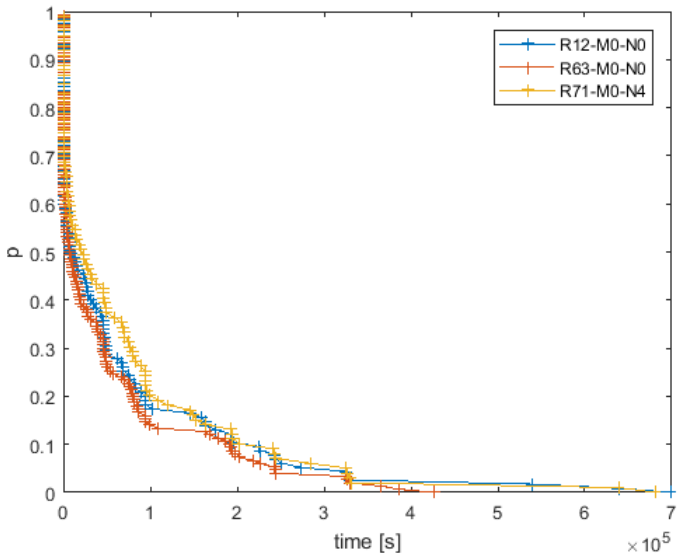

Figura 5.10: Empirical Reliability degli entries-prone nodes di BlueGene/L

5.1.1 Nodi funzionalmente simili

Possiamo concentrarci su nodi con funzionalità simili. In Mercury tg-c401, tg-c572, tg-c238, tg-c242 sono tutti nodi di calcolo. I primi tre sono stati analizzati anche precedentemente (**Tabella 5.4**). L'ultimo nodo ha una finestra di 90s e conta 66 tuple. Essi hanno un numero di tuple ampiamente differente. Consideriamo, nello studio della Reliability, solo i nodi con più tuple. In **Figura 5.11** possiamo osservare come i nodi di calcolo presentino un andamento della Reliability comune, che tende prima ad abbattersi velocemente, per poi tendere lentamente a 0.

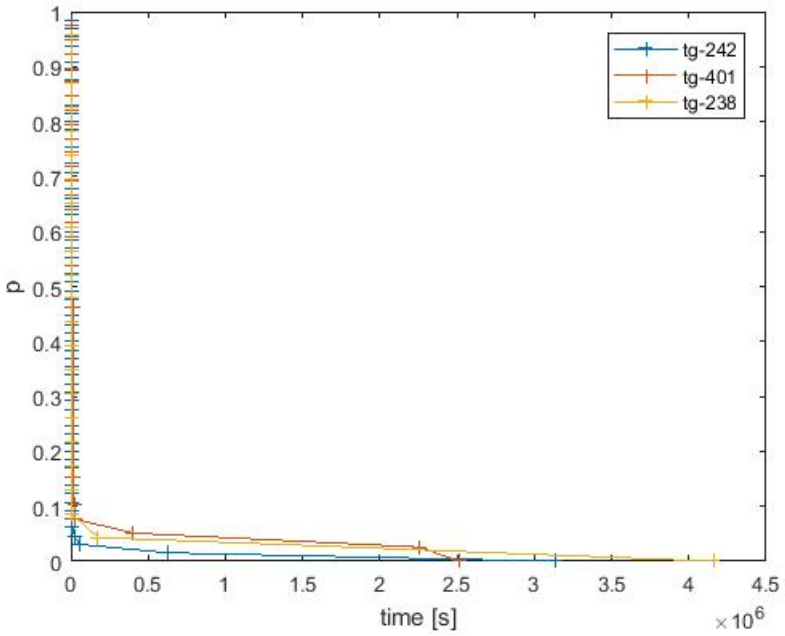

Figura 5.11: Empirical Reliability dei nodi di calcolo di Mercury

In BlueGene/L sono nodi funzionalmente simili R71-M0-N4, R12-M0-N0, R63-M0-N0, poiché tutti di IO. Essi, come già osservato, presentano tutti un grande numero di tuple con la stessa finestra temporale (**Tabella 5.5**). Anche la relativa Reliability presenta un andamento simile (**Figura 5.10**).

5.2 Analisi a livello di errore

Possiamo procedere allo stesso modo per le categorie di errore dei nodi di Mercury e le Compute Unit di BlueGene/L.

In Mercury le categorie di errore più frequenti sono:

- **DEV**
- **MEM**
- **IO**
- **NET**
- **PRO**

Tabella 5.6: Coalescence Window delle categorie di errore di Mercury

Errore	Window Size [s]	Tuple Count
DEV	350	284
MEM	350	68
IO	70	112
NET	80	79
PRO	100	73

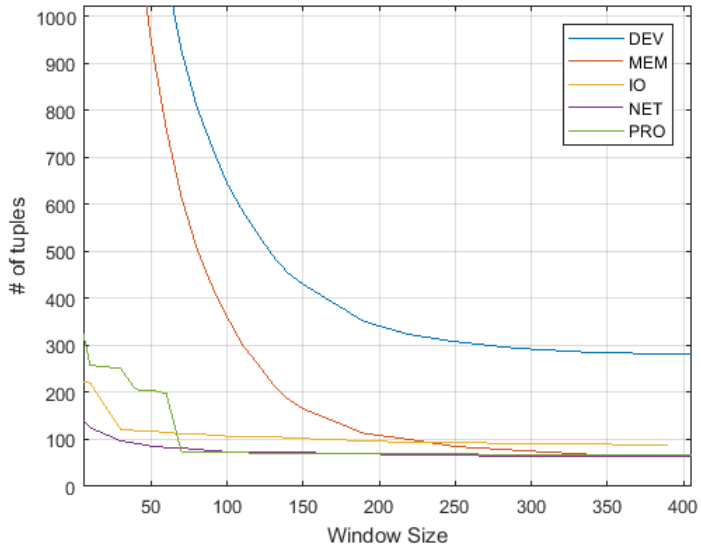

Figura 5.12: Sensitivity Analysis delle categorie di errore di Mercury

Gli errori con più tuple sono relativi all'interconnessione ed alle operazioni di IO.

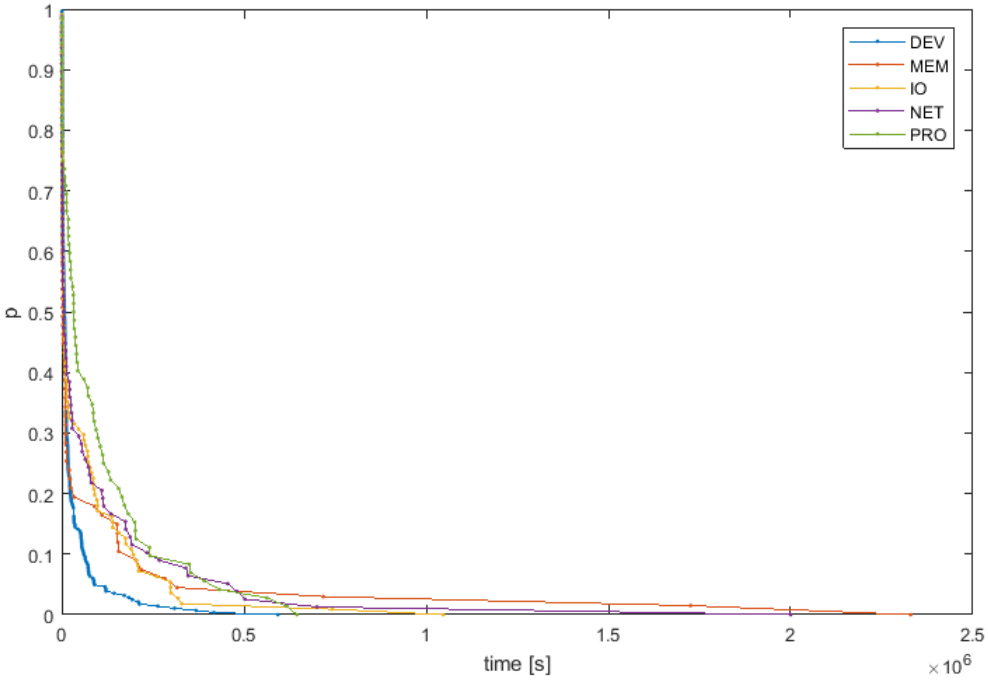

Figura 5.13: Empirical Reliability delle categorie di errore di Mercury

L'interconnessione risulta essere una risorsa critica con una Reliability che si abbatte più velocemente delle altre.

Si analizzano le Compute Unit relative all'IO di BlueGene/L:

- `J18-U11`
- `J18-U01`

Entrambe le unità hanno un alto numero di tuple e la stessa dimensione per la finestra temporale. Il numero di tuple presenta, infatti, andamento simile, al variare della dimensione della finestra. Ciò è comprensibile anche alla luce del fatto che sono entrambe unità facenti parte di Compute Card di IO.

Tabella 5.7: Coalescence Window delle Compute Unit di BlueGene/L

Compute Unit	Window Size [s]	Tuple Count
J18-U11	100	296
J18-U01	100	256

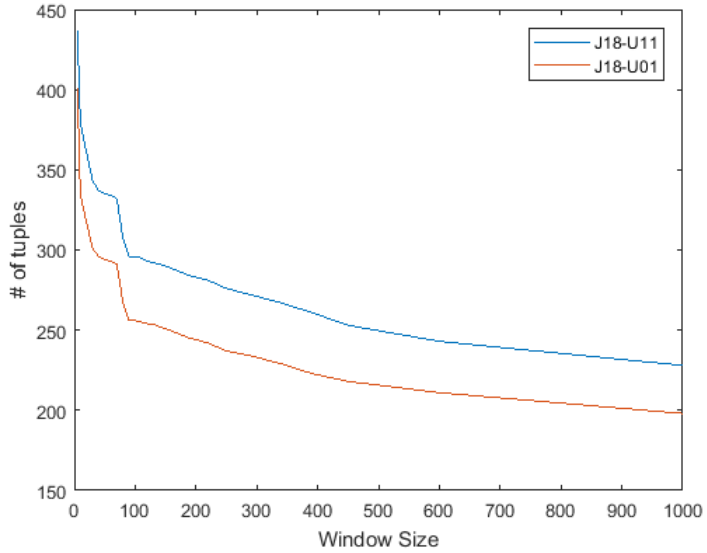

Figura 5.14: Sensitivity Analysis delle Compute Unit di BlueGene/L

Tabella 5.8: Nodi più ricorrenti per categoria di errore in Mercury

	DEV	MEM	IO	NET	PRO
1	tg-c401 50782	tg-c401 11558	tg-s044 3208	tg-master 3639	tg-c648 616
2	tg-c572 3176	tg-c572 845	tg-master 452	tg-c550 33	tg-c324 239
3	tg-c238 1071	tg-c238 197	tg-login3 381	tg-c196 14	tg-c284 180
4	tg-c242 918	tg-c242 149	tg-s038 230	tg-c238 3	tg-c451 159
5	tg-c117 263	tg-c894 28	tg-c550 220	tg-s044 2	tg-c447 136

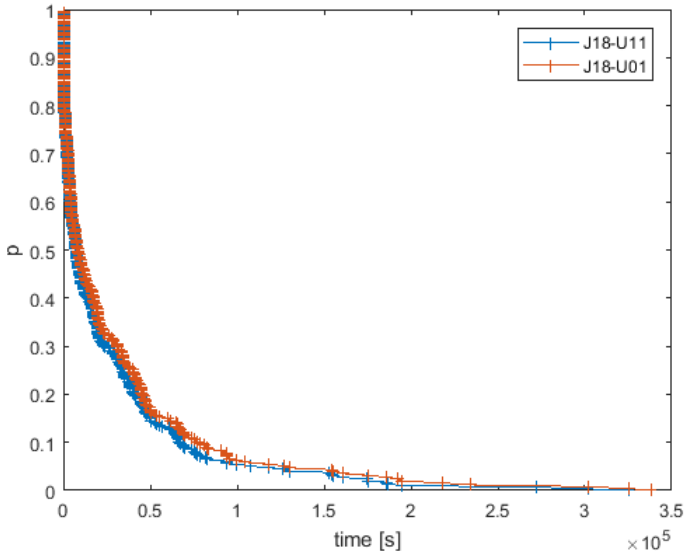

Figura 5.15: Empirical Reliability delle Compute Unit di BlueGene/L

Analoga è anche la curva della Reliability, comunque leggermente superiore per l'unità U01 con meno tuple.

5.3 Analisi combinata

In Mercury estraiamo i 5 nodi più ricorrenti dai log di ogni categoria di errore. In **Tabella 5.8** possiamo notare come i nodi di calcolo siano la causa esclusiva di errori di tipo DEV, MEM e PRO. In particolare, i nodi fonte di errori DEV e MEM sono gli stessi, il che può far sospettare ad una comunanza di fault. Come ci si poteva aspettare, gli errori di IO sono causati principalmente da nodi di storage, di login e dal master. Infine gli errori NET di rete sono quasi totalmente ascrivibili al nodo master.

Tabella 5.9: Rack con maggiori eventi di errore di BlueGene/L

	Rack	**Occorrenze**
1	R63	7192
2	R62	4057
3	R57	3581
4	R56	3527
5	R46	3485

In BlueGene/L siamo interessati ai rack con maggior numero di occorrenze nel log. Per fare ciò sono state omesse le informazioni sul midplane e sul nodo. In **Tabella 5.9** osserviamo che il rack con più eventi di errore è R63. Precedentemente abbiamo riscontrato che i nodi N0 e N2 di questo rack fanno anche parte della top-5 entries-prone nodes. I rack che abbiamo trovato sono tutti vicini tra loro. Sembra ragionevole, nel calcolo scientifico, parallelizzare i calcoli, assegnandoli a nodi vicini, fino a coinvolgere interi rack. Può essere plausibile, dunque, una relazione tra l'elevato numero di eventi di errore che essi presentano.