Università degli Studi di Napoli "Federico II"

Scuola Politecnica e delle Scienze di Base
Corso di Laurea Magistrale in Ingegneria Informatica

Giuseppe Francesco Di Cecio - M63001211
Nicola D'Ambra - M63001223
Emma Melluso - M63001176

Elaborato di *Impianti di Elaborazione*

Università degli Studi di Napoli *Federico II*
Napoli
A.A 2020/2021

Indice

Capitolo 1

Benchmark

L'obiettivo dell'esercizio è quello di confrontare due sistemi con lo stesso sistema operativi, ma processori differenti, utilizzando il benchmark *Nbody*. Esso simula l'evoluzione di N corpi celesti, sotto l'influenza della forza di gravità ed è molto utile per confrontare le prestazioni tra i due sistemi, dato che stressa:

- **CPU**
- **Sottosistemi Floating-Point**
- **Chiamate ricorsive**

La complessità dell'algoritmo che lo implementa è un $O(n^2)$.

1.1 Sistemi

Il confronto avviene tramite due macchine virtuali, su architetture diverse. Ogni macchina virtuale però è stata dotata delle stesse caratteristiche e prestazioni, in modo da ridurre il più possibile l'eterogeneità tra i due sistemi fisici. L'unico fattore su cui non si può agire è il processore.
Per rendere il confronto quanto il più possibile affidabile è stata usata anche la stessa versione dell'hypervisor (in questo caso VirtualBox).

Sistema 1	
CPU	Intel Core i7-7500U 7th Generation - 2.70 GHz
Memoria Centrale	2 GB
Hard-Disk	25 GB - SSD
Memoria Video	16 MB
OS	Xubuntu 20.04 LTS - 64bit

Sistema 2	
CPU	Intel Core i5-5700U 5th Generation - 2.20 GHz
Memoria Centrale	2 GB
Hard-Disk	25 GB - SSD
Memoria Video	16 MB
OS	Xubuntu 20.04 LTS - 64bit

Ogni macchina virtuale è stata dotata di 2 processori.

1.2 Test e Risultati

Su entrambi i sistemi è stato avviato lo script *launch_ nbody.sh*, con i seguenti parametri di input:

```
./launch_nbody.sh -r 25 -n N
```

Esso non fa altro che lanciare l'eseguibile *nbodySim* per un numero di ripetizioni impostato a 25. *nbodySim* esegue sul sistema l'algoritmo che implementa il benchmark in questione.
Lo script è stato eseguito facendo variare di volta in volta N:

$$N = \begin{bmatrix} 10 & 100 & 500 & 1.000 & 5.000 & 10.000 & 50.000 & 100.000 & 500.000 & 1.000.000 \end{bmatrix}$$

L'output fornito da *nbodySim* consiste nel tempo d'esecuzione (in microsecondi) impiegato dal sistema per eseguire l'algoritmo. Dunque per ogni valore di N sono state effettuate 25 misurazioni, ciascuna delle quali è stata collezionata in un file *.csv* per poter poi essere processata attraverso il seguente script MATLAB

```
%% Read
N = [10 100 500 1000 5000 10000 50000 100000 500000 1000000];
sys1 = zeros(1, length(N));
sys2 = zeros(1, length(N));
index = 1;

for i=N
        path1 = strcat('Emma/data',num2str(i, '%d'),'.csv');
        path2 = strcat('Peppe/data',num2str(i, '%d'),'.csv');
        samples = readtable(path1);
        samples = table2array(samples(:,2));
        sys1(index) = mean(samples);
        samples = readtable(path2);
        samples = table2array(samples(:,2));
        sys2(index) = mean(samples);
        index = index +1;
end

%% Grafico
figure;
loglog(N,sys1./1000,'LineWidth', 2);
hold on;
loglog(N,sys2./1000,'LineWidth', 2);
grid;
legend('SYS 1', 'SYS 2');
title('Grafico in scala logaritmica');
xlabel('Dimensione dei dati');
ylabel('Tempo di esecuzione [ms]');

figure;
semilogx(N,sys1,'LineWidth', 2);
hold on;
semilogx(N,sys2,'LineWidth', 2);
grid;
legend('SYS 1', 'SYS 2');
title('Grafico in scala normale');
xlabel('Dimensione dei dati');
ylabel('Tempo di esecuzione [ms]');
```

Il quale fa una media dei 25 tempi di risposta e ne plotta il grafico, al variare di N, sia in scala normale che in scala logaritmica per evidenziare nel dettaglio le differenze tra i due sistemi.

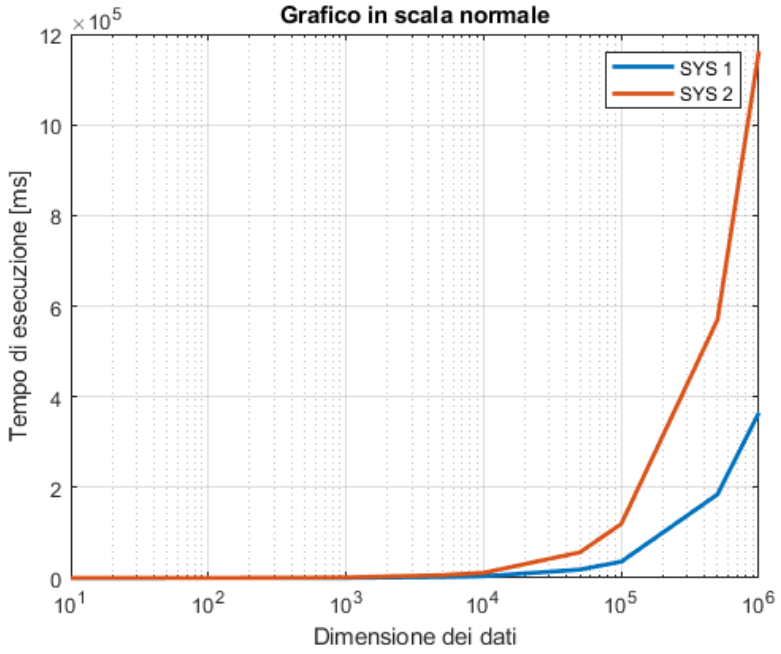


Figura 1.1: *Confronto andamento tempi di risposta sys1 e sys2*

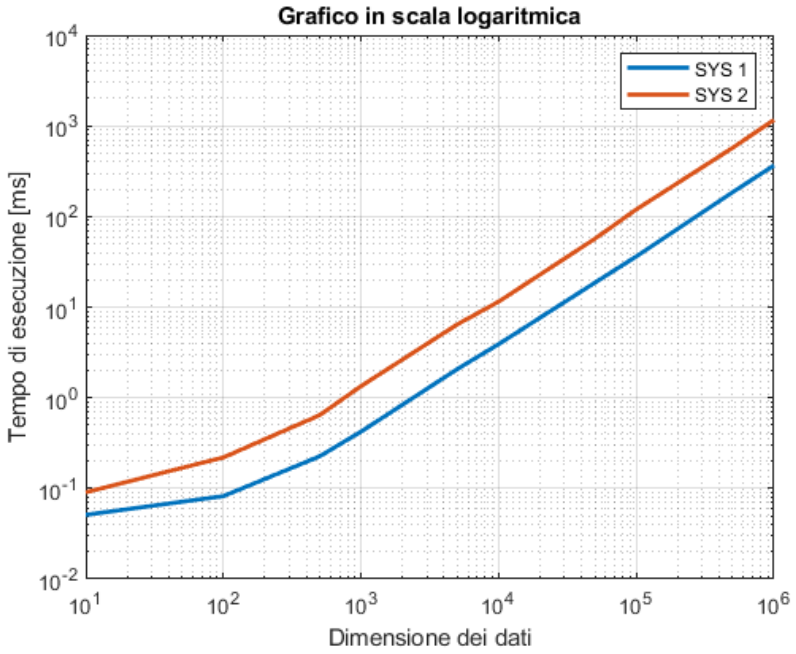


Figura 1.2: *Confronto andamento tempi di risposta - scala logaritmica*

Quindi il primo sistema è con evidenza quello più performante. Le differenze si percepiscono a vista d'occhio già a partire da $N > 10^4$.

Il motivo principale di tale differenza è proprio l'eterogeneità dei processori. Per quanto i due sistemi possano essere simili, il sistema più performante (Sistema 1) è dotato infatti non solo di un processore di architettura più avanzata, ma anche di alcune generazioni successive.

Capitolo 2

Workload Characterization

Il dataset di partenza è composto da **3000 righe** e **24 colonne**, ciascuna delle quali rappresenta uno dei parametri del sistema oggetto di studio. Si tratta di parametri caratterizzanti l'esecuzione di vari Threads su un sistema operativo.
In particolar modo le colonne con prefisso Vm rappresentano informazioni sulla memoria virtuale occupata e utilizzata dai Threads, mentre le altre colonne rappresentano informazioni di carattere generale, come memoria libera, numero di threads, pagine inattive ecc.

2.1 Filtraggio

2.1.1 Colonne Identiche

Innanzitutto osservando il workload e effettuando un grafico delle distribuzioni è possibile rendersi conto della presenza di ben 4 colonne costanti:

- **Active**
- **AnonPages**
- **AvbLatency**
- **Error**

Tali colonne in quanto costanti non spiegano varianza, dunque possono essere tranquillamente trascurate ai fini dell'analisi.
Osservando le distribuzioni dei parametri **WriteBack** e **MemFree** si sono notate alcune caratteristiche comuni. Per avere una maggiore chiarezza si è preferito calcolare la matrice delle correlazioni su questi due parametri.

	'MemFree'	'Writeback'
'MemFree'	1,0000	1,0000
'Writeback'	1,0000	1,0000

Figura 2.1: *Matrice di correlazione tra MemFree e WriteBack*

Osservando la matrice appare evidente che le due colonne sono esattamente identiche, fornendo quindi la stessa informazione. Per questo motivo si è deciso di trascurare una delle due, in particolare quella di WriteBack.

Le 24 colonne iniziali sono state ridotte a 19 colonne, riducendo il dataset di un numero di osservazioni pari a:

$$n_{dati} = 3.000 \times (24 - 19) = 15.000 \tag{2.1}$$

2.1.2 Outlier

Gli outliers sono valori anomali all'interno dell'insieme di osservazioni, in altre parole sono valori che si discostano notevolmente dagli altri valori dell'insieme. Essendo valori anomali la loro frequenza di occorrenza è bassa rispetto agli altri valori e ciò li porta ad essere identificati all'esterno del range interquartile. In alcuni casi gli outlier possono essere eliminati ma ciò è possibile solo a monte di una analisi accurata. Tali valori infatti influiscono sulle analisi statistiche in modo considerevole e non sempre rappresentano situazioni trascurabili per l'analisi da svolgere . Osservando attraverso box plot e grafici di distribuzione l'andamento dei seguenti parametri :

- **VmSize** (quanta memoria virtuale utilizza l'intero processo);
- **VmHWM** (di quanta RAM il processo necessita al massimo);
- **VmRSS** (quanta RAM il processo sta correntemente usando);
- **VmPTE** (quanta memoria Kernel è occupata dalle entries della tabella delle pagine);

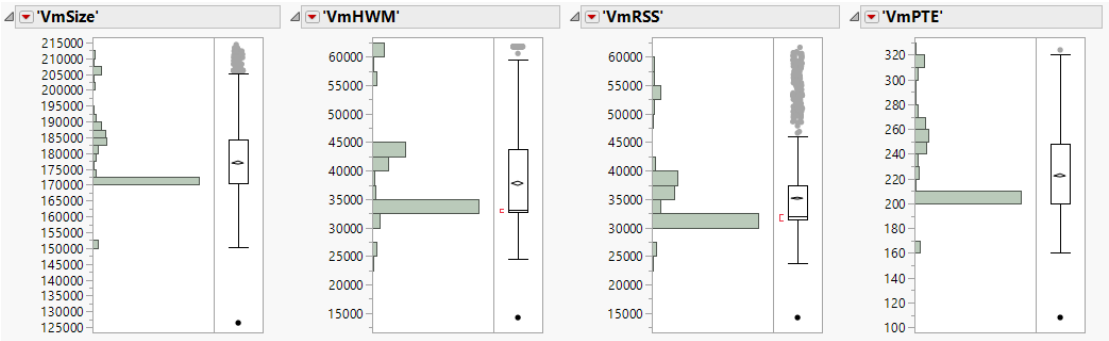


Figura 2.2: *Grafici di distribuzione di VmSize, VmHWM, VmRSS, VmPTE*

Si è notato che essi presentano un outlier isolato (in basso ad ogni grafico) in comune associato alla prima riga del dataset. Analizzando gli altri parametri (*MemFree*, *Dirty*, *PageTables*, *Buffer*, ...) è stato possibile evidenziare che anche per la maggior parte di essi lo è, ma non è un punto isolato.
L'ipotesi fatta è che con molta probabilità le prime righe del dataset (da 0 a 100 circa), rappresentano la fase di avvio del processo e l'outlier oggetto di studio è la prima istanza di questa fase. Dato che l'obiettivo della caratterizzazione del workload è quello di analizzare le prestazioni a regime del sistema oggetto di studio (in questo caso), si è deciso di trascurare quel singolo outlier. Tuttavia è bene notare che in ogni caso le informazioni riguardo questa fase di avvio non saranno del tutto perse dato che è stato rimosso un singolo punto e non tutti i punti che la rappresentano.

Un secondo outlier che può essere agevolmente rimosso è la riga 512 in cui il parametro **Slab** assume valore 4.

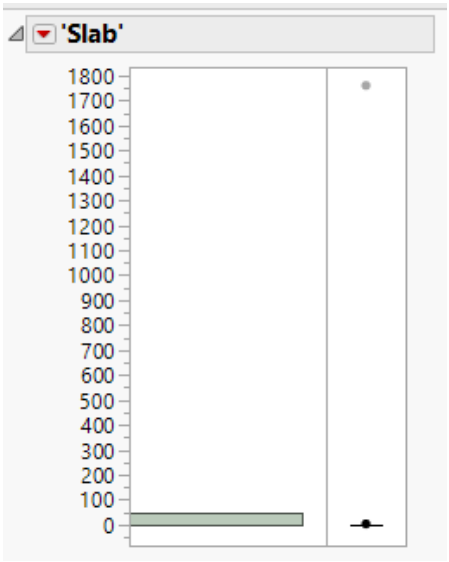


Figura 2.3: *Grafico di distribuzione di Slab*

Oltre ad avvicinarsi molto al valore medio assunto da Slab (zero), esso risulta essere un outlier solo per il parametro stesso dato che per gli altri è un valore compreso tra i quartili. Una sua rimozione quindi non influenza gli indici di caratterizzazione sintetica dei parametri del workload complessivo.

Un terzo outlier è il valore 1760 del parametro *Slab*, associato alla riga 90 del workload.

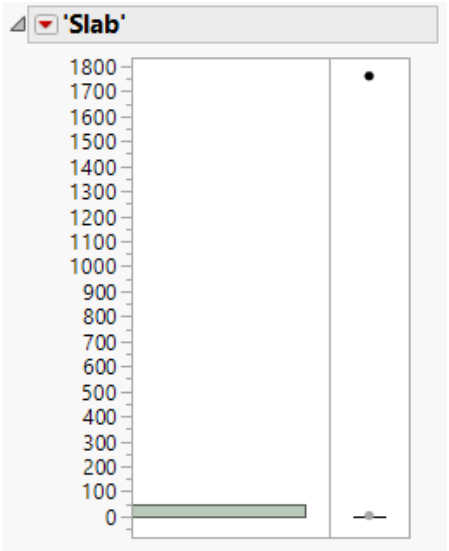


Figura 2.4: *Grafico di distribuzione di Slab*

Rispetto al precedente, tale outlier richiede un' analisi più approfondita visto che influenza significativamente l'andamento di parametri quali *Mapped* e *PageTables*. Per

descrivere meglio la dipendenza tra questi parametri si può effettuare un grafico tra il numero dell'osservazione e il valore assunto da *Mapped*, analogo discorso con *PageTables*.

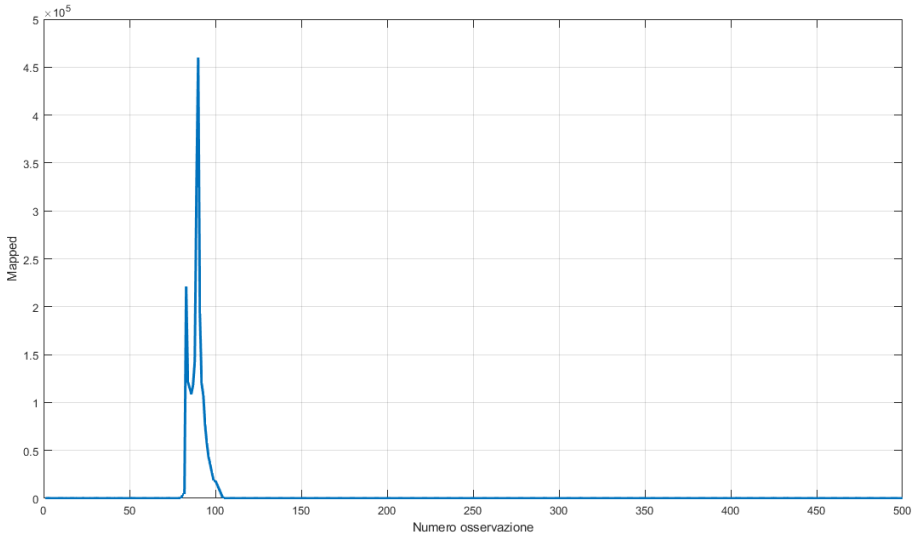


Figura 2.5: *Grafico tra numero di osservazione e valore assunto dal parametro Mapped*

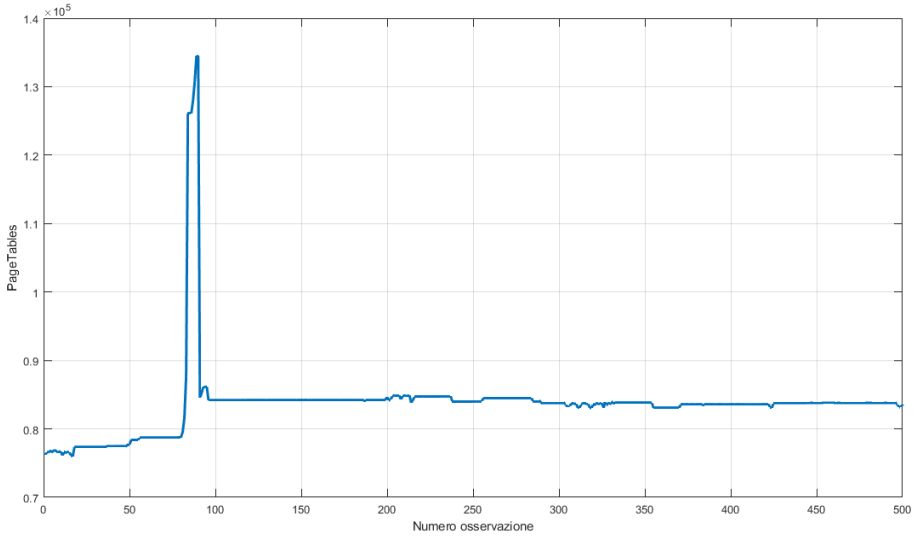


Figura 2.6: *Grafico tra numero di osservazione e valore assunto dal parametro PageTables*

Si nota che in corrispondenza (in realtà nell'osservazione appena precedente) dell'outlier del parametro *Slab* i due parametri sopra indicati hanno un picco, durante la fase di avvio del sistema.
Lo Slab si riferisce ad un particolare meccanismo di allocazione/deallocazione della memoria nel Kernel. Dato che influenza in particolar modo altri parametri si è preferito non trascurarlo.
In conclusione sono stati eliminati dal dataset solo 2 outlier che corrispondono a 38 osservazioni. Quindi il dataset è stato ridotto in totale di 15.038 elementi, provocando una

diminuzione dei dati iniziali di poco più del 20%.

2.2 PCA

A seguito del filtraggio il dataset risulta ridotto grazie alla rimozione di alcune colonne che non esprimevano varianza e alcune righe rappresentanti outlier trascurabili.
Su questo dataset si possono quindi iniziare a fare le prime considerazioni.
Utilizzando la tecnica della *Principal Component Analysis* il dataset può essere estremamente ridotto, sfruttando solo le *Componenti Principali* che mantengono più varianza.
Il risultato della PCA è quindi:

Numero	Autovalore	Percentuale	20 40 60 80	Percentuale cumulativa
1	9,8395	51,787		51,787
2	3,6873	19,407		71,194
3	2,4576	12,935		84,129
4	0,7782	4,096		88,225
5	0,7075	3,724		91,949
6	0,6258	3,293		95,242
7	0,3409	1,794		97,036
8	0,2658	1,399		98,435
9	0,1612	0,848		99,283
10	0,0904	0,476		99,759
11	0,0178	0,094		99,853
12	0,0114	0,060		99,913
13	0,0078	0,041		99,954
14	0,0058	0,030		99,985
15	0,0014	0,007		99,992
16	0,0009	0,005		99,997
17	0,0006	0,003		100,000
18	0,0000	0,000		100,000
19	0,0000	0,000		100,000

Figura 2.7: *PCA applicata al dataset filtrato*

La scelta del numero di componenti principali ricade in particolar modo sulla devianza che quelle componenti mantengono rispetto al dataset reale. Inoltre essa dipende anche dal tipo di osservazioni ed esperimento che è stato effettuato.
In questo caso la scelta è ricaduta sul prendere 5 componenti principali poiché rappresentano il 92% della devianza totale. Esso rappresenta un valore abbastanza elevato, ma è stato scelto per mantenersi in una regione di tolleranza durante la clusterizzazione.
Anche se il workload sintentico verrà costruito considerando 5 componenti principali, in seguito sono riportati i risultati di PCA e clusterizzazione anche nel caso in cui fosse stato scelto un numero diverso di componenti principali, ovvero:

- Prendere 4 PC
$$DEV_{PCA-MANTENUTA} \approx 88\%$$
- Prendere 5 PC
$$DEV_{PCA-MANTENUTA} \approx 92\%$$
- Prendere 6 PC
$$DEV_{PCA-MANTENUTA} \approx 95\%$$

Per ognuno di questi 3 insiemi di Principal Components è stata effettuata la procedura di clustering .

2.3 Clustering

Il clustering è una tecnica che consiste nel raggruppare osservazioni "simili" tra loro. La similitudine tra un elemento e un cluster, o tra un cluster e un altro cluster, può essere calcolata secondo varie tecniche. In questa analisi si è preferito utilizzare il **metodo di Ward**, il quale pesa la distanza tra due cluster in relazione al numero di elementi che li compongono.
Dati due cluster P e Q (un elemento non appartenente ad un cluster, può essere visto come un cluster di dimensione 1), sia $|P|$ la cardinalità di P, analogo con $|Q|$, e sia $\bar{x}_p$ il centroide di P, analogo con $\bar{x}_q$, la distanza tra P e Q viene calcolata come:

$$d(P,Q) = 2\frac{|P|\,|Q|}{|P|+|Q|}||\bar{x}_p - \bar{x}_q||^2$$

Per ogni raggruppamento viene poi scelto una singola osservazione che la rappresenta, riducendo quindi il numero di righe del dataset pari al numero di cluster scelti durante l'analisi.
Il numero di cluster da scegliere può dipendere da vari fattori:

- Omogeneità dei cluster: i cluster devono raggruppare un numero di osservazioni quanto il più possibile omogeneo rispetto agli altri cluster. Avere un cluster con un numero di elementi di vari ordini di grandezza rispetto ad un altro cluster non sempre può portare a buoni risultati (in termini di devianza).
- Devianza mantenuta: a seguito della PCA parte della devianza nei dati viene persa. Dato che il clustering viene effettuato sulle *Componenti Principali* allora esso produce un'ulteriore perdita di devianza nel risultato finale.

Devianza Persa

Per effettuare il calcolo della devianza totale persa persa bisogna prima calcolare la devianza intra-cluster (la somma delle devianze per ogni cluster) e sulla base di questa si può calcolare la quantità richiesta.
Matematicamente, definita $DEV_{PCA-PERSA}$ la devianza persa (in termini percentuali) a causa della PCA, viceversa $DEV_{PCA-MANTENUTA}$ la devianza mantenuta dalla PCA, e DEV_{INTRA} la devianza intra-cluster, la devianza totale persa percentuale vale:

$$DEV_{PCA-LOST} + DEV_{INTRA} \times DEV_{PCA-MANTENUTA}$$

Essa può essere calcolata in MATLAB passando ad uno script i cluster, le componenti principali e il dataset iniziale.
Per dare valore ai fattori sopra citati il clustering viene effettuato scegliendo un numero di cluster che varia da 6 a 16 (per ogni gruppo di componenti principali) su cui poi viene calcolata la devianza persa.

2.3.1 4 Componenti Principali

Utilizzando le prime quattro PC si ha:

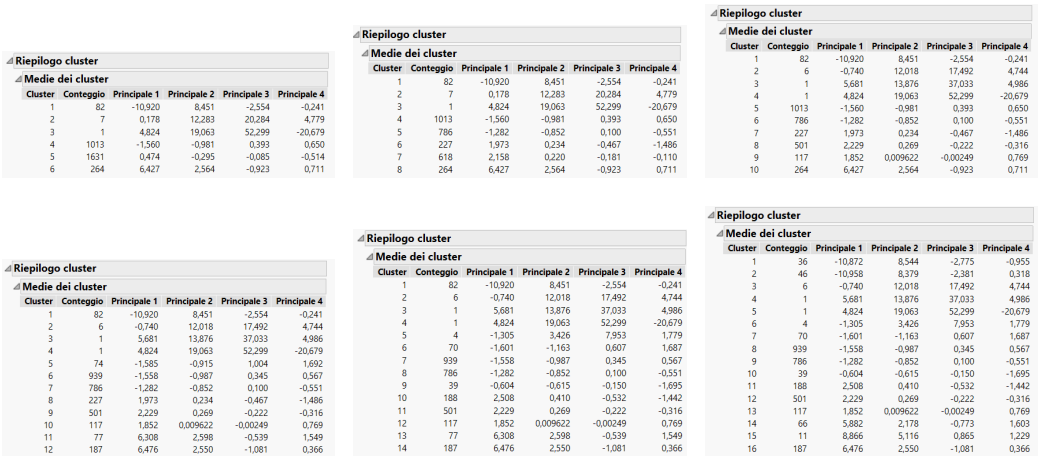


Figura 2.8: *Numero di cluster e dimensione per diversi valori*

La devianza persa durante la clusterizzazione varia in relazione al numero di cluster scelti. Si possono racchiudere le informazioni in un'unica tabella:

6 Cluster	8 Cluster	10 Cluster	12 Cluster	14 Cluster	16 Cluster
26%	17%	16%	15%	14.5%	14%

2.3.2 5 Componenti Principali

Utilizzando le prime quattro PC si ha:

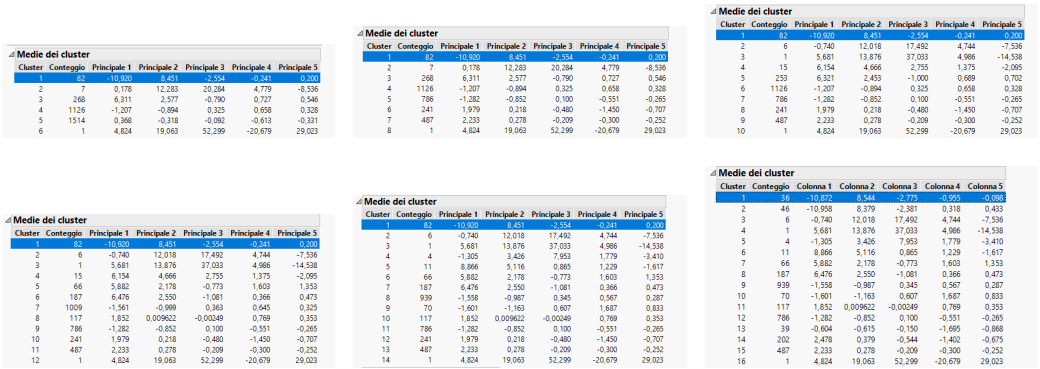


Figura 2.9: *Numero di cluster e dimensione per diversi valori*

La devianza persa durante la clusterizzazione varia in relazione al numero di cluster scelti. Si possono racchiudere le informazioni in un'unica tabella:

6 Cluster	8 Cluster	10 Cluster	12 Cluster	14 Cluster	16 Cluster
25.5%	16%	15%	12%	11%	10%

2.3.3 6 Componenti Principali

Utilizzando le prime quattro PC si ha:

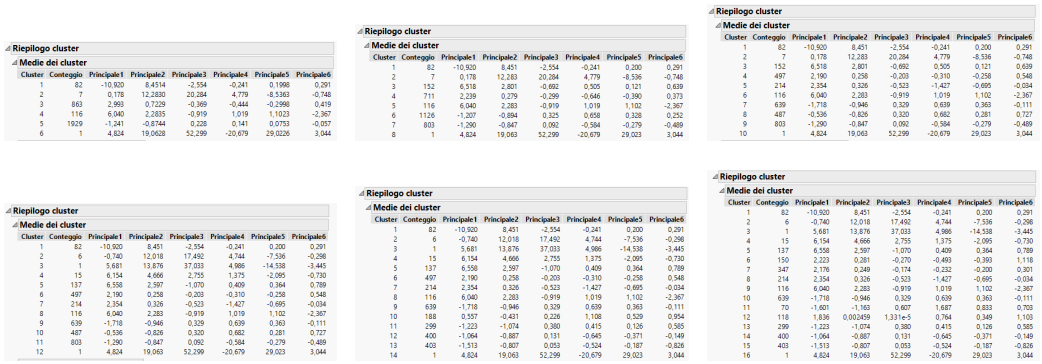


Figura 2.10: *Numero di cluster e dimensione per diversi valori*

6 Cluster	8 Cluster	10 Cluster	12 Cluster	14 Cluster	16 Cluster
22%	15%	13%	12%	11%	9%

2.3.4 Interpretazione

All'inizio dell'analisi sono state effettuate delle ipotesi che hanno trovato riscontro nella procedura di clustering:

1. La fase iniziale del sistema (descritto dalle prime righe) trova riscontro con il cluster numero uno qualunque siano le componenti principali e qualunque sia il numero di cluster scelto. Questo quindi prova l'ipotesi definita inizialmente

2. L'ultimo cluster contiene sempre un elemento singolo. Questo accade a causa del fatto che è stato identificato un picco nella fase iniziale delle misure. Esso inoltre è stato definito grazie all'outlier nel parametro Slab che non è stato eliminato, di conseguenza il picco viene racchiuso in un unico cluster in tutte le situazioni.

In conclusione si può costruire una tabella che racchiude le informazioni riguardanti le PCA e la clusterizzazione in termini di percentuale di devianza persa.

	6 Cluster	8 Cluster	10 Cluster	12 Cluster	14 Cluster	16 Cluster
4 PC	26%	17%	16%	15%	14.5%	14%
5 PC	25.5%	16%	15%	12%	11%	10%
6 PC	22%	15%	13%	12%	11%	9%

2.4 Workload Sintetico

Come anticipato nel paragrafo precedente, sono state scelte 5 PC e per non perdere troppa varianza ma al tempo stesso non sfociare in un numero di cluster molto elevato, si è scelto di considerare 10 cluster. In tal caso la perdita di devianza con PCA e Cluster è di circa del 15%.
In conclusione dopo aver calcolato i centroidi con uno script MATLAB il workload sintetico risulta essere:

VmPeak'	VmSize'	VmHWM'	VmRSS'	VmPTE'	Threads'	MemFree'	Buffers'	Cached'	Inactive'	Dirty'	Mapped'	Slab'	PageTables'	Committed_AS'	NumOfAlloc...	proc-fd'	avgThrough...	avgElaps...	Cluster
152272	150216	25228	25208	160	58	5604588	33440	334672	141440	304136	148	0	77472	23772	27852	7192	294280	1530	1
153484	151436	26612	26468	160	20	5044360	83684	732680	365572	576940	115360	0	126116	28324	91172	7984	347336	2040	2
170344	170340	31144	31140	200	52	4655664	89480	1090572	541220	773332	324400	1760	134500	29956	106788	8028	365264	2040	10
170344	170340	31144	31140	200	43	4471580	94884	1259568	597900	891332	459632	0	134504	29956	116216	8024	365692	1530	3
172392	170340	32696	31516	200	35	4608100	158180	1117364	424808	934596	176	0	83860	23740	110212	7208	318956	1020	7
172392	170340	32696	31340	200	35	4605508	160628	1117388	427232	934468	152	0	83684	23740	110320	7208	318508	1530	6
192256	184832	43804	37584	252	87	4557264	198720	1118136	488652	918124	56	0	89880	23848	113836	7260	329604	510	8
192384	185460	43804	37396	256	97	4555172	201052	1118212	494300	914696	36	0	89736	23848	113556	7264	330928	1020	9
215352	210232	61784	59084	316	116	4527992	204160	1118424	526896	908536	148	0	112848	23996	113028	7772	356604	510	4
215352	206136	61784	53204	312	9	4539096	204400	1118460	520240	908140	148	0	105548	23848	110300	7320	348476	510	5

Figura 2.11: *Workload sintetico*

Capitolo 3

Web Server - Capacity Test

L'obiettivo del Capacity Test è quello di valutare le performance di un qualsiasi sistema quando è sottoposto a carichi di lavoro di diversa intensità, in modo da caratterizzare le sue prestazioni al limite (sotto condizioni di lavoro severe).
Per realizzare queste valutazioni sono necessari gli **high-level parameters**, ovvero tutti quei parametri reperibili ed osservabili lato client. Essi possono riferirsi alla richiesta (quando è stata fatta, chi l'ha fatta ecc..) o alla risposta (tempi di risposta, errori).
Essendo il sistema in questione un server, si è scelto di descrivere le sue performance attraverso:

1. **Response Time**, intervallo di tempo che intercorre tra l'istante in cui il client inoltra la richiesta e quello in cui riceve la risposta.
2. **Throughput**, richieste servite correttamente per unità di tempo.

L'andamento atteso da parte di queste due metriche è il seguente:

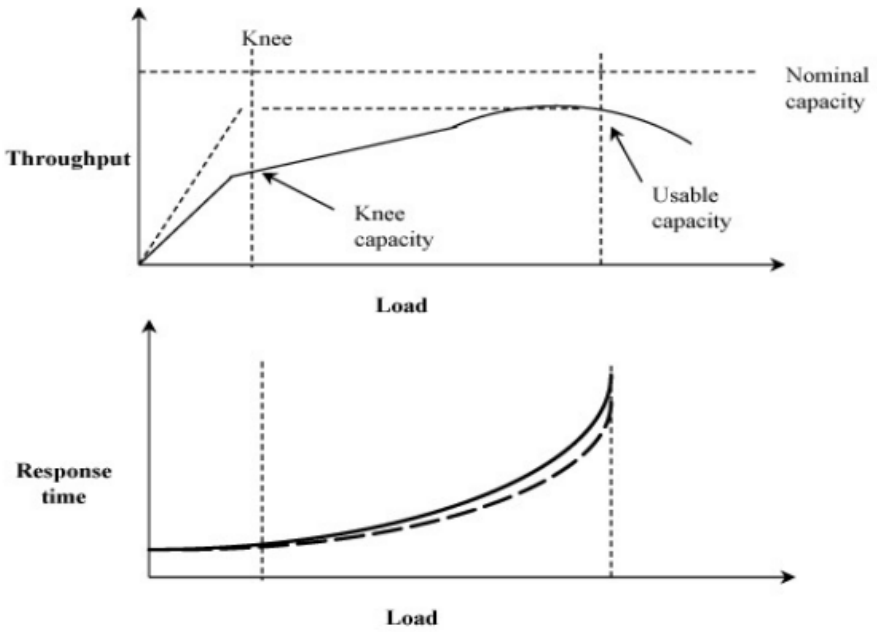


Figura 3.1: *Grafici Throughput e Response time*

Di nostro interesse sono i valori di:

- *Knee Capacity*, punto prima del quale il throughput cresce linearmente all'aumentare del carico, ma il tempo di risposta non varia significativamente ed oltre il quale il guadagno in throughput è basso mentre il tempo di risposta aumenta con il carico.
- *Usable Capacity*, massimo throughput raggiungibile portando il sistema al limite, senza eccedere un dato tempo di risposta.

Per ottenere agevolmente la Knee Capacity, viene introdotto un terzo parametro, la *Potenza*.

$$Power = \frac{Throughput}{ResponseTime} \tag{3.1}$$

Tale punto coincide con il punto di massimo della potenza e rappresenta l'ottimo in corrispondenza del quale conviene operare per ottenere le prestazioni migliori.

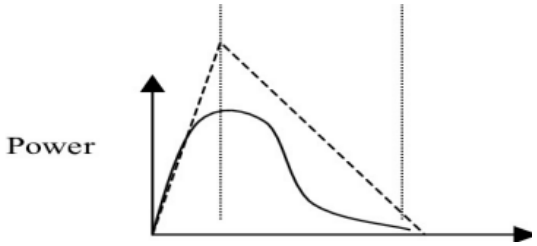


Figura 3.2: *Grafico Potenza*

3.1 Experimental Setup

Il sistema oggetto di studio è un *Web Server Apache* installato sulla macchina virtuale guest, che funge da server.
Tramite la modalità *Host-only Network Adapter*, configurabile nelle impostazioni della macchina virtuale, è stato possibile far comunicare la macchina guest con quella host, che lo ospita. Su quest'ultima è stata installata l'applicazione Java *JMeter*, che ha permesso l'analisi delle prestazioni complessive del Server, sottoponendolo a diversi tipi di carico.

In questa analisi è stato scelto di valutare le prestazioni in media del server, considerando solo richieste (HTTP di tipo GET) casuali. Esse sono differenziate dalla dimensione della risorsa che chiedono al server.

3.1.1 Server Setup

Il server è stato installato su una macchina virtuale *Ubuntu 2021* in esecuzione su una macchina host di uso comune. Essa è stata dotata di circa 4GB di RAM e di 2 processori (intel I5-5200u con frequenza massima di 2.70 GHz).
Per creare un scenario reale, sul Server sono state caricate 5 pagine in formato testuale, di diversa dimensione:

- **Small**: 50 KB

- **Small-Medium**: 100 KB
- **Medium**: 300 KB
- **Medium-Large**: 500 KB
- **Large**: 1 MB

Questi sono i file oggetto delle richieste realizzate da ipotetici client.

3.1.2 Clients Setup - JMeter

Innanzitutto è stato settato, nel *ThreadGroup*, il numero di thread che JMeter usa per realizzare i test. Questa quantità rappresenta il numero di utenti "virtuali" che visitano il nostro server. Nel nostro esperimento sono stati previsti **50 threads**, un valore in linea con i suoi scopi (dato che si tratta di un banale webserver virtualizzato su una macchina host di uso comune). In più, prevedendo dei test di durata pari a *5 min*, sono stati impostati:

- il **Ramp-up period** - numero di secondi entro il quale deve essere attivato l'ultimo thread - a *300 s*. Ciò ci ha permesso di dilazionare l'attivazione degli utenti nei 5 minuti.
- il **Thread lifetime** - durata massima di ogni thread - a *300 s*.
- il **Loop count** - numero di volte in cui un singolo thread effettua una richiesta. Esso corrisponde al numero di richieste nell'intervallo di tempo di simulazione (in questo caso 300s) diviso il numero di threads.

Al ThreadGroup sono stati aggiunti 5 *HTTP Request Sampler*, uno per tipologia di richiesta da realizzare e nei quali sono stati specificati i path delle rispettive risorse sul server. Ad essi è stato integrato un *Random Controller*, grazie al quale, quando un thread viene attivato, effettua solo una tra le cinque tipologie di richieste, selezionata in maniera randomica. Ancora una volta, aggiungendo variabilità alle nostre richieste, è stato possibile simulare una situazione realistica e, soprattutto, non predicibile.
Attraverso il *Constant Throughput Timer* è stato possibile impostare il carico da sottoporre al sistema, in termini di numero di richieste al minuto. Infine, il listener *Simple Data Writer*, ci ha permesso di collezionare in un file, quei parametri di alto livello che sono d'interesse ai fini dell'esperimento.
L'idea è quella di simulare quindi 50 utenti, di cui ognuno effettua un numero di richieste in relazione al carico, per 5 min.
Ciò equivale a:

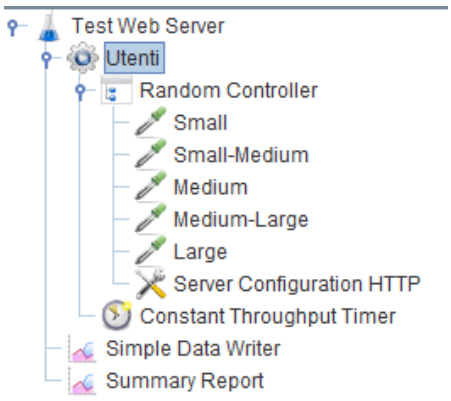


Figura 3.3: *Configurazione delle richieste e del carico in JMeter*

I risultati, grazie al Simple Data Wirter, vengono salvati in formato .csv, i cui paramentri vengono raggruppati in forma tabellare.

TimeStamp	Elapsed	Latency	...
⋮	⋮	⋮	⋮

3.2 Esecuzione Capacity Test

Inizialmente sono stati effettuati dei test il cui scopo era quello di far operare il Web Server "al limite". Ci si è resi conto che il massimo valore di carico entro il quale il sistema risponde adeguatamente (in quelle condizioni), è di circa *6000 richieste al minuto*.
A partire da questo limite e ragionando sull'andamento di throughput e response time, si sono scelti i seguenti valori di carico da sottoporre al sistema:

$$workloads = 100, 500, 800, 1000, 2000, 3000, 4000, 5000, 6000, 7000, 8000, 9000$$

Gli ultimi tre carichi (7000, 8000 e 9000 richieste al minuto) hanno permesso di evidenziare nei grafici il degradamento delle prestazioni del sistema.
Per ogni valore di carico è stato calcolato il **Throughput**:

$$Throughput = \frac{NumeroRichieste}{Timestamp(N) - Timestamp(1)} \quad \left[\frac{N}{s}\right] \tag{3.2}$$

Il *timestamp* fa parte degli high-level parameters collezionati dal Simple Data Writer, e corrisponde all'istante di tempo (in millisecondi poi convertito in secondi) in cui il client ha inoltrato una data richiesta.
Come **Response Time** è stato scelto il parametro *Elapsed*, coincidente con il tempo che intercorre tra la sottomissione della richiesta da parte del client e la risposta del server. Dato che contiene anche il tempo di elaborazione della richiesta da parte del server, esso cresce all'aumentare della dimensione dei file richiesti, oltre che all'aumentare del carico.

Ogni misurazione (per ogni carico) è stata ripetuta **3 volte** in modo da tenere traccia dell'errore, e notando che i dati ottenuti non differivano di molto tra loro, come indice di posizione è stata scelta la loro media.

3.2.1 Risultati

I file .csv sono stati caricati in uno script Matlab tramite cui sono stati automatizzati i procedimenti descritti sopra, i parametri sono stati plottati in funzione del carico considerato. differenziale).

```
%% Data
workloads = [100 500 800 1000 2000 3000 4000 5000 6000 7000 8000 9000];
througputs = zeros(1, length(workloads));
resp_times = zeros(1, length(workloads));
k = 1; %Indice di riferimento per i due vettori

%% Elaborazione
for i = workloads
        mean_resp_t = zeros(1,3);
        thr = zeros(1,3);
        for j = 1:3
                path = strcat(num2str(i,
↪  '%d'),'\dati',num2str(j,'%d'),'.csv');

                %Data from Jmeter output file (csv format)
                simple_data = readmatrix(path);

                %Calculating the number of requests
                [N, M] = size(simple_data);
                num_req = N; %Number of requests

                %Throughput = number_of_requests_completed /
↪  time_window_of_the_experiment
                t_wind_mills = simple_data(num_req,1) - simple_data(1,1);
↪  %Time window (milliseconds)
                t_wind_sec = t_wind_mills/1000; %Time window (seconds)
                thr(j) = num_req/t_wind_sec; %Throughput

                %Average response time
                elap_times = simple_data(:,2);
                mean_resp_t(j) = mean(elap_times);
        end

        check deviazione_std
        COV_thr(k) = std(thr)/mean(thr);
        COV_resp(k) = std(mean_resp_t)/mean(mean_resp_t);

        if COV_thr(k) > 0.5
                througputs(k) = median(thr);
        end

        if COV_resp(k) > 0.5
                resp_times(k) = median(mean_resp_t);
        end

end

power = througputs./(resp_times/1000);
power_max = max(power);
KNEE_CAPACITY = througputs(find(power == power_max));
```

Effettuando un grafico dei risultati:

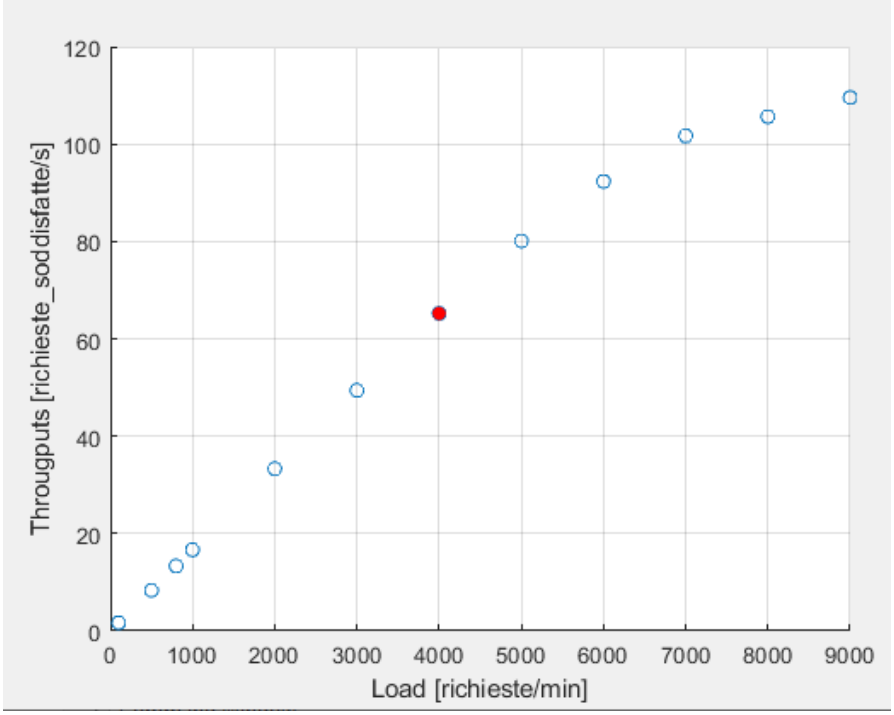


Figura 3.4: *Grafico Throughput*

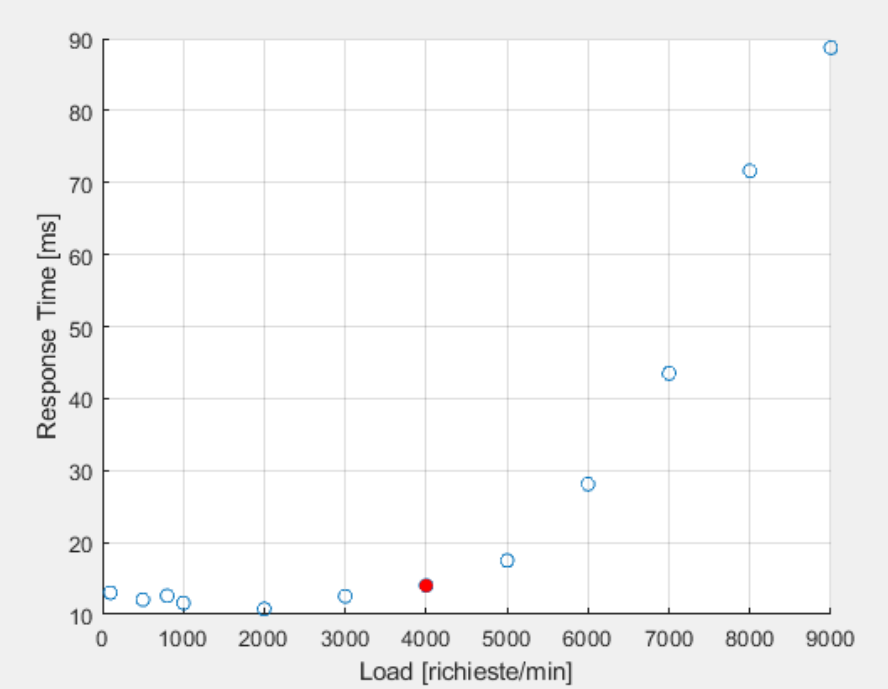


Figura 3.5: *Grafico Response Time*

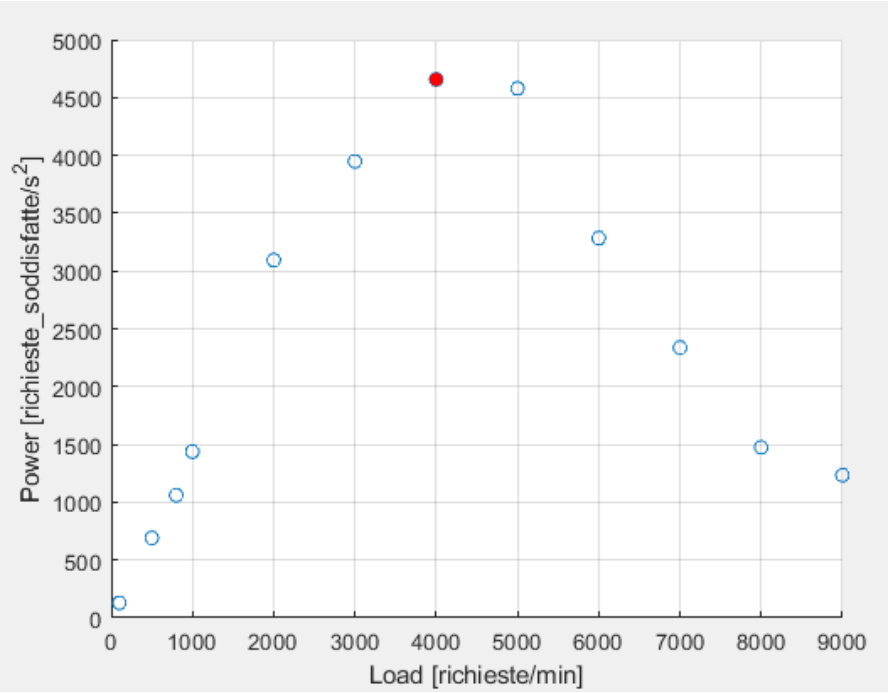


Figura 3.6: *Grafico Potenza*

Come si evince dal grafico della Potenza, il suo punto di massimo è associato ad un carico di 4000 richieste/min.
La *Knee Capacity* (in rosso nel grafico dei Throughputs) è il valore throughput associato a questo carico. Ciò significa che se il nostro server opera in condizioni ottimali riesce a soddisfare circa 65,2112 richieste al secondo, ovvero 3836 richieste al minuto mediamente.

Per quanto riguarda il calcolo della *Usable Capacity*, possiamo relazionarla al tempo di risposta del server quando è sottoposto ad un carico di 6000 richieste/min (il carico limite). Questo response time è pari a 28,11 ms e coincide con quel tempo oltre il quale il sistema inizia a non rispondere più adeguatamente. Pertanto, la Usable Capacity può essere espressa come il valore di throughput associato al carico limite, ovvero 92,3314 richieste al secondo (5431 richieste al minuto circa).
Ovviamente il valore di carico a cui conviene far lavorare il nostro Web Server non è quello massimo che riesce a soddisfare (Usable Capacity). Difatti non sarebbe efficiente per due motivi:

1. I tempi di risposta associato a tale carico sono elevati.
2. Essendo il carico limite, bastano poche richieste in più per ricadere nella zona in cui i tempi di risposta diventano estremamente elevati, rendendo inutilizzabile il server stesso.

Capitolo 4

Web Server - Workload Characterization

L'obiettivo dell'homework è quello di realizzare un workload sintetico semplice e ripetibile, sulla base di un workload reale, attraverso le tecniche descritte nei capitoli precedenti. Successivamente esso deve essere applicato al sistema e, infine, si deve dimostrare che statisticamente si ottiene lo stesso risultato di un workload reale. L'esperimento può essere descritto in tre fasi:

1. *Simulazione di un workload reale.* In questa fase si simulano delle richieste random con carico prefissato al sistema. Si collezionano quindi i dati del client (lista delle richieste) di "alto livello" e i dati del server (memoria, utilizzo della CPU, ecc.) di "basso livello". Alla fine di questa fase bisogna analizzare i dati di alto livello per costruire un workload sintetico.

2. *Applicazione di un workload sintetico.* Dopo aver ricavato il workload sintetico al termine della fase precedente, esso deve essere applicato al sistema. Nuovamente quindi devono essere collezionati i dati di alto livello e basso livello.

3. *Validazione dei dati.* Prevede un'analisi approfondita dei dati di basso livello del workload reale e sintetico. Essi devono essere oppurtunamente caratterizzati per essere infine confrontati statisticamente. Per farlo si utilizzano dei test statistici meglio descritti successivamente.

Le tre fasi possono essere rappresentate graficamente come nella successiva figura.

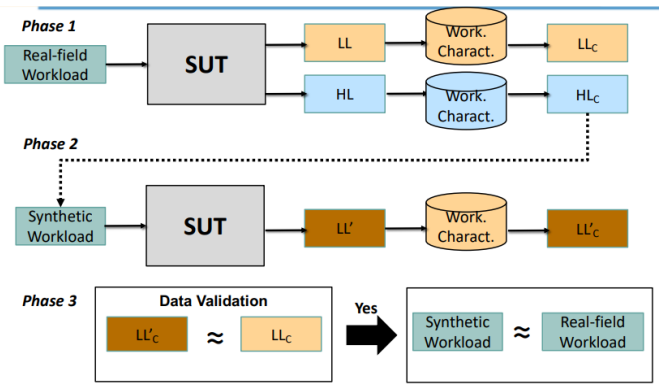


Figura 4.1: *Overview WL Characterization*

Il server deve essere configurato allo stesso in tutte le fasi. In particolare si è utilizzato lo stesso Web Server descritto nel *Capitolo 2* ma con una diminuzione di prestazioni solo a scopo didattico.

4.1 Real-field Workload

Il primo step prevede di applicare o simulare un workload reale. I dati di alto e basso livello devono essere presi contemporaneamente nel client e nel server.

4.1.1 Server

Il server contiene 10 file di diversa dimensione. I file sono stati scelti tutti dello stesso tipo (file di testo) solo per dimensionarli a piacimento. Nulla vieta però di utilizzare file di tipologie diverse (immagini, documenti, audio, etc.).
Essi sono stati dimensionati differenziando i file tra loro di 50 KB e partendo da un minimo di 50 KB. Quindi:

- 50k.txt - File di 50 KB
- 100k.txt - File di 100 KB
- 150k.txt - File di 150 KB
- 200k.txt - File di 200 KB
- 250k.txt - File di 250 KB
- 300k.txt - File di 300 KB
- 350k.txt - File di 350 KB
- 400k.txt - File di 400 KB
- 450k.txt - File di 450 KB
- 500k.txt - File di 500 KB

Parametri di basso livello

Il web server è una macchina virtuale linux. Esistono dunque molti tool in grado di collezionare i parametri caratteristici del sistema. In questo caso è stato utilizzato il tool **vmstat** (**Virtual Memory STATistics reporter**) il quale fornisce informazioni circa le performance del sistema su cui viene eseguito. In particolare esse riguardano:

- **Processi**: numero processi in esecuzione o in attesa di essere eseguiti.
- **Memoria**: memoria libera, swap, buffer etc. Ad esempio
    1. *free*, quantità di memoria libera.
- **Input/Output**: blocchi ricevuti o inviati da/verso un dispositivo a blocchi.
- **Sistema**: parametri di sistema come ad esempio:
    1. *in*, numero di interruzioni al secondo.

2. *cs*, numero di cambi di contesto al secondo.

- **CPU**: tipologia di istruzioni che esegue la CPU etc. Ad esempio:

  1. *us*, tempo trascorso dalla CPU nell'eseguire codice non-kernel
  2. *sy*, tempo trascorso dalla CPU nell'eseguire codice kernel
  3. *id*, tempo trascorso dalla CPU nello stato di idle

Tali informazioni sono tutte utili ai fini dell'esperimento, ma quelle che evidenziano maggiormente gli effetti della nostra analisi sono quelle relative alla CPU e all'I/O.
Vmstat offre inoltre la funzionalità di eseguire un campionamento dei parametri ad una frequenza e durata prefissata, tramite l'apposito comando eseguibile da terminale:

```
vmstat -n 1 400
```

Il primo parametro "1" indica il periodo di campionamento in secondi, mentre il secondo parametro "400" indica la durata totale di esecuzione in secondi. L'output poi può essere semplicemente salvato in un file di testo o csv.
Tale comando è stato avviato pochi secondi prima dell'avvio del test su JMeter, ed ha continuato a campionare per qualche secondo anche dopo la sua fine, dunque nell'analisi dei dati sono attesi parametri il cui andamento evidenzia le varie fasi.

4.1.2 Client - JMeter

La simulazione degli utenti che fanno accesso al server è stata possibile tramite il tool JMeter, già descritto nei capitoli precedenti.
In particolare sono stati realizzati tre Thread Group ognuno composto da 10 Thread (l'equivalente di 10 utenti) con una durata di simulazione pari a 2 min ciascuno. Ogni gruppo contiene 10 richieste riferite alle 10 risorse disponibili nel web server. Esse vengono poi eseguite in modo casuale tramite un apposito controller.

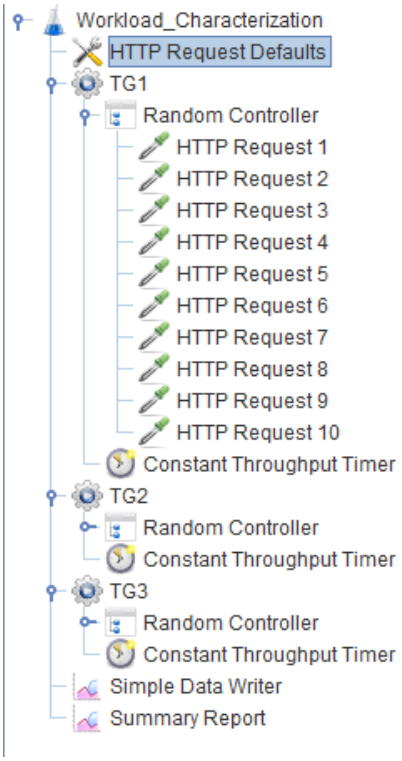


Figura 4.2: *Configurazione di JMeter per la simulazione di un workload reale*

Ogni gruppo inoltre ha un suo specifico carico. In questo caso si ha:

- **TG1** : 400 richieste al minuto
- **TG2** : 550 richieste al minuto
- **TG3** : 700 richieste al minuto

Essi vengono poi eseguiti in sequenza e non in parallelo. Questo perché si possono creare possibili conflitti sulle risorse, dati dal fatto che ogni gruppo richiede le stesse risorse degli altri.

Parametri di alto livello

I parametri di alto livello possono essere collezionati direttamente tramite il tool JMeter e salvati in formato *.cvs*. Non sono necessari dunque programmi esterni. I parametri utili ai fini dell'analisi sono:

- **Timestamp**, l'istante di tempo in cui viene effettuata la corrispettiva richiesta (in millisecondi)
- **elapse**, inteso come Response Time
- **label**, contiene l'informazione categorica della richiesta effettuata.
- **bytes**, numero di byte ricevuti tramite la relativa richiesta.
- **sentBytes**, numero di byte inviati per effettuare la richiesta.

- **latency**
- **connect**, tempo di connessione misurato per effettuare l'handshake TCP (in millisecondi).

4.1.3 Workload Characterization

Le misure vengono effettuate correttamente avviando prima *vmstat* nel server e in seguito JMeter sul client in modo che i dati vengono salvati contemporaneamente nel lato client (alto livello) e nel lato server (basso livello). Al termine della simulazione quindi ci si ritrovano due file di dati.

Parametri di alto livello

I parametri di alto livello vanno incontro alla procedura di filtraggio, PCA e Clustering per ridurne la dimensionalità.
Essi appaiono nel seguente modo:

	timeStamp	elapsed	label	bytes	sentBytes	Latency	Connect
1	1,638814e+12	7	HTTP Request 1	51512	123	6	3
2	1,638814e+12	27	HTTP Request 10	512313	124	2	0
3	1,638814e+12	8	HTTP Request 4	205113	124	2	0
4	1,638814e+12	6	HTTP Request 3	153913	124	2	0
5	1,638814e+12	11	HTTP Request 5	256313	124	2	0
6	1,638814e+12	4	HTTP Request 2	102713	124	2	0
7	1,638814e+12	5	HTTP Request 3	153913	124	2	0
8	1,638814e+12	5	HTTP Request 4	205113	124	1	0

Figura 4.3: *Parametri di alto livello utili ai fini dell'analisi*

La fase di filtraggio non prevede nessuna azione di modifica del dataset.
Sul dataset originale quindi deve essere effettuata la PCA per cercare di ridurne la dimensionalità senza perdere troppa varianza. Bisogna soprattutto considerare che per questi parametri la fase di Clustering è molto importante poiché racchiude le informazioni principali per costruire il workload sintetico.

Tramite la PCA sono state scelte tutte le componenti principali, mantenendo una varianza del 100%.

Autovalori

Numero	Autovalore	Percentuale	20 40 60 80	Percentuale cumulativa
1	1,7076	28,459		28,459
2	1,4831	24,718		53,177
3	0,9992	16,653		69,830
4	0,9836	16,393		86,224
5	0,4669	7,782		94,006
6	0,3597	5,994		100,000

Figura 4.4: *Analisi della varianza tramite autovalori*

Sulla base di queste può essere effettuato il clustering gerarchico.

Il numero di cluster rappresenta il numero di richieste del workload sintetico, poiché in ogni cluster viene scelto un elemento rappresentativo di esso stesso. A tal proposito quindi il numero di cluster non deve essere maggiore del numero di *HTTP Request* totali utilizzate durante la simulazione (in questo caso 30) e non deve essere un numero molto elevato. Al tempo stesso però non si deve perdere molta varianza a causa della clusterizzazione.
La via più semplice è quella di effettuare delle prove scegliendo un numero di cluster minore della metà (in questo caso minore di 15) e valutare per ogni numero quanta varianza si perde.
Partendo da 6 componenti principali si può scegliere un numero di cluster variabile e calcolare la devianza persa per ogni valore.

6 Cluster	**8 Cluster**	**10 Cluster**	**12 Cluster**
35%	27%	21.5%	17%

La scelta ricade su 10 cluster in modo da avere un workload sintetico abbastanza ristretto e una perdita di varianza non troppo elevata.
Per scegliere gli elementi rappresentativi di un cluster si può ricorrere a vari metodi:

- Il punto più centrale possibile
- Il punto in cui un valore categorico si ripete più volte. Applicabile però solo se il dataset ha un parametro categorico in un insieme limitato di valori.
- Casualmente
- Il punto che si avvicina il più possibile alla media del cluster
- Etc.

Parametri di basso livello

	r	b	swpd	free	buff	cache	si	so	bi	bo	in	cs	us	sy	id	wa	st
1	0	0	0	1394332	44704	946952	0	0	6025	234	660	2998	51	20	23	5	0
2	0	0	0	1394324	44704	946952	0	0	0	0	494	236	8	1	91	0	0
3	0	0	0	1394324	44704	946952	0	0	0	0	468	221	12	1	87	0	0
4	1	0	0	1394324	44704	946952	0	0	0	0	494	389	8	0	92	0	0
5	0	0	0	1394324	44704	946952	0	0	0	0	478	202	9	1	90	0	0
6	0	0	0	1394324	44704	946952	0	0	0	0	472	237	12	1	87	0	0
7	0	0	0	1394324	44704	946952	0	0	0	0	503	370	10	1	89	0	0
8	0	0	0	1394308	44704	946952	0	0	0	0	522	488	20	2	78	0	0
9	0	0	0	1394308	44704	946956	0	0	4	0	457	249	13	1	86	0	0
10	0	0	0	1394308	44704	946956	0	0	0	0	462	249	15	1	84	0	0
11	0	0	0	1394016	44712	947000	0	0	64	0	482	251	11	1	88	0	0
12	0	0	0	1393760	44712	948188	0	0	1208	0	632	310	20	6	73	1	0
13	0	0	0	1393032	44712	948628	0	0	400	0	612	263	13	5	82	0	0
14	0	0	0	1392044	44712	949380	0	0	752	4	741	308	22	10	68	0	0
15	0	0	0	1391664	44712	949776	0	0	352	8	711	280	13	8	79	0	0
16	0	0	0	1391696	44712	949780	0	0	0	0	674	292	18	7	75	0	0
17	0	0	0	1391696	44720	949780	0	0	0	52	582	318	20	5	75	0	0
18	0	0	0	1391696	44720	949780	0	0	0	0	699	265	11	6	83	0	0
19	0	0	0	1391696	44720	949780	0	0	0	0	635	262	17	4	79	0	0
20	0	0	0	1391696	44720	949780	0	0	0	0	660	295	19	7	74	0	0
21	0	0	0	1391696	44720	949784	0	0	0	0	665	286	13	7	80	0	0
22	0	0	0	1391696	44728	949784	0	0	0	12	655	306	19	6	75	0	0
23	0	0	0	1391632	44728	949784	0	0	0	0	684	312	13	7	79	0	0
24	0	0	0	1391316	44872	949792	0	0	144	0	716	289	11	7	81	1	0
25	0	0	0	1391156	44872	949792	0	0	0	0	671	296	18	8	74	0	0
26	0	0	0	1391092	44872	949796	0	0	0	0	697	303	12	7	80	0	0
27	0	0	0	1390996	44880	949788	0	0	0	12	633	284	17	7	76	0	0
28	0	0	0	1390836	44880	949796	0	0	0	0	658	293	16	6	78	0	0
29	0	0	0	1390772	44880	949796	0	0	0	0	740	280	14	7	79	0	0
30	0	0	0	1390676	44880	949800	0	0	0	0	697	272	17	8	75	0	0
31	2	0	0	1390388	44880	949800	0	0	0	0	620	314	15	5	80	0	0

Figura 4.5: *Porzione dataset - Low Level Parameters*

I parametri di basso livello costituiscono un dataset formato da 17 colonne e 400 righe. Su di esso sono state dunque effettuate operazioni di *filtraggio*, *PCA* e *clustering*, procedendo in maniera analoga a quanto già si era fatto nel Capitolo 1.

Analizzando le distribuzioni sono state individuate 4 colonne costanti (e quindi da rimuovere): **swpd**, **si**, **so**, **st**.
Dopodichè sono state eliminate tutte le righe associate ai campioni prelevati da vmstat quando il client non stava sottoponendo richieste al server, in quanto non forniscono informazioni utili agli scopi della caratterizzazione.
Per fare ciò sono stati analizzati gli andamenti in funzione del tempo dei parametri relativi all'utilizzo della CPU precedentemente descritti.

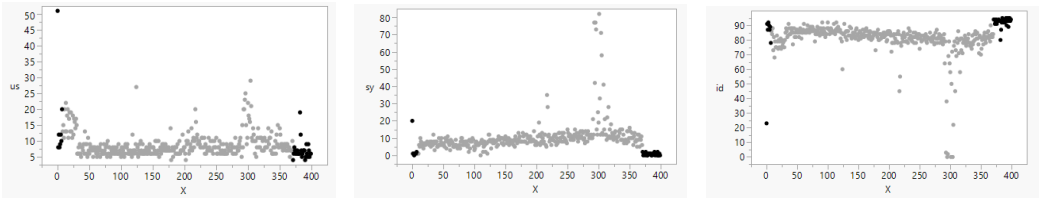


Figura 4.6: *Andamento parametri CPU*

In particolare, osservando i grafici di *sy* e *id* risultano evidenti queste "fasi" nelle quali il processore trascorre meno tempo ad eseguire codice kernel e più tempo in idle rispetto a quando si sta sottoponendo il workload, visto che non è impegnato nel servire le richieste.
Discorso analogo lo si può fare osservando le interruzioni al secondo (**in**), che incrementano drasticamente nel corso della recezione delle richieste, per poi diminuire in queste fasi.

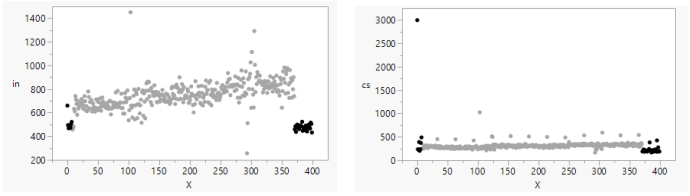


Figura 4.7: *Andamento parametri di Sistema e CPU*

Contestualmente è stata eliminata anche la prima riga (outlier per quasi tutti i parametri), in quando è stata associata alla fase di avvio dell'esecuzione del comando vmstat e quindi non di particolare interesse per gli scopi dell'esperimento.
L'ultima operazione effettuata in questa prima fase di filtraggio è stata la rimozione di un outlier associato al parametro **b**, il quale è indice del numero di processi sospesi ed in attesa di risorse per poter essere riattivati. Essendo outlier anche di **wa** (tempo trascorso dalla CPU in attesa di input/output) è quasi sicuramente indice dello stesso fenomeno, il quale è stato da noi considerato come casuale e quindi trascurabile.

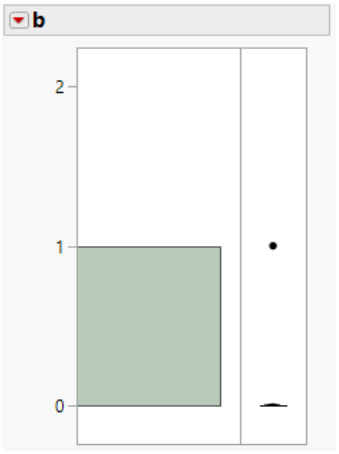


Figura 4.8: *Distribuzione di b*

La riga ad esso associata è la 122, la quale è stata opportunamente rimossa. A seguito di questa cancellazione, la colonna associata al parametro b è diventata costante ed è dunque stata eliminata.
Il dataset ottenuto è formato da 12 colonne e circa 360 righe. Il numero di righe risultante può essere ulteriormente validato se consideriamo che la durata del test è proprio di 360 secondi.

Su questi dati è stata effettuata la PCA, a seguito della quale sono state prese in considerazione *7 componenti principali*, le quali spiegano il 92,768 % della devianza totale.

Numero	Autovalore	Percentuale	20 40 60 80	Percentuale cumulativa
1	3,8516	32,096		32,096
2	2,6266	21,888		53,985
3	1,4119	11,766		65,751
4	1,1762	9,802		75,552
5	0,8663	7,219		82,772
6	0,6281	5,235		88,006
7	0,5713	4,761		92,768
8	0,4057	3,381		96,149
9	0,2423	2,019		98,168
10	0,1464	1,220		99,388
11	0,0726	0,605		99,992
12	0,0009	0,008		100,000

Figura 4.9: *PCA Low-Level Filtered*

Per non perdere una porzione significativa di devianza, dopo vari test, si è deciso di selezionare **20 Cluster** conservandone il 78,36%.
Infine è stata implementata in Matlab la seguente funzione per scegliere l'elemento rappresentativo di ogni cluster:

```
function [new_workload] = random_selection(workload)
%workload = colonne PCA + colonna cluster
N_cluster = max(workload(:,end)); %numero cluster
[r,c] = size(workload);

%isolo la colonna dei cluster
cluster_data = workload(:,end);

new_workload = zeros(N_cluster,c);
```

```
%itero per ogni cluster
for i = 1:N_cluster
%prelevo tutti gli indici di riga associati ad uno stesso cluster i
        index = find(cluster_data==i);
        %se ho più di una riga ne scelgo una random
        if length(index) > 1
                new_workload(i,:) = workload(randsample(index,1),:);
        else
                %se ho solo una riga scelgo quella
                new_workload(i,:) = workload(index,:);
        end
end
end
```

La funzione riceve in input le colonne estratte dalla PCA affiancate alla colonna dei cluster, la quale specifica il cluster di appartenenza di ciascuna riga. Come si evince dal codice, i centroidi sono stati selezionati in maniera randomica nel caso in cui nel cluster sia presente più di un elemento.
L'output della funzione è un workload costituito da 7 colonne e 20 righe, rappresentativo di quello originario.

4.2 Synthetic Workload

A partire dalla clusterizzazione di alto livello, identificati gli elementi rappresentativi di ogni cluster, si deve rifare la simulazione ma con il workload sintetico.
Il dataset in esame è composto da un parametro categorico che rappresenta la richiesta effettuata (risorsa e utente) facendo parte di un insieme molto limitato. La scelta dell'elemento rappresentativo può ricadere nel scegliere la *label* che si ripete in più punti nello stesso cluster.
A tal proposito si può calcolare una tabella che per ogni cluster indica il numero di ricorrenze del valore del parametro categorico.

	Cluster 10	N righe	N(HTTP Request 1)	N(HTTP Request 2)	N(HTTP Request 3)	N(HTTP Request 4)	N(HTTP Request 5)	N(HTTP Request 6)	N(HTTP Re
1	1	51	1	3	0	1	0	2	
2	2	587	0	70	70	90	79	54	
3	3	701	0	0	0	0	0	20	
4	4	977	0	0	0	0	0	0	
5	5	598	0	0	0	0	0	0	
6	6	2	0	0	0	0	0	0	
7	7	39	0	0	0	0	0	0	
8	8	9	0	0	0	0	0	0	
9	9	15	0	0	0	0	0	0	
10	10	325	79	0	0	0	0	0	

Figura 4.10: *Tabella che associa ad ogni cluster il numero di punti con una determinata label*

La matrice che compone la tabella è una matrice 10x30 (10 cluster e 30 richieste possibili) e bisogna calcolare il massimo di riga per ogni riga. Inoltre se il massimo di riga è associato ad una *label* già selezionata in precedenza, allora si deve scegliere il secondo massimo nella riga e così via. Si può automatizzare il tutto tramite uno script MATLAB:

```
%% Dati
[data, txt] = xlsread('label-cluster 10');
```

```
data_filter = data(:, 3:end);
txt_filter = txt(:, 3:end)';

%% Ricerca centroidi
[r,c] = size(data_filter);
max_list = zeros(1,r);  % Lista dei massimi
max_index = zeros(1,r); % Lista degli indici dei massimi

%Inizializzo vettore degli indici
for i=1:r
        max_index(i) = -1;
end

for i=1:r
        [temp_v, temp_i] =  max(data_filter(i,:));
        while ismember(temp_i,max_index)
                data_filter(i,temp_i) = -1;
                [temp_v, temp_i] =  max(data_filter(i,:));
        end
        max_list(i) = temp_v;
        max_index(i) = temp_i;
end

%% Stampa risultati
sort(txt_filter(max_index))
```

Il risultato dello script è la lista delle *label* che identificano il workload sintetico.

1. HTTP Request 1 : TG1 con risorsa *200k.txt*
2. HTTP Request 11 : TG2 con risorsa *50k.txt*
3. HTTP Request 17 : TG2 con risorsa *350k.txt*
4. HTTP Request 21 : TG3 con risorsa *50k.txt*
5. HTTP Request 22 : TG3 con risorsa *100k.txt*
6. HTTP Request 23 : TG3 con risorsa *150k.txt*
7. HTTP Request 25 : TG3 con risorsa *250k.txt*
8. HTTP Request 27 : TG3 con risorsa *350k.txt*
9. HTTP Request 28 : TG3 con risorsa *400k.txt*
10. HTTP Request 29 : TG3 con risorsa *450k.txt*

4.2.1 Server

Il server non deve essere assolutamente modificato. Tutte le configurazioni effettuate per il workload reale rimangono invariate

Parametri di basso livello

I parametri di basso livello devono essere collezionati allo stesso modo utilizzato nel workload reale. Essi devono essere confrontati con i parametri di basso livello salvati durante la simulazione del workload reale. Questa operazione verrà analizzata nel paragrafo *Data Validation*.

4.2.2 Client - JMeter

Il client deve, a questo punto, simulare le nuove richieste applicando il workload sintetico ricavato con l'analisi. Anche in questo caso la configurazione di JMeter non deve essere modificata, tranne che per le richieste.

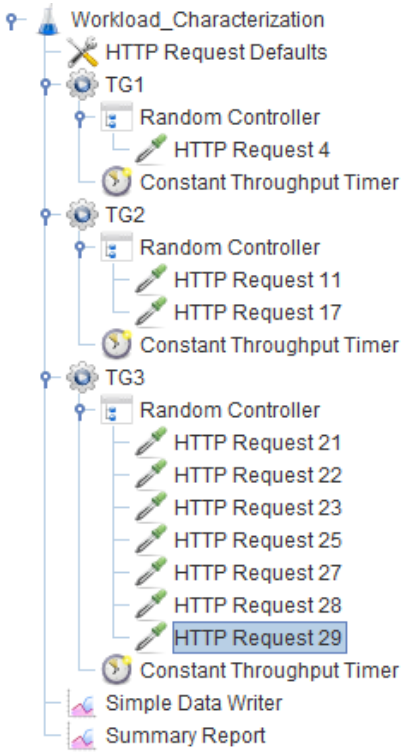


Figura 4.11: *Configurazione di JMeter per il workload sintetico*

Parametri di alto livello

I parametri di alto livello possono anche non essere collezionati poiché non servono per ulteriori analisi. Per completezza però si possono salvare tramite lo stesso JMeter usato per la simulazione.

4.2.3 Workload Characterization

La caratterizzazione può essere applicata solo ai parametri di basso livello. Il risultato viene confrontato con la caratterizzazione degli stessi parametri misurati con il workload reale. Deve necessariamente accadere che il numero di colonne di questi parametri sia lo stesso.

Parametri di basso livello

Anche in questo caso il dataset di basso livello è costituito da 17 colonne e 400 righe. Analogamente a quanto fatto in precedenza per i parametri relativi al Workload reale sono state eseguite le operazioni di filtraggio, PCA e clustering.
Sono state eliminate le colonne costanti, che in questo caso sono: **b**, **swpd**, **si**, **so** e **st**.

Dopodichè, eliminando le righe relative ai campionamenti nelle fasi in cui il sistema non stava servendo alcuna richiesta, il set di dati è stato ulteriormente ridotto. L'analisi degli outlier non ha portato all'eliminazione di alcuna riga, in quanto tutti i punti oltre i quartili dei box-plot sono stati considerati significativi.

Come accaduto anche in precedenza, il dataset risultate è composto da 12 colonne e circa 360 righe (coerentemente con il tempo di sottomissione del workload).

Numero	Autovalore	Percentuale	20 40 60 80	Percentuale cumulativa
1	4,6975	39,146		39,146
2	2,0232	16,860		56,006
3	1,1235	9,362		65,368
4	1,0373	8,644		74,012
5	0,8961	7,468		81,480
6	0,7930	6,608		88,088
7	0,6228	5,190		93,279
8	0,5109	4,258		97,536
9	0,2079	1,732		99,268
10	0,0625	0,521		99,789
11	0,0207	0,172		99,961
12	0,0047	0,039		100,000

Figura 4.12: *PCA syn Low-Level Filtered*

Per la PCA è stato scelto lo stesso numero di componenti selezionate nella caratterizzazione dei parametri di basso livello del workload reale. Questa è stata una scelta obbligata dal fatto che le funzioni Matlab utilizzate per la validazione operano su matrici le quali devono necessariamente avere lo stesso numero di colonne.
In ogni caso selezionando tali componenti si riesce a mantenere un'ottima percentuale di devianza: il 93,279%.

Anche qui la scelta più conveniente in termini di devianza persa è stata quella di suddividere il dataset in 20 Cluster, conservandone il 75,51% della totale.
Infine il workload di basso livello così ottenuto, è stato dato in input alla funzione *random_ selection*, già descritta in precedenza, per ricavare i centroidi di ogni cluster.

4.3 Data Validation

I parametri di basso livello relativi a workload reale e sintetico, ottenuti in seguito alla caratterizzazione sono i seguenti:

real =

-7.6467	3.8108	-0.2363	-1.1546	1.8595	0.9876	-1.1104	1
-2.0806	1.2423	0.0243	-0.7371	0.5829	0.2139	-0.6864	2
-3.2323	2.7898	2.1293	-1.2852	-0.7765	-1.3999	1.1639	3
-1.3219	-0.2341	-0.5549	-0.2241	0.1265	0.2619	-0.2032	4
0.0722	-0.6433	-0.3106	-0.2053	-0.4204	-0.1077	0.0387	5
0.7494	-1.9382	-0.8197	0.2383	-1.1200	-0.4296	0.5033	6
4.1076	2.7490	-0.0809	-0.2197	0.3025	-0.2231	-0.9849	7
-2.8183	2.7099	1.4750	0.9798	2.6722	-1.3798	-0.5066	8
-1.7812	0.0375	-0.2656	1.5176	1.5185	-0.0486	1.1666	9
-0.5033	-1.6519	3.3529	6.3848	6.0619	-4.6155	1.9613	10
2.9730	4.3130	-1.4532	1.0525	-0.1025	1.0709	2.2629	11
1.3715	-0.8105	-0.3164	-0.1945	-0.1326	0.5953	0.7807	12
4.8340	7.2083	-3.4839	2.8833	0.1221	2.3996	4.0902	13
1.0610	-0.7975	1.4556	-1.5270	0.8094	1.1570	-0.0215	14
7.7519	4.5686	3.1382	-2.6228	2.7335	1.8081	-1.3047	15
2.9350	-2.8465	8.4219	-5.4299	6.2165	4.6344	0.4080	16
4.9911	8.8931	-3.8761	3.1680	-0.6799	-1.1646	-0.8345	17
-0.1524	-0.1828	1.1103	2.5281	-1.2307	1.8948	-0.0178	18
-0.4422	0.5430	2.8811	4.3395	-3.1234	2.2766	-2.1920	19
-6.8105	7.8404	9.0113	-2.4265	-5.9991	-2.5519	4.3179	20

Figura 4.13: *Low-Level Parameters Real WL*

synthetic =

1.3613	3.8828	-2.3363	-0.5061	0.5182	4.2447	1.0046	1
-2.0107	-0.1983	0.1911	-0.1098	-0.3192	-0.2824	-0.7827	2
1.2592	1.1510	-0.4001	-0.0740	-0.5915	-0.6392	0.5182	3
-2.4652	-0.8116	0.3711	-0.5470	-0.1421	0.7935	0.0496	4
-1.4773	-1.0322	-0.0326	-0.3627	-0.0170	0.1397	-0.3634	5
-1.1208	0.2234	0.8272	-0.0980	-1.0114	-0.1495	2.4649	6
0.1193	-0.3989	-0.3491	-0.5094	-0.2201	0.1424	1.2521	7
4.9240	2.6595	5.4939	1.4931	-1.6292	-1.3502	2.2706	8
1.6783	-0.3867	2.5966	-0.5528	0.8415	0.7942	-0.3666	9
2.0998	-1.8761	-0.1893	-0.2033	0.2341	0.1760	-0.2447	10
3.8096	0.5758	-0.3417	0.3092	-0.4044	-0.8870	-0.5317	11
2.6983	-0.3619	0.1014	0.1586	-0.4047	-0.5110	0.2097	12
-2.5112	2.0493	-0.3773	1.1422	0.0644	-0.5302	0.0175	13
-2.2360	2.5785	-0.7645	2.3690	0.9516	-0.3645	-0.2480	14
-1.9063	0.7859	-0.5217	2.1736	1.2891	0.1960	0.5051	15
-0.4700	0.6165	-0.2866	1.0818	0.3627	-0.1609	0.4268	16
-2.9928	-0.5533	-0.7877	2.9954	2.5809	1.1836	0.9577	17
-0.3187	5.7494	6.7172	-7.4846	11.6770	-4.0430	0.4878	18
3.8729	0.4539	8.2417	0.7729	-0.8239	4.1174	0.2207	19
7.2334	13.1554	-6.6097	-5.0437	1.1857	9.6592	-1.1957	20

Figura 4.14: *Low-Level Parameters Synthetic WL*

Sono questi che andranno statisticamente confrontati per validare il Workload Sintetico.
La procedura per la validazione è molto semplice:

1. Normalità verificata. Se i due dataset provengono da una distribuzione normale allora si possono eseguire test parametrici per la validazione. In particolare si possono usare vari tipi di test statistici anche in base all'omoschedasticità dei campioni.

   (a) Omoschedasticità verificata. Se i due dataset sono omoschedastici allora si possono utilizzare test sulla base di questa condizione verificata.

   (b) Omoschedasticità non verificata. Se i due dataset non rispettano la proprietà di omoschedasticità allora bisogna utilizzare test che si basano su tale condizione non verificata.

2. Nomarlità non verificata. Se i due dataset non provengono da una distribuzione normale allora si devono usare necessariamente test statistici non parametrici per la validazione.

4.3.1 Normalità

Per verificare se un campione proviene da una popolazione con distribuzione normale, ci si può affidare a test visivi oppure a test statistici analitici.

Test Visivo

Visivamente si può capire se un campione proviene da una popolazione con distribuzione normale effettuando un grafico dei quantili del campione rispetto ai quantili di una distribuzione normale. Ciò lo si realizza sfruttando la funzione Matlab **qqplot**.
Ad esempio prendendo una componente principale del workload reale (dopo la caratterizzazione) si può effettuare un grafico dei quantili:

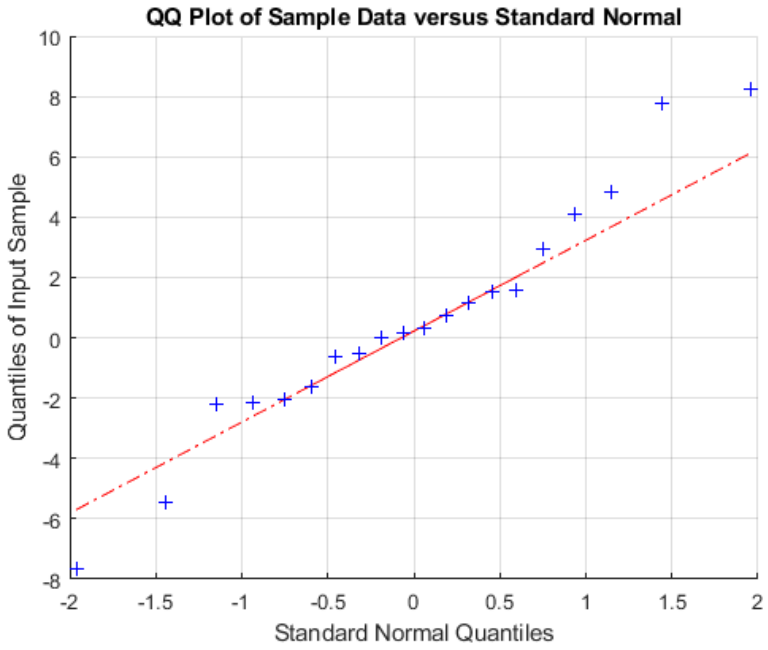


Figura 4.15: *Grafico Quantili-Quantili della prima componente principale del workload reale caratterizzato*

Come si può notare il campione non proviene da una distribuzione normale.

Test analitico

Un test analitico è il test di **Kolmogorov-Smirnov**. Esso si basa sull'ipotesi nulla H_0:

$$H_0 : N(0,1)$$

ovvero che i dati in input provengono da una distribuzione normale standard.
In MATLAB esiste una funzione già definita per eseguire questo tipo di test: **[h,p] = kstest(x)**. Esso fornisce in output il risultato del test (se H_0 viene rigettata o meno) con il relativo *P-Value*. In particolare h è 1 se il test rigetta l'ipotesi nulla H_0 con un livello di significatività del 5%, 0 altrimenti.

Caso di studio - Implementazione

Per la verifica della normalità si sono quindi applicate queste funzioni ai dataset (Fig.4.13 e Fig.4.14).

```
%% Data
real1 = xlsread('real');
synthetic1 = xlsread('syn');

real = random_selection(real1);
synthetic = random_selection(synthetic1);

N = size(real,2); %numero di colonne lo stesso per i due set di dati
real = real(:, 1:N-1); % rimuovo la colonna associata ai cluster
synthetic = synthetic(:, 1:N-1); % lo stesso

%verifica Normal Distribution kstest
[h_ks_real, p_ks_real] = kstest(real);
[h_ks_syn, p_ks_syn] = kstest(synthetic);

%verifica Normal Distribution visual test
figure();
subplot(2,1,1);
qqplot(real);
subplot(2,1,2);
qqplot(synthetic);
```

L'output della kstest è 1 per entrambi i workload, quindi entrambi non provengono da una distribuzione normale. Per completezza si riportano i grafici del quantile-quantile plot associato ad ogni PC dei dataset, i quali confermano il risultato del test di Kolmogorov-Smirnov.

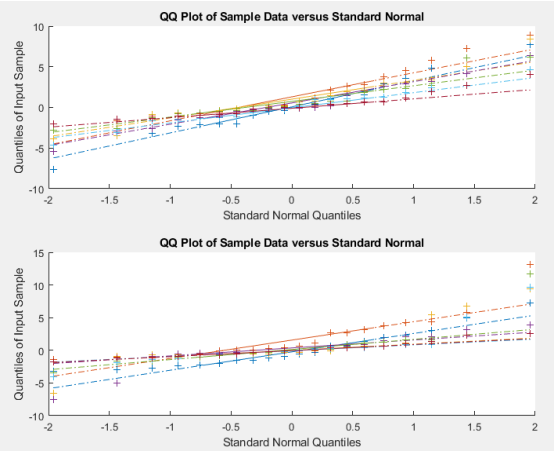


Figura 4.16: *quantile-quantile plot di Fig.4.13 e Fig.4.14*

Ciò suggerisce l'utilizzo di test non parametrici per la validazione.

4.3.2 Omoschedasticità

Anche se nel caso di studio la verifica dell'omoschedasticità non è richiesta, visto che entrambi i set di dati non provengono da una distribuzione normale, per completezza ne riportiamo il procedimento.

Tale proprietà è verificata nel momento in cui diversi campioni hanno stessa varianza (non è detto che essa sia nota), anche se provengono da popolazioni (distribuzioni) differenti. Questa è una caratteristica necessaria da verificare prima di applicare i test parametrici, ad esempio. Difatti, eseguire un test senza aver effettuato questo controllo può avere un impatto significativo sui suoi risultati, fino ad invalidarli.

Test Visivo

Visivamente possiamo capire se le varianze delle due distribuzioni sono simili, tramite il **boxplot**. All'aumentare della lunghezza del box aumenta la varianza dei dati che "rappresenta".

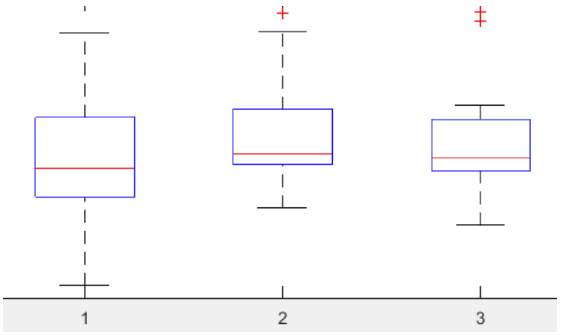


Figura 4.17: *boxplot I,II e III componente del WL reale*

Test analitico

Un test analitico per il check sull'uguaglianza delle varianze dei campioni è:
h = vartest2(x,y).
Esso è applicabile esclusivamente se i due campioni provengono da una distribuzione normale. Si basa sull'ipotesi nulla H_0:

1. H_0 : i vettori x ed y provengono da distribuzioni normali e con uguale varianza.

Il suo output h è 1 se il test rigetta l'ipotesi nulla H_0 con un livello di significatività del 5%, 0 altrimenti.

4.3.3 Validazione

Per completezza è riportato l'intero script, nel quale è stato previsto anche il caso in cui i campioni provengono da distribuzioni normali.

```
%se almeno una delle due distribuzioni non è normale
%applico il test non parametrico
if ((h_ks_real | h_ks_syn) == 1)
[p_wilc,h_wilc] = NoParametric(real,synthetic,N);
else
%distribuzioni normali (risultato di quell if è 0)
%check sulle varianze
[h_var, p_var] = vartest2(synthetic, real);

        %se le due distribuzioni hanno stessa varianza
        if (h_var == 0)
```

```
                %applico il two sample t-test
                [h_ttest, p_ttest] = ttest2(syntetic, real);
                else
                %se le due distribuzioni non hanno stessa varianza
                [h_ttest_novar, p_ttest_novar] = ttest2(syntetic, real,
↪  'Vartype', 'unequal');
                end

        end
```

Come già detto precedentemente, i nostri campioni soddisfano la condizione del primo *if* quindi viene eseguita la funzione *NoParametric*:

```
        function [p_wilc,h_wilc] = NoParametric(real,synthetic,N)
                p_wilc = zeros(1,N-1);
                h_wilc = zeros(1,N-1);
                for i = 1:N-1
                        [p_wilc(i),h_wilc(i)] =
                        ranksum(synthetic(:,i), real(:,i));
                end
        end
```

Essa richiama la funzione MATLAB
[p,h] = ranksum(x,y) la quale esegue il test non parametrico di **Wilcoxon**. Esso si basa sull'ipotesi nulla H_0:

1. H_0 : i dati di x ed y provengono da distribuzioni continue con uguale mediana.

Tale funzione opera sulle singole colonne delle matrici x ed y e quindi bisogna iterare il procedimento per il numero di colonne. Se l'output h è uguale ad 0, si ha un fallimento nel rigettare l'ipotesi nulla con un livello di significatività del 5 %, il contrario se è 1.
L'output di *NoParametric* sarà dunque costituito da due vettori:

1. p_wilc : il vettore degli p-value per ogni componente principale.
2. h_wilc : il risultato della ranksum su ogni componente principale di x ed y.

L' **h_wilc** calcolata sui campioni sotto studio è un vettore di tutti zeri:

```
h_wilc =

     0     0     0     0     0     0     0
```

Figura 4.18: *output test di Wilcoxon*

ciò significa che l'ipotesi nulla non è stata rigettata per ogni componente principale e che quindi Workload Reale e Workload Sintetico hanno determinato effetti statisticamente simili sul server.
Il Workload Sintetico ottenuto è una buona approssimazione di quello Reale.

Capitolo 5

Web Server - Design of Experiment

L'homework si pone l'obiettivo di valutare le performance del web server utilizzato per i precedenti lavori. In particolare lo studio è centrato sul *response time*. Nel dettaglio bisogna studiare l'incidenza di due fattori sul tempo di risposta ricavando un'analisi ANOVA. I fattori di interesse sono:

- **Page Type**, dimensione della risorsa da richiedere al server.
- **Intensità**, inteso come carico da applicare al sistema.

Le performance possono essere valutate utilizzato le tecniche DOE (Design Of Experiment).
Ogni misura effettuata inoltre deve essere ripetuta almeno 5 volte.

5.1 Design

I fattori utilizzati durante l'analisi, come già descritti, hanno un diverso numero di livelli. In particolare:

- **Page Type**
  - *Small* (S): 100 KB
  - *Small-Medium* (SM): 350 KB
  - *Medium-Large* (ML): 600 KB
  - *Large* (L): 800KB
- **Intensity**
  - *LOW*: 1500
  - *HIGH*: 4500

I livelli del fattore *Intensity* non sono scelti casualmente. Essi rappresentano una frazione della *usable capacity* calcolata nel Capitolo 2. A tal proposito quindi il web server deve essere necessariamente identico a quello utilizzato per il ***Capacity Test***.
Un modello che può essere utilizzato per descrivere l'esperimento è il *Two Factor Full Factor Design con Repliche*. poiché si hanno due fattori (con un numero arbitrario di livelli) e 5 ripetizioni per ogni misura. Per misura si intende la *response time*.
Tale Design prevede di effettuare le misure per ogni combinazione tra i livelli dei due

fattori; ogni misura deve poi essere ripetuta 5 volte. Posto $FACT_i$ come il numero di livelli del fattore i e R il numero di ripetizioni per ogni combinazione:

$$N = FACT_1 \times FACT_2 \times R = 2 \times 4 \times 5 = 40$$

in cui N è il numero di misure da effettuare.

5.1.1 Client - JMeter

Anche in questo caso per simulare le richieste con carico prefissato si utilizza il tool JMeter, ampiamente descritto in precedenza.
Per questo esperimento la configurazione è molto più semplice poiché bisogna porre solo una richiesta e un carico (che variano in relazione alla misura che si sta effettuando).
Ogni misura può durare anche solo 1min e il parametro di misurazione è *elapsed* (ricavato stesso dal tool).
Per ogni misura il numero di richieste varia in relazione al carico, per cui il *Response Time* può essere calcolato come la media tra tutti gli *elapsed* che compaiono nelle misure.

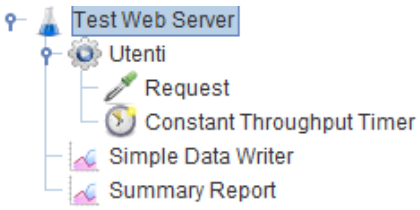


Figura 5.1: *Configurazione di JMeter*

A questo punto per permettere di automatizzare il calcolo del *response time*, i file di misurazione devono essere salvati con uno standard. Così facendo si può utilizzare MATLAB per calcolare il *response time* (come media di *elapsed*) in modo iterativo.
Ogni file viene dunque memorizzato nel seguente modo:

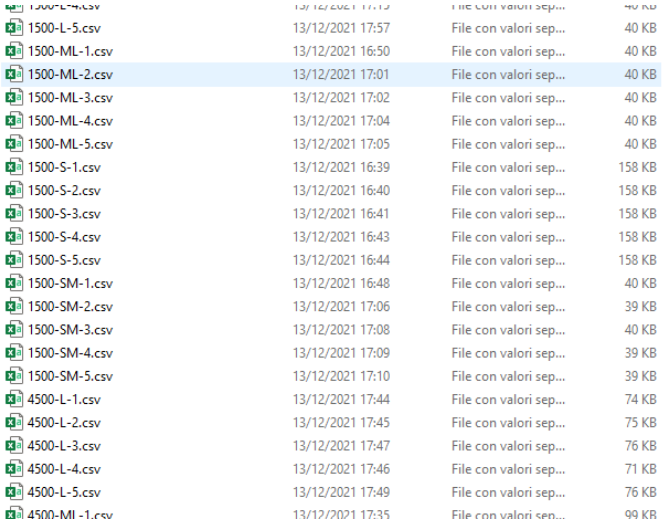


Figura 5.2: *Memorizzazione delle misure*

Il primo valore indica il carico (il livello del fattore *intensity*), il secondo valore indica la risorsa (il livello del fattore *page-type*), il terzo valore indica la ripetizione.
Dopo avere collezionato 40 file, con MATLAB si può automatizzare il calcolo:

```
intensity = [1500 4500];
page_type = ["S", "SM", "ML", "L"];
N = 5;

resp_time = zeros(length(intensity),length(page_type), N);
factor_intensity = 1;
factor_page = 1;

for i=intensity
        for p=page_type
                for r=1:N
                        path = strcat(num2str(i, '%d'),'-',p,'-',num2str(r,
↪  '%d'),'.csv');
                        data = readtable(path);
                        resp_time(factor_intensity,factor_page,r) =
↪  mean(table2array(data(:,2)));
                end
                factor_page = factor_page +1;
        end
        factor_intensity = factor_intensity +1;
        factor_page = 1;
end
```

L'output è quindi una matrice tridimensionale che associa ad ogni combinazione di livelli e ad ogni ripetizione il corrispettivo valore del *RT*.

5.2 Analisi

Il design oggetto di studio è un *Two-factor Full Factorial Design con repliche*. I fattori che lo interessano sono categorici, quindi possono assumere solo valori finiti.
Utilizzando l'output prodotto durante la fase di design e di misurazione, si può costruire una tabella che racchiude tutti i tempi di risposta y_{ijk}.
La seguente tabella descrive i tempi medi di risposta ottenuti in funzione delle combinazioni dei fattori e delle repliche:

Intensity	Small	Small-Medium	Meduim-Large	Large
1500	5,2082	23,5713	42,0207	78,6531
	5,7017	16,3747	32,3323	54,5792
	5,4777	19,1214	32,6010	39,0433
	5,9955	14,0742	36,6634	47,6427
	5,1065	16,8909	41,6693	55,1706
4500	5,1412	15,4733	380,9312	552,1893
	4,5569	18,6695	379,4760	538,0078
	4,6629	23,1843	375,8755	539,3999
	4,4400	21,3396	366,9233	575,0426
	4,3709	20,0086	368,5614	541,5188

5.2.1 Modello

Sulla base della tabella Tab.5.2 si può costruire il modello che rappresenta il design scelto:

$$y_{ijk} = \mu + \alpha_j + \beta_i + \gamma_{ij} + e_{ijk} \tag{5.1}$$

in cui:

Parametro	**Significato**	**Dimensione**
μ	Media di tutte le misure	1
α_j	Effetto del fattore *Page-Type*	a = Numero di livelli di *Page-Type*
β_i	Effetto del fattore *Intensisty*	b = Numero di livelli di *Intensisty*
γ_{ij}	Effetto dell'interazione dei due fattori	$b \times a$ = Numero di livelli di *Intensisty* per il Numero di livelli di *Page-Type*
e_{ijk}	Errore di ogni misura	$b \times a \times r$ = Numero di livelli di *Intensisty* per il Numero di livelli di *Page-Type* per il Numero di Ripetizioni

Grazie alle conoscenze teoriche acquisite durante il corso, i parametri che descrivono il modello possono essere facilmente calcolati tramite uno script MATLAB.

```
a = length(page_type);
b = length(intensity);
r = N;
%Parametri del modello
mu = 0;              %Media totale
alpha = zeros(1, a);%Effetto di PAGE TYPE
beta = zeros(1, b); %Effetto di INTENSITY
gamma = zeros(b, a);%Effetto dell'interazione
```

```
e = zeros(b, a, r); %Errore

%Calcoli
mu = 1/(a*b*r) * sum(resp_time,'all');
for j=1:a
        alpha(j) = sum(resp_time(:,j,:), 'all')/(r*b) - mu;
end
for i=1:b
        beta(i) = sum(resp_time(i,:,:), 'all')/(r*a) - mu;
end
for i=1:b
        for j=1:a
                gamma(i,j) = sum(resp_time(i,j,:),'all')/r - mu - alpha(j) -
↪  beta(i);
        end
end
for i=1:b
        for j=1:a
                for k=1:r
                        e(i,j,k) = resp_time(i,j,k) -
↪  sum(resp_time(i,j,:),'all')/r;
                end
        end
end
```

Esso fornisce come risultato:

```
alpha =

 -127.8756 -114.0710   72.7636  169.1830


beta =

 -104.0469  104.0469


gamma =

  104.4786  103.1826  -64.6012 -143.0601
 -104.4786 -103.1826   64.6012  143.0601
```

5.2.2 Importanza - Allocation of Variation

L' *importanza* di un fattore viene misurata in base alla porzione di **variazione totale** che esso riesce a spiegare.
Quest'ultima è espressa tramite la Sum of Squares Total o SST la quale ci fornisce informazioni circa quanto i dati ottenuti si discostano dal loro valore medio.

$$SST = \sum_{i=1}^{b}\sum_{j=1}^{a}\sum_{k=1}^{r}(y_{ijk} - \mu)^2 \tag{5.2}$$

con a la dimensione di α, b la dimensione di β e r numero di ripetizioni.
In particolare la variazione totale può essere anche vista come somma delle variazioni

spiegate dai fattori, dalle loro interazioni e dall'errore commesso:

$$SST = SSA + SSB + SSAB + SSE$$

La percentuale di variazione spiegata dal fattore A ad esempio è:

$$A = \frac{SSA}{SST} \times 100$$

Queste informazioni possono essere agevolmente ottenute in *JMP*, operando sulla tabella che caratterizza il design in questione, analizzando le sezioni:

1. *Analisi della varianza*
2. *Test degli effetti*

Caso di Studio

Analisi della varianza

Origine	DF	Somma dei quadrati	Media quadratica	Rapporto F
Modello	7	1527867,4	218267	3232,616
Errore	32	2160,6	68	Prob > F
C. totale	39	1530028,0		<,0001*

Test degli effetti

Origine	Nparm	DF	Somma dei quadrati	Rapporto F	Prob > F
Intensity	1	1	433030,15	6413,345	<,0001*
Page-Type	3	3	632817,86	3124,093	<,0001*
Intensity*Page-Type	3	3	462019,35	2280,895	<,0001*

Figura 5.3: *Sum of Squares in JMP*

Component	Sum of Squares	% Variation
$y - \overline{y}_{...}$	1530028,0	100
Intensity (CTTs)	433030,15	28,30
Page-Types	632817,86	41,36
Interactions	462019,35	30,20
Errors	2160,6	0,14

Osservando i risultati ottenuti si nota che la percentuale maggiore di variazione la spiega il fattore *Page Types*, con il 41,36% della totale, seguito da *Interazioni* e *Intensità del carico* che ne spiegano una percentuale più o meno simile (rispettivamente 28,30% e 30,20%). Il restante 0,14% è attribuita all'errore sperimentale.
Gli stessi risultati possono essere tranquillamente raggiunti anche con uno script MATLAB che sfrutta il modello precedentemente calcolato.

```
SSY = sum(resp_time(:,:,:).^2, 'all');
SS0 = a*b*r*mu*mu;
SSA = b*r*sum(alpha.^2);
SSB = a*r*sum(beta.^2);
SSAB = r*sum(gamma.^2, 'all');
SSE = sum(e.^2, 'all');
SST = SSY - SS0;
IMPORTANZA_PAGE_TYPE = SSA/SST;
```

```
IMPORTANZA_INTENSITY = SSB/SST;
IMPORTANZA_INTERACTION = SSAB/SST;
IMPORTANZA_ERRORE = SSE/SST;
```

Dunque i risultati di JMP possono essere integrati con questi ottenuti tramite lo script MATLAB, racchiudendo il tutto in un'unica tabella:

Component	Sum of Squares	% Variation	DF	Mean Square
y	$SSY = 2236968$		40	
$\bar{y}$	$SS0 = 706940$		1	
$y - \overline{y}_{...}$	$SST = 1530027$	100	39	
Page-Type	$SSA = 632817$	$41,36$	3	$MSA = 210939$
Intensity	$SSB = 433030$	$28,30$	1	$MSB = 433030$
Interaction	$SSAB = 462019$	$30,20$	3	$MSAB = 154006$
e	$SSE = 2160$	$0,14$	32	$MSE = 67,52$

Tuttavia l'importanza non è un concetto statistico, dunque si necessita la valutazione di un altro parametro che invece lo è, la significatività. Difatti può accadere che un fattore importante non sia significativo.

5.2.3 Significatività - Analysis of Variance

La *significatività* di un fattore, come precedentemente specificato, è un concetto statistico il quale esplicita il contributo che spiega quel fattore rispetto a quello relativo all'errore. Se un fattore è significativo, ripetendo l'esperimento, con elevata probabilità (associata al livello di significatività) esso influenzerà sempre allo stesso modo l'output, dimostrando che quindi quei risultati non sono dettati dal caso.

Gli ANOVA (Analysis of Variance) tests permettono di verificare la significatività dei fattori di un esperimento. Al pari dei test d'ipotesi discussi nel capitolo precedente, anch'essi si basano su delle assunzioni tra cui:

- **Normalità dei residui**.
- **Omoschedasticità**.

In base a se queste due condizioni sono verificate o meno, verrà utilizzato un particolare test. Nella seguente tabella sono sintetizzate tutte le combinazioni tra le varie condizioni con il relativo test da applicare:

Normality	Homoscedasticity	ANOVA Test
Verified	Verified	Parametric Homoscedastic ANOVA (F-test)
Not Verified	Verified	Non Parametric Homoscedastic ANOVA (Kruskal-Wallis test)
Verified	Not Verified	Parametric Heteroschedastic ANOVA (Welch's test)
Not Verified	Not Verified	Non Parametric Heteroscedastic ANOVA (Kruskal-Wallis or Friedman test)

Figura 5.4: *Tabella ANOVA tests*

Caso di Studio - Check Normalità

Per poter decidere quale test utilizzare per studiare la significatività di *Intensity* e *Page-Types* si deve quindi verificare innanzitutto se i residui (risposta osservata - valore previsto) provengono da una distribuzione normale.
JMP permette di ricavare la colonna dei residui automaticamente a partire da livelli - ripetizioni e output. In seguito alla generazione si può effettuare un plot della distribuzione, in particolare il *normal quantile plot*

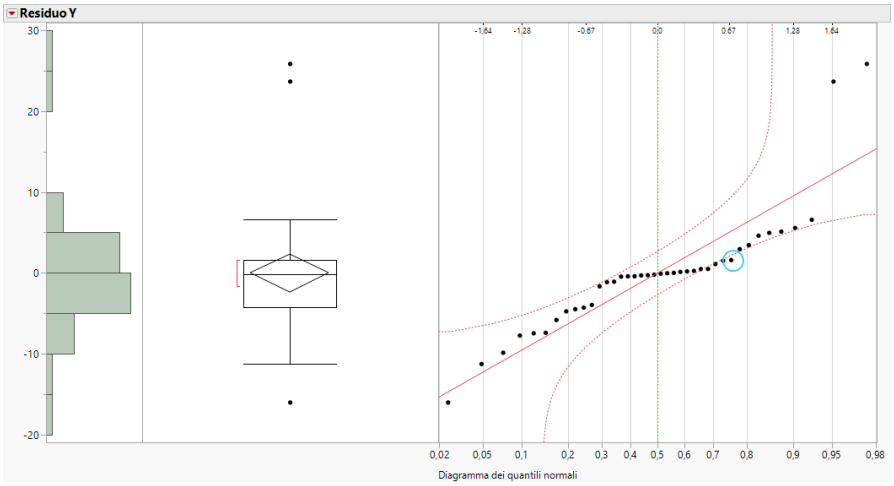


Figura 5.5: *Normal quantile plot JMP*

il quale non è altro che il q-qplot già discusso per la workload characterization.
Si assume che la distribuzione considerata non è normale se anche uno solo dei residui esce al di fuori delle bande di confidenza (tratteggiate e in rosso). In questo caso si verifica con evidenza proprio questa situazione, dunque **la normalità non è verificata**.
Si può validare ulteriormente tale assunzione attraverso il test statistico di *Shapiro-Wilk*, tenendo conto del fatto che se dovesse fornire un risultato diverso da quello del test visivo, in ogni caso sarà l'esito di quest'ultimo (del test visivo) ad essere preferito.

Test della bontà di adattamento

Test W di Shapiro-Wilk

W	Prob<W
0,832591	<,0001*

Nota: Ho = i dati provengono dalla distribuzione Normale. I p-value bassi rifiutano Ho.

Figura 5.6: *Test di Shapiro Wilk*

Il test restituisce un *P-value* basso, dunque l'ipotesi nulla è stata rigettata ad ulteriore conferma della non normalità della nostra distribuzione.

Caso di Studio - Check Omoschedasticità

L'uguaglianza delle varianze viene valutata esclusivamente per ogni fattore. Lo si fa prendendo in considerazione uno dei seguenti test:

1. *Bartlett*
2. *Levene*
3. *O'Brien*
4. *Brown-Forsythe*

i cui risultati sono agevolmente forniti da JMP.

Test	Rapporto F	Num DF	Den DF	p-value
O'Brien[.5]	64,2774	1	38	<,0001*
Brown-Forsythe	141,7879	1	38	<,0001*
Levene	203,8559	1	38	<,0001*
Bartlett	65,3794	1	.	<,0001*

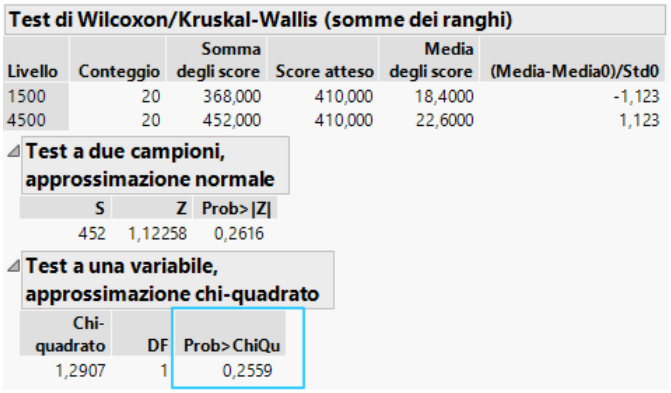
Test	Rapporto F	Num DF	Den DF	Prob > F
O'Brien[.5]	564,2359	3	36	<,0001*
Brown-Forsythe	2190,2380	3	36	<,0001*
Levene	2607,4413	3	36	<,0001*
Bartlett	51,1844	3	.	<,0001*

Figura 5.7: *Test omoschedasticità Intensity e Page Type*

Indipendentemente dal test scelto, per entrambi i fattori **non vale l'ipotesi di omoschedasticità**. Difatti il *P-value* basso ci suggerisce che l'ipotesi nulla è stata rigettata.

Caso di Studio - Check Significatività fattori

Considerando la 5.4 ci si trova nel caso 4, con normalità e omoschedasticità non verificate. I test da poter applicare fanno parte degli ANOVA non parametrici eteroschedastici: **Kruskal-Wallis test o Friedman test**. Prendendo in considerazione il primo (sarebbe stato lo stesso anche se avessimo verificato l'omoschedasticità):



Test di Wilcoxon/Kruskal-Wallis (somme dei ranghi)

Livello	Conteggio	Somma degli score	Score atteso	Media degli score	(Media-Media0)/Std0
1500	20	368,000	410,000	18,4000	-1,123
4500	20	452,000	410,000	22,6000	1,123

Test a due campioni, approssimazione normale

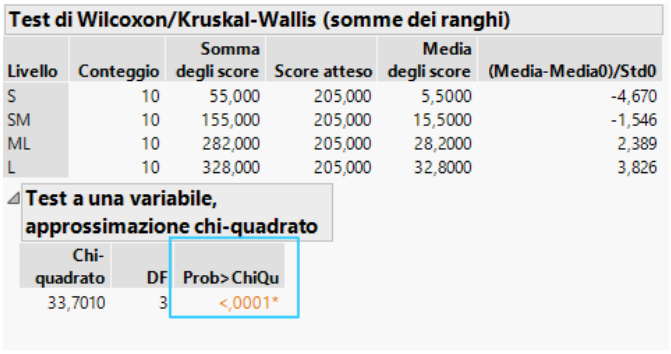
S	Z	Prob>|Z|
452	1,12258	0,2616

Test a una variabile, approssimazione chi-quadrato

Chi-quadrato	DF	Prob>ChiQu
1,2907	1	0,2559

Figura 5.8: *Test di Kruskal-Wallis Intensity*

Il valore alto (in nero) del P-value indica che il fattore Intensity non è risultato significativo ai fini dei tempi di risposta.



Test di Wilcoxon/Kruskal-Wallis (somme dei ranghi)

Livello	Conteggio	Somma degli score	Score atteso	Media degli score	(Media-Media0)/Std0
S	10	55,000	205,000	5,5000	-4,670
SM	10	155,000	205,000	15,5000	-1,546
ML	10	282,000	205,000	28,2000	2,389
L	10	328,000	205,000	32,8000	3,826

Test a una variabile, approssimazione chi-quadrato

Chi-quadrato	DF	Prob>ChiQu
33,7010	3	<,0001*

Figura 5.9: *Test di Kruskal-Wallis Page-Type*

Il P-value in questo caso assume un valore abbastanza piccolo (in arancione), dunque il fattore Page-Type oltre ad essere quello più importante risulta anche l'unico fattore significativo per l'output.
Ciò si poteva intuire perché, osservando i dati in Tab.5.2, il tempo di risposta per pagine Small e Small-Medium è molto simile indipendentemente dal carico. Non appena la dimensione varia, e quindi il tipo di pagina, il tempo di risposta aumenta/varia e i tempi di risposta associati ai due carichi sono completamente diversi.

Capitolo 6

Reliability

6.1 Esercizio 1

Calcolare **Reliability** e **MTTF** per il sistema il cui Reilability Block Diagram è rappresentato nell'immagine sottostante. Assumere che tutti i componenti sono identici e falliscono randomicamente con tasso di fallimento λ.

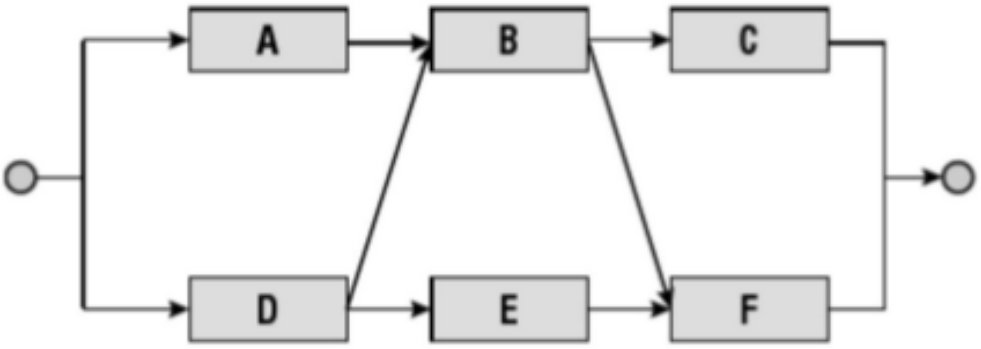


Figura 6.1: *Reliability Block Diagram Es.1*

6.1.1 Svolgimento

Dalla traccia si evince che la *reliability* di un singolo blocco del sistema in questione ha andamento esponenziale ed è:

$$R_i = e^{-\lambda * t}$$

Success Diagram

Non è ancora possibile riconoscere una serie da un parallelo in questo caso, quindi una prima cosa da fare è ricavare un *success diagram*, ovvero un sistema composto da un parallelo di serie. Le serie sono anche dette *success path* e coincidono con tutti i percorsi possibili dall'ingresso all'uscita del sistema.

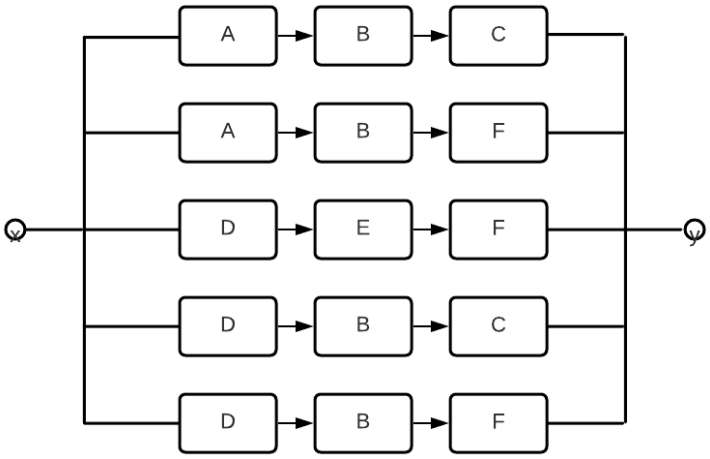


Figura 6.2: *Success Diagram Es.1*

$$Rsys \leq 1 - \prod_{i=1}^{N}(1 - Rpath_i)$$

Con N pari al numero di serie del diagramma, e $Rpath_i$ la reliability del path i-esimo. Nel caso specifico dell'esercizio:

$$Rsys \leq 1-[(1-R_A*R_B*R_C)*(1-R_A*R_B*R_F)*(1-R_D*R_E*R_F)*(1-R_D*R_B*R_C)*(1-R_D*R_B*R_F)]$$

In questo modo è possibile ricavare agevolmente una reliability che sarà un upper bound per quella effettiva del sistema. Difatti i vari path non sono tra di loro indipendenti, dato che il fallimento di un blocco potrebbe interessare più di una serie.

$$Rsys \leq 1 - (1 - e^{-3\lambda*t})^5$$

Conditioning

La tecnica del **conditioning** consente di ricavare la reliability di un sistema facendo uso della formula del teorema di Bayes:

$$P(A) = \sum_{i=1}^{N} P(A/B_i)P(B_i)$$

In poche parole, dato il sistema, viene supposto che uno dei suoi componenti R_m (quello più critico per l'analisi) sia fallito o meno. Otteniamo quindi due versioni del diagramma iniziale:

1. *con componente selezionato up (circuito chiuso).*
2. *con componente selezionato down (circuito aperto).*

$$Rsys = Rsys_1 + Rsys_2 = R_m * P(sys\,works|m\,up) + (1 - R_m) * P(sys\,works|m\,down)$$

Conditioning sul blocco E

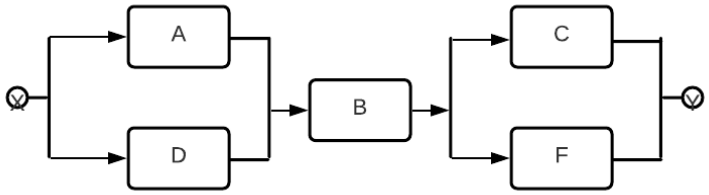


Figura 6.3: *Diagramma con Blocco E down Es.1*

Il sistema è composto da 2 paralleli in serie con il blocco B. La reliability totale può essere calcolata agevolmente sfruttando le formule di blocchi in serie e in parallelo.

$$Rsys_2 = (1 - R_E) * P(sys\,works|E\,down)$$

$$\begin{aligned}
Rsys_2 &= (1 - R_E) * \{[1 - (1 - R_A) * (1 - R_D)] * R_B * [1 - (1 - R_C) * (1 - R_F)]\} \\
&= (1 - e^{-\lambda t}) * [1 - (1 - e^{-\lambda t})^2]^2 * e^{-\lambda t} \\
&= (1 - e^{-\lambda t}) * [1 - (1 + e^{-2\lambda t} - 2e^{-\lambda t})]^2 * e^{-\lambda t} \\
&= (1 - e^{-\lambda t}) * [-e^{-2\lambda t} + 2e^{-\lambda t}]^2 * e^{-\lambda t} \\
&= (1 - e^{-\lambda t}) * [e^{-4\lambda t} + 4e^{-2\lambda t} - 4e^{-3\lambda t}] * e^{-\lambda t} \\
&= (1 - e^{-\lambda t}) * [e^{-5\lambda t} + 4e^{-3\lambda t} - 4e^{-4\lambda t}] \\
&= [e^{-5\lambda t} + 4e^{-3\lambda t} - 4e^{-4\lambda t}] - [e^{-6\lambda t} + 4e^{-4\lambda t} - 4e^{-5\lambda t}] \\
&= 4e^{-3\lambda t} - 8e^{-4\lambda t} + 5e^{-5\lambda t} - e^{-6\lambda t}
\end{aligned}$$

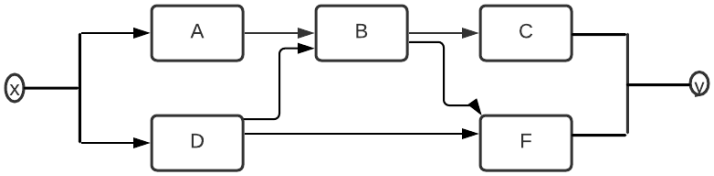


Figura 6.4: *Diagramma con Blocco E up Es.1*

$$Rsys_1 = R_E * P(sys\,works|E\,up)$$

Tuttavia il sistema così ottenuto deve ancora essere condizionato, visto che non essendo combinazione di serie e/o paralleli non è possibile ricavare immediatamente la probabilità condizionata.

Conditioning sul blocco B - E up

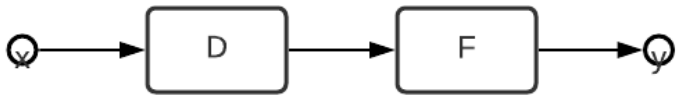


Figura 6.5: *Blocco E up e Blocco B down Es.1*

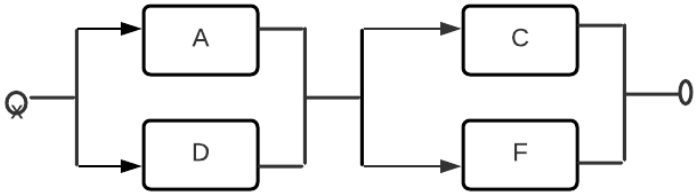


Figura 6.6: *Blocco E up e Blocco B up Es.1*

$$P(sys\,works|E\,up) = P(sys\,works|B\,down)*P(B\,down)+P(sys\,works|B\,up)*P(B\,up) =$$

$$\begin{aligned} P(sys\,works|E\,up) &= (1-R_B)*R_D*R_F + R_B*[1-(1-R_A)*(1-R_D)]*[1-(1-R_C)*(1-R_F)] \\ &= (1-e^{-\lambda t})*e^{-2\lambda t} + e^{-\lambda t}*[1-(1-e^{-\lambda t})^2]^2 \\ &= e^{-2\lambda t} - e^{-3\lambda t} + e^{-\lambda t}*[e^{-4\lambda t} + 4e^{-2\lambda t} - 4e^{-3\lambda t}] \\ &= e^{-2\lambda t} - e^{-3\lambda t} + e^{-5\lambda t} + 4e^{-3\lambda t} - 4e^{-4\lambda t} \\ &= e^{-2\lambda t} + 3e^{-3\lambda t} - 4e^{-4\lambda t} + e^{-5\lambda t} \end{aligned}$$

Dunque otteniamo che :

$$\begin{aligned} Rsys_1 &= e^{-\lambda t}*[e^{-2\lambda t} + 3e^{-3\lambda t} - 4e^{-4\lambda t} + e^{-5\lambda t}] \\ &= e^{-3\lambda t} + 3e^{-4\lambda t} - 4e^{-5\lambda t} + e^{-6\lambda t} \end{aligned}$$

Quindi la **reliability totale** del sistema sarà:

$$\begin{aligned} Rsys &= Rsys_1 + Rsys_2 \\ &= 4e^{-3\lambda t} - 8e^{-4\lambda t} + 5e^{-5\lambda t} - e^{-6\lambda t} + e^{-3\lambda t} + 3e^{-4\lambda t} - 4e^{-5\lambda t} + e^{-6\lambda t} \\ &= 5e^{-3\lambda t} - 5e^{-4\lambda t} + e^{-5\lambda t} \end{aligned}$$

Mentre il **MTTF - Mean Time To Failure** è:

$$MTTF = \int_0^\infty Rsys(t)dt = \frac{5}{3\lambda} - \frac{5}{4\lambda} + \frac{1}{5\lambda}$$

6.2 Esercizio 2

Confrontare i due schemi diversi di uno stesso sistema che sfrutta la ridondanza. Supponendo che il sistema ha bisogno di s componenti identici in serie per le proprie operazioni. Inoltre siano dati $m \times s$ componenti totali.

1. Data la reliability di un singolo componente pari a R, ricavare l'espressione della reliability delle due configurazioni.
   Per $m = 3$ e $s = 4$, confrontare le due espressioni in funzione del *mission time* t.
2. Dati i due schemi nella figura sottostante, quale avrà una reliability maggiore? Modificare lo schema che ha la reliability minore in modo da raggiungere la stessa reliability dell'altro.

Sia MTTF del singolo componente pari a *100 ore*.

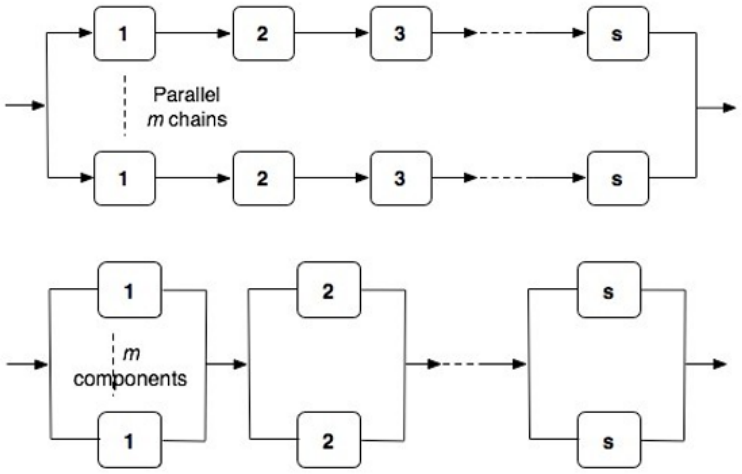


Figura 6.7: *Reliability Block Diagrams Es.2*

6.2.1 Svolgimento

In entrambi i sistemi si possono riconoscere facilmente le parti che hanno componenti in serie e in parallelo. Non c'è bisogno di utilizzare tecniche come *Legge di Bayes* o *Teorema dell'Upperbound* per ricavare la legge della reliability complessiva.

Sistema A

Per "Sistema A" si intende il sistema in alto che compare in Fig.6.2.
Per $m = 3$ e $s = 4$ esso ha 4 componenti in serie, disposti all'interno di 3 blocchi in parallelo. Quindi:

$$R_{serie} = \prod_{i=1}^{4} R_i = R^4$$

$$R_{parallelo} = 1 - \prod_{j=1}^{3}(1 - R_{serie}) = (1 - R_{serie})^3$$

$$R_A = 1 - (1 - R^4)^3$$

Sistema B

Per "Sistema B" si intende il sistema in basso che compare in Fig.6.2.
Per $m = 3$ e $s = 4$ esso ha 4 blocchi in serie, ognuno dei quali contine 3 componenti in parallelo. Quindi:

$$R_{parallelo} = 1 - \prod_{j=1}^{3}(1 - R_j) = 1 - (1 - R)^3$$
$$R_{serie} = \prod_{i=1}^{4} R_{parallelo} = R_{parallelo}^4$$
$$R_B = [1 - (1 - R)^3]^4$$

Dato che le due reliability R_A e R_B sono in funzione del *mission time* t, per confrontarle si può valutare un grafico per vari valori del tempo t. Sapendo che:

$$MTTF = 100h \quad \Rightarrow \quad \lambda = \frac{1}{100h}$$

Si ha:

$$\begin{aligned} R_A(t) &= 1 - (1 - e^{-4\lambda t})^3 \\ R_B(t) &= [1 - (1 - e^{-\lambda t})^3]^4 \end{aligned} \tag{6.1}$$

Con un semplice script MATLAB:

```
MTTF = 100; % [h]
lambda = 1/MTTF;
m = 3; s = 4;
t = 0:1:200;
R = exp(-lambda*t);  %PDF

R1 = 1-(1-R.^s).^(m);    % Reilability del Sistema 1
R2 = (1-(1-R).^m).^s;    % Reilability del Sistema 2

figure;
plot(t,R1); hold on; plot(t,R2);
legend("Sistema Serie (A)", "Sistema Parallelo (B)");
grid;
```

Fornendo come risultato il seguente grafico.

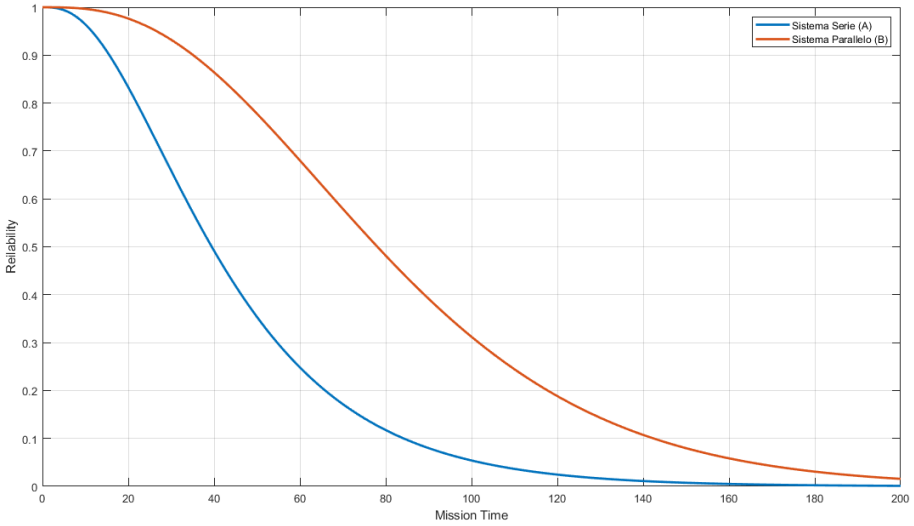


Figura 6.8: *Confronto tra le reliability*

Come si può notare il "Sistema B" ha una reliability sempre maggiore rispetto al "Sistema A", per ogni *mission time*. In realtà la stessa considerazione poteva essere fatta anche senza calcolare le due espressioni R_A e R_B, poiché il secondo sistema ha un numero di *success path* pari a m^s mentre il primo sistema pari a m. Di conseguenza essendoci più percorsi alternativi, l'affidabilità è sicuramente maggiore.

Il "Sistema A" può essere opportunamente modificato per raggiungere la stessa reliability del "Sistema B" a parità di *mission time*. Dato che il parametro s non può essere cambiato (vincolo definito dalla traccia) l'unico parametro che può variare è m, ovvero il numero di blocchi in parallelo.
R_A e R_B non sono la stessa funzione, quindi non esiste nessun valore di m per cui le due funzioni si sovrappongono. Per raggiungere la stessa reliability allora bisogna prima fissare il *mission time* e poi calcolare l'm necessario.
Per far ciò, in linea generale, posto k come il numero di blocchi-serie da mettere in parallelo deve accadere:

$$\begin{aligned} 1 - (1 - R(t)^s)^k &= R_B(t) \\ 1 - R_B(t) &= (1 - R(t)^s)^k \\ k &= \frac{ln(1 - R_B(t))}{ln(1 - R(t)^s)} \end{aligned} \tag{6.2}$$

Quindi k dipende dal *mission time*. Effettuando un grafico del valore di k per vari istanti di tempo t, nel caso in esame con $m = 3$ e $s = 4$:

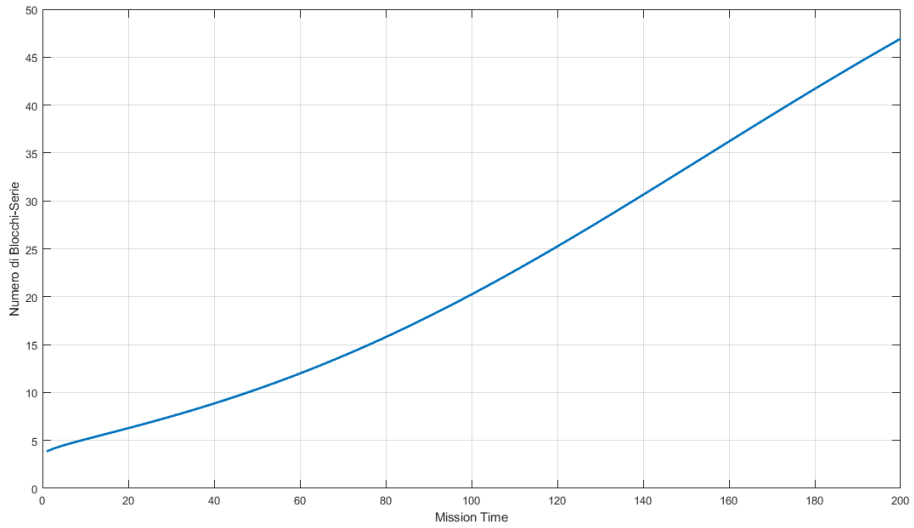


Figura 6.9: *Numero di Blocchi-Serie in funzione del Mission Time*

Se si volesse avere quindi la stessa reliability per i due sistema con $t = 100$ (ore) allora $k \approx 20$, servendosi di circa $20 \times 4 = 80$ componenti.

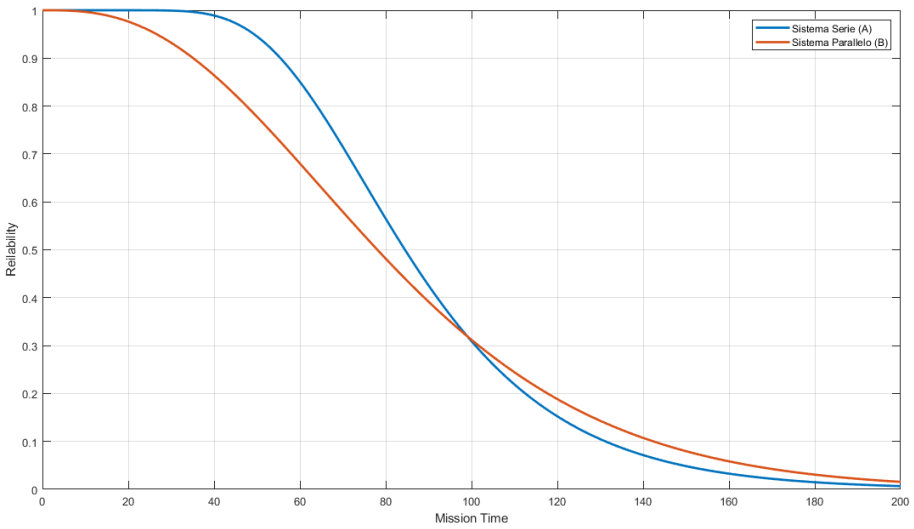


Figura 6.10: *Confronto tra le reilability dei due sistemi*

6.3 Esercizio 3

L'architettura di una rete di computer in un sistema bancario e la seguente. L'architettura è chiamata **Skip-Ring Network** ed è progettata per consentire ai componenti di comunicare anche in presenza di nodi falliti.
Per esempio, se il *Nodo 1* fallisce, il *Nodo 8* può oltrepassare il nodo fallito attraverso un link alternativo, potendo raggiungere il *Nodo 2*.

Assumendo che tutti i link sono perfetti e ogni nodo ha una reilability R_m, ricavare l'espressione della reilability della rete.
Se R_m ha una legge di fallimento esponenziale e il *failure rate* di ogni nodo è di 0.005 fallimenti per ora, determinare la reilability del sistema dopo un periodo di 48 ore.

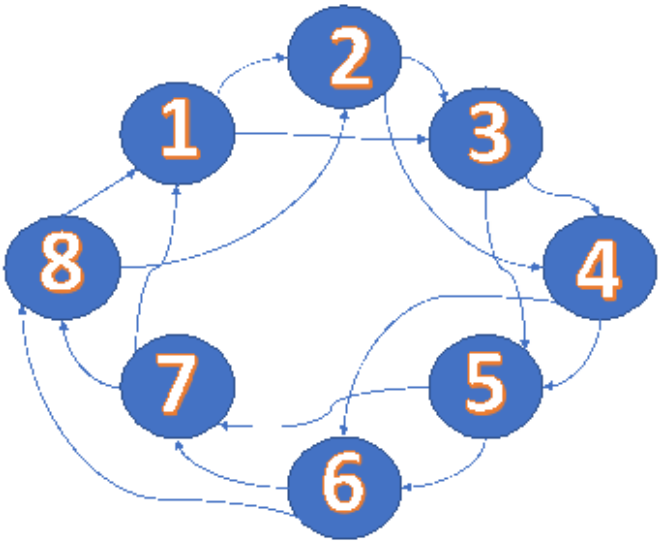


Figura 6.11: *Traccia*

6.3.1 Svolgimento

Il sistema funziona anche in presenza di nodi guasti, purché non si guastino nodi consecutivi. In tale scenario infatti l'anello viene interrotto e il sistema smette di funzionare. Sulla base di questo si può notare che il sistema continua a funzionare anche se falliscono 4 nodi, purché non siano consecutivi.
Per ricavare la legge della reilability ci si può ispirare alla legge per un sistema generale *M-out-of-N* in cui M sono il numero di nodi che devono funzionare, su N affinché il sistema continua a funzionare.
In questo caso $M = 4$ e $N = 8$, a causa del fatto che se falliscono più di 4 nodi, automaticamente almeno due nodi falliti saranno consecutivi, interrompendo l'anello.

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

in cui i indica il numero di nodi falliti e $g(i)$ indica il numero di permutazioni di quel fallimento.
Nel caso specifico:

$$R_{sys} = \sum_{i=0}^{4} g(i) R_m^{N-i} (1 - R_m)^i$$

- i=0. Nessun nodo è guasto.

$$g(0) = 1$$

Poiché è un evento che non dipende dal numero dei nodi. La reilability all'iterazione 0 vale:

$$R_{sys}[0] = R_m^8$$

- i=1. Un solo nodo è guasto. Il sistema continua a funzionare normalmente. Al massimo si possono guastare N nodi.

  $$g(1) = 8$$

  La reilability vale:

  $$R_{sys}[1] = 8R_m^7(1 - R_m)$$

- i=2. Due nodi sono guasti. In questo caso bisogna prestare attenzione. Il sistema continua a funzionare se i nodi guasti non sono adiacenti. Il numero di permutazioni vale come tutte le possibili permutazioni dei due nodi, sottratto il numero di combinazioni in cui i due nodi sono adiacenti:

  $$g(2) = \binom{8}{2} - 8 = 20$$

  Quindi:

  $$R_{sys}[2] = 20R_m^6(1 - R_m)^2$$

- i=3. Se falliscono 3 nodi, come prima, bisogna stare attenti a considerare solo le combinazioni per cui non ci siano due nodi guasti consecutivi. Al totale delle combinazioni devono essere sottratte:

  1. Combinazioni in cui 3 nodi sono adiacenti, come nel caso precedente, ovvero 8 combinazioni.
  2. Combinazioni in cui 2 nodi sono adiacenti e il terzo non è adiacente. Fissati due nodi adiacenti, il terzo non è adiacente solo in 4 casi.
     Ad esempio se il *Nodo 2* e il *Nodo 3* sono guasti, il terzo nodo deve essere tra *Nodo 5*, *Nodo 6*, *Nodo 7*, *Nodo 8*. Dato che può capitare per 8 volte, allora si hanno $8 * 4 = 32$ combinazioni.

  $$g(3) = \binom{8}{3} - 8 - 32 = 16$$

  $$R_{sys}[3] = 16R_m^5(1 - R_m)^3$$

- i=4. Quattro nodi guasti. Questo è un caso limite, poiché i quattro nodi guasti devono alternarsi tra loro, altrimenti si avranno necessariamente due nodi guasti adiacenti. Quindi esistono solo due combinazioni:

  1. Nodi: 1-3-5-7
  2. Nodi: 2-4-6-8

  $$g(4) = 2$$

  Per cui:

  $$R_{sys}[4] = 2R_m^4(1 - R_m)^4$$

Avendo calcolato il vettore $g(i)$ la reilability complessiva vale:

$$R_{sys} = R_m^8 + 8R_m^7(1 - R_m) + 20R_m^6(1 - R_m)^2 + 16R_m^5(1 - R_m)^3 + 2R_m^4(1 - R_m)^4$$

Dato che si conosce il failure rate, e ogni componente ha una legge di reilability esponenziale:

$$R_m = e^{-\lambda t} = e^{-0.005t}$$

Per cui il grafico complessivo di R_{sys} è il seguente.

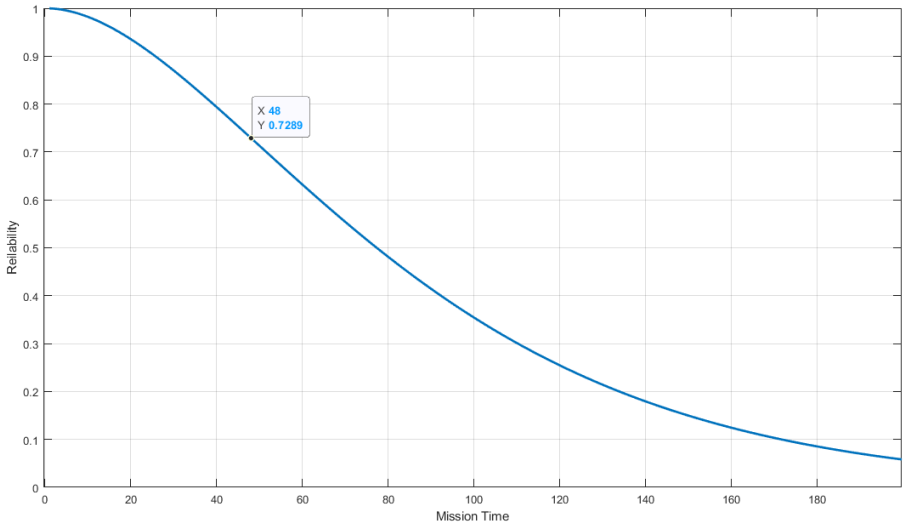


Figura 6.12: *Reilabilty del sistema*

Inoltre dopo 48h di lavoro, il sistema ha una reilability pari a:

$$R_sys(48) \approx 0,7289$$

6.4 Esercizio 4

Confrontare la reilability dei seguenti sistemi, assumendo un *Mean Time to Failure* esponenziale, con i seguenti valori:

- $MTTF_A = 1000h$
- $MTTF_B = 9000h$
- $MTTF_C = 2000h$

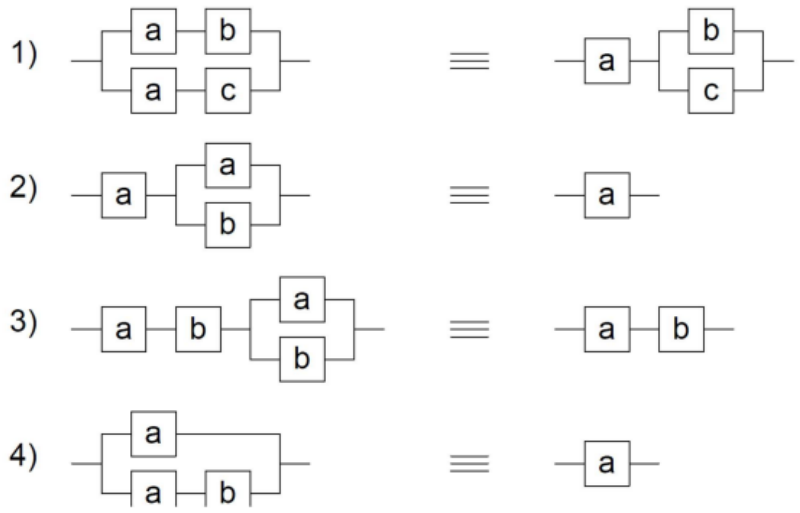


Figura 6.13: *Sistemi da confrontare*

6.4.1 Svolgimento

Innanzitutto sono state calcolati tassi di fallimento λ:

$$\lambda_A = \frac{1}{MTTF_A} = \frac{1}{1000h}$$
$$\lambda_B = \frac{1}{MTTF_B} = \frac{1}{9000h}$$
$$\lambda_C = \frac{1}{MTTF_C} = \frac{1}{2000h}$$

e di conseguenza le reliability relative ad ogni singolo componente:

$$R_A(t) = e^{-\lambda_A t} = e^{-\frac{1}{1000h}t}$$
$$R_B(t) = e^{-\lambda_B t} = e^{-\frac{1}{9000h}t}$$
$$R_C(t) = e^{-\lambda_C t} = e^{-\frac{1}{2000h}t}$$

Confronto 1

Confrontiamo il parallelo tra le serie (A e B) e (A e C), con la serie tra il blocco A e il parallelo tra i blocco B e C. Reliability **sistema 1**:

$$R11(t) = 1 - (1 - R_A * R_B) * (1 - R_A * R_C) =$$
$$1 - (1 - e^{-\frac{1}{1000h}t} * e^{-\frac{1}{9000h}t}) * (1 - e^{-\frac{1}{1000h}t} * e^{-\frac{1}{2000h}t}) = 1 - (1 - e^{\frac{1}{900h}t})(1 - e^{-\frac{3}{2000h}t}) =$$
$$e^{-\frac{3}{2000h}t} + e^{-\frac{1}{900h}t} - e^{-\frac{47}{18000h}t}$$

Reilability **sistema 2**:

$$R12(t) = R_A * [1 - (1 - R_B) * (1 - R_C)] = e^{-\frac{1}{1000h}t} * [1 - (1 - e^{-\frac{1}{9000h}t}) * (1 - e^{-\frac{1}{2000h}t})] =$$
$$e^{-\frac{3}{2000h}t} + e^{-\frac{1}{900h}t} - e^{-\frac{29}{18000h}t}$$

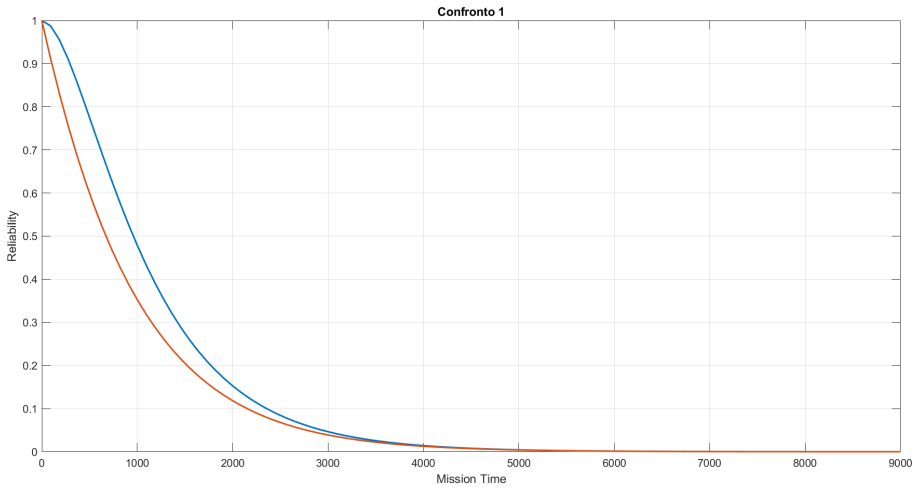


Figura 6.14: *Primo confronto*

La reliability del primo sistema (in blu nel grafico) risulta, con evidenza, maggiore rispetto a quella del secondo, fino a valori di t pari a circa 4000h.

Confronto 2

Nel secondo punto confrontiamo la reliability del blocco A messo in serie ad un parallelo tra A e B, con quella del singolo blocco A.
Reliability **sistema 1**:

$$R21 = R_A * [1 - (1 - R_A) * (1 - R_B)] = e^{-\frac{1}{1000h}t} * [1 - (1 - e^{-\frac{1}{1000h}t}) * (1 - e^{-\frac{1}{9000h}t})] = e^{-\frac{1}{900h}t} + e^{-\frac{1}{500h}t} + e^{-\frac{19}{9000h}t}$$

Reliability **sistema 2**:

$$R22 = R_A = e^{-\frac{1}{1000h}}$$

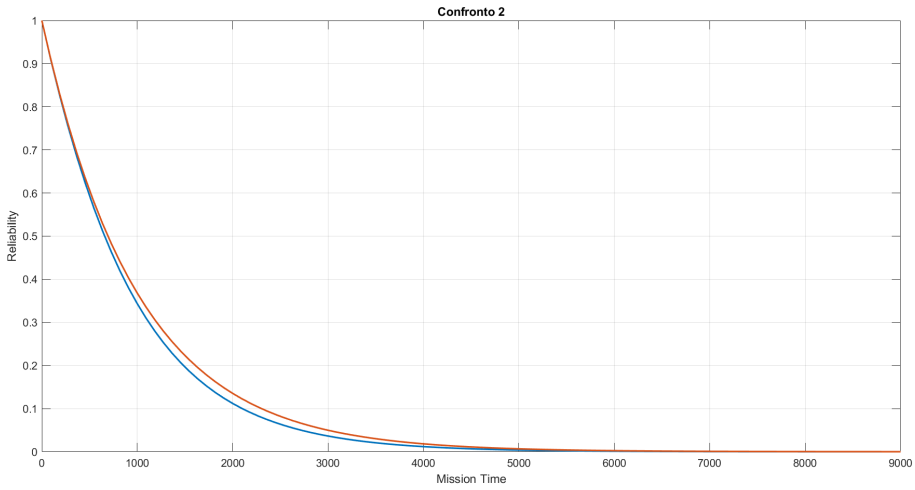


Figura 6.15: *Secondo confronto*

La reliability del blocco A, per valori di mission time compresi tra 1000h e 5000h circa, risulta essere maggiore di quella del sistema 1.

Confronto 3

In questo caso viene confrontato un primo sistema composto dai blocchi A e B in serie tra di loro e a sua volta in serie con il loro parallelo, ed un secondo sistema formato dalla semplice serie tra A e B.
Reliability **sistema 1**:

$$R31 = R21 * R_B = e^{-\frac{1}{1000h}t} * e^{-\frac{1}{9000h}t} * [1 - (1 - e^{-\frac{1}{1000h}t}) * (1 - e^{-\frac{1}{9000h}t})] = e^{-\frac{11}{9000h}t} + e^{-\frac{19}{9000h}t} + e^{-\frac{2}{900h}t}$$

Reliability **sistema 2**:

$$R32 = R_A * R_B = e^{-\frac{1}{1000h}t} * e^{-\frac{1}{9000h}t} = e^{-\frac{1}{900h}t}$$

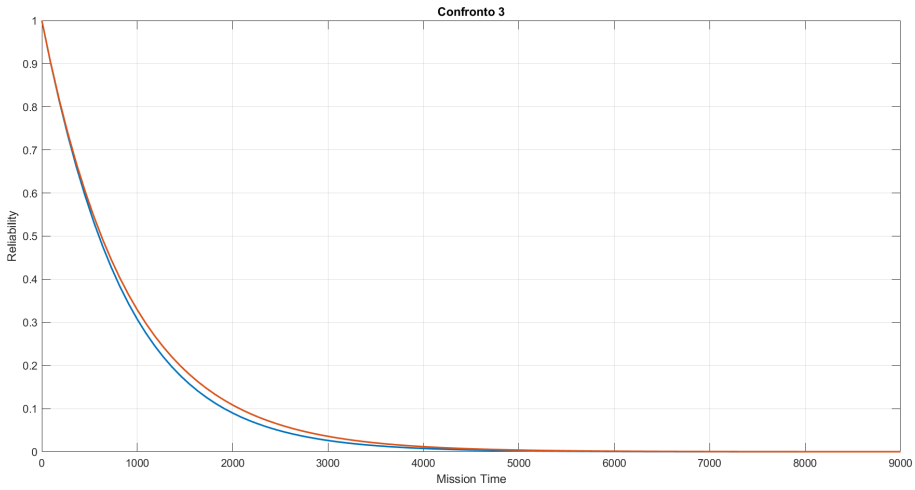


Figura 6.16: *Terzo confronto*

Nel primo sistema (in blu), la serie con il parallelo implica una riduzione della reilability per mission time compresi tra 1000h e 5000h circa.

Confronto 4

Infine viene confrontato il parallelo tra il blocco A e la serie tra A e B, con il singolo blocco A.
Reliability **sistema 1**:

$$R41 = 1 - (1 - R_A) * (1 - R_A * R_B) = 1 - (1 - e^{-\frac{1}{1000h}t}) * (1 - e^{-\frac{1}{1000h}t} * e^{-\frac{1}{9000h}t}) = e^{-\frac{1}{900h}t} + e^{-\frac{1}{1000h}t} - e^{-\frac{19}{9000h}t}$$

Reliability **sistema 2**:

$$R42 = R_A = e^{-\frac{1}{1000h}}$$

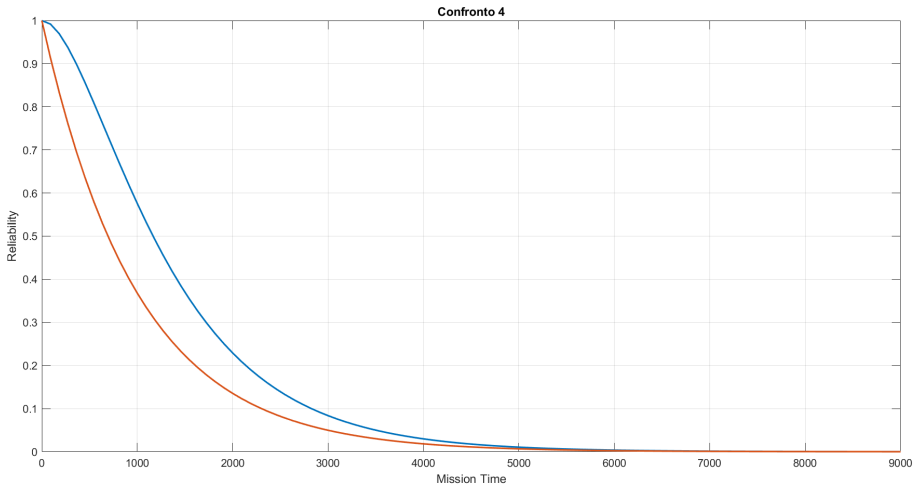


Figura 6.17: *Quarto confronto*

La reliability del primo sistema risulta incrementata rispetto a quella del blocco A, come conseguenza del parallelo tra i blocchi.

6.5 Esercizio 5

Il sistema mostrato nella figura sottostante è un sistema di elaborazione per un elicottero. Esso ha due processori e due terminali ridondanti. Sono utilizzati due bus, anch'essi dual-redundant. La parte interessante del sistema è il "navigation equipment". Il velivolo può essere completamente navigato utilizzando l' *Inertial Navigation System (INS)*. Se l'INS fallisce, il velivolo può essere navigato utilizzando la combinazione del *Doppler* con l' *Altitude Heading and Reference System (AHRS)*. Il sistema contiene 3 unità AHRS, delle quali solo una è necessaria. Questo è un esempio di ridondanza funzionale dove i dati provenienti dall'AHRS e dal Doppler possono essere utilizzati per rimpiazzare l'INS, se esso fallisce. A causa di altri sensori e strumenti, entrambi i bus sono necessari per il funzionamento del sistema a prescindere dalla modalità di navigazione.

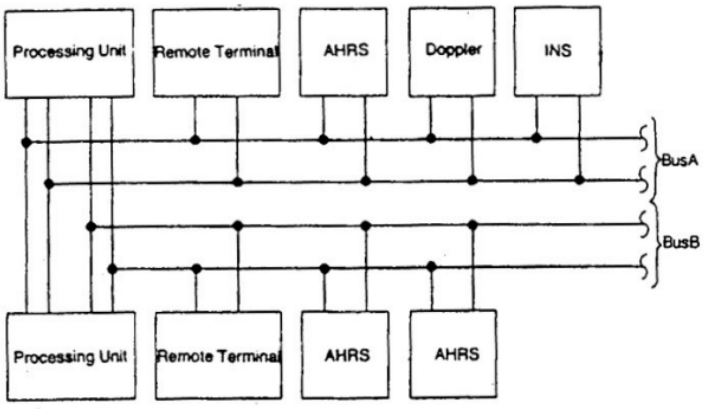


Figura 6.18: *Processing system for a helicopter*

Equipment	MTTF (hr)
Processing Unit	10000
Remote Terminal	4500
AHRS	2000
INS	2000
Doppler	500
Bus	60000

A) Disegnare il Reilability Block Diagram del sistema

B) Disegnare il Fault Tree del sistema e analizzare il *minimal cutset*

C) Calcolare la reilability per un'ora di volo utilizzando gli MTTF nella tabella seguente. Assumere che la legge di fallimento è esponenziale e che la *fault coverage* è perfetta

D) Ripetere il punto precedente, ma questa volta, considerare un fattore per la *fault detection* e la riconfigurazione delle *processing units*. Usando i dati della stessa tabella, determinare il valore approssimato di *fault coverage* che è richiesto per ottenere una reilability (al termine di un'ora) di 0.99999.

6.5.1 Punto A

Possiamo suddividere il sistema in cinque "macro-blocchi":

1. **Bus A**.
2. **Bus B**.
3. **Sistema di navigazione**.
4. **Processing unit**.
5. **Remote terminal**.

Se almeno uno di essi fallisce, il sistema intero fallisce. Ogni macro-blocco, (eccetto il sistema di navigazione per il quale è stata pensata una particolare ridondanza funzionale già descritta nella traccia), è dual-redundant. Inoltre, ciascuna delle due *Processing Unit* è connessa ad entrambi i bus, in modo da avere dual-redoundance sul singolo bus.

Partendo da queste informazioni è stato realizzato il seguente *Reliability Block Diagram*:

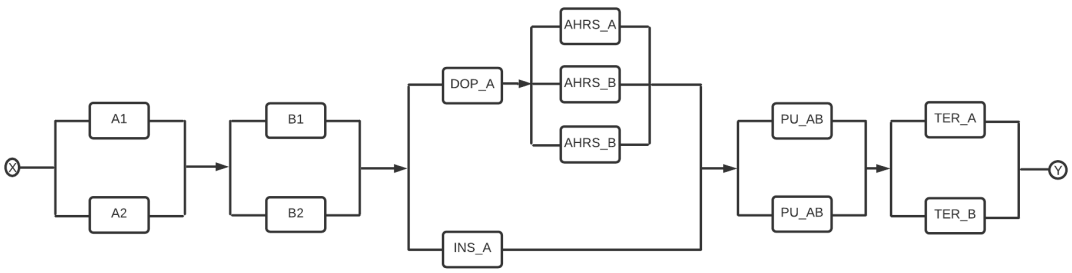


Figura 6.19: *Reliability Block Diagram*

6.5.2 Punto B

Partendo dal Reliability Block Diagram è possibile ricavare il rispettivo Fault Tree, considerando che il sistema fallisce se il *Top Level Event* ha valore logico 1.
In particolare lo serie saranno sostituite con delle porte OR, mentre i paralleli con porte END.

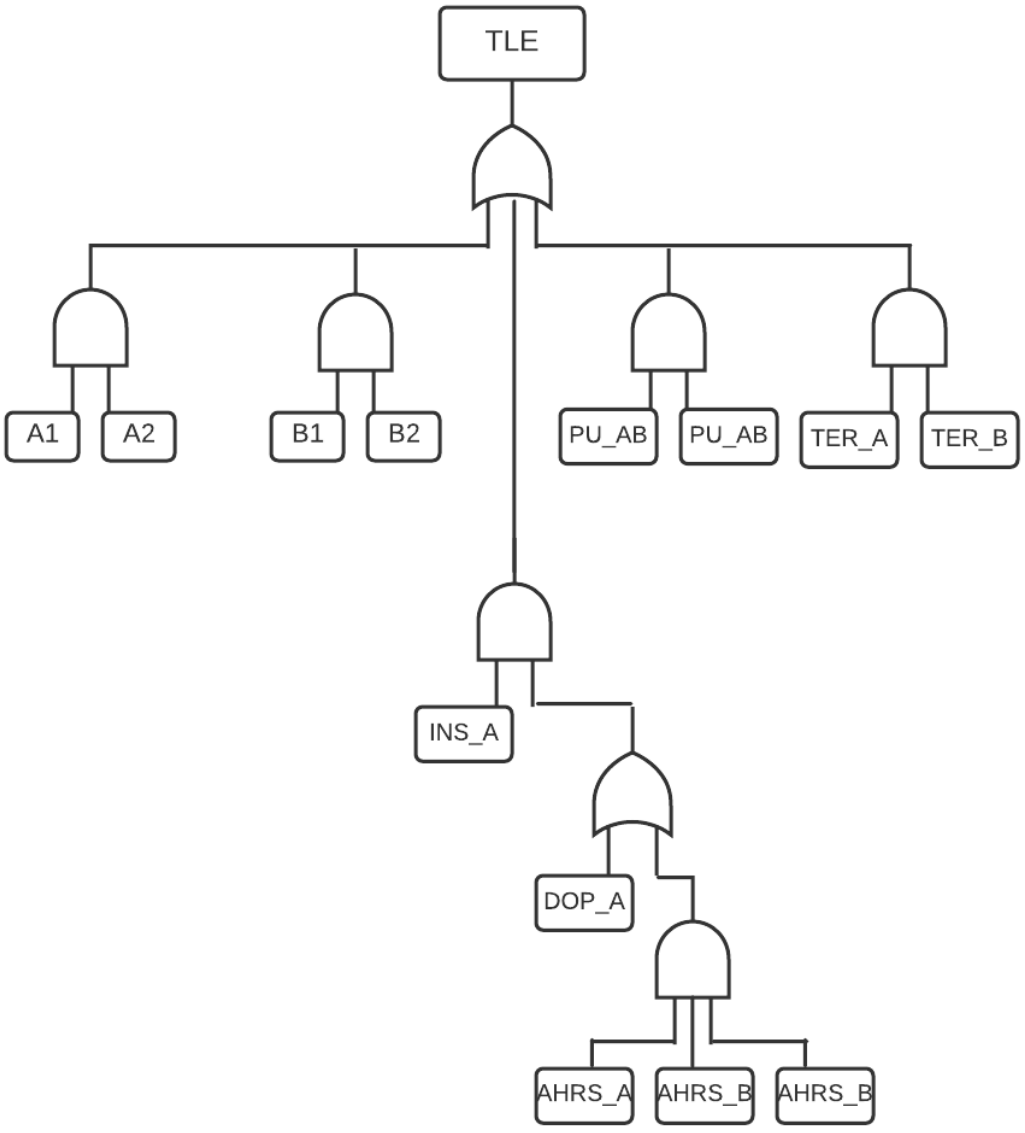


Figura 6.20: *Fault Tree*

Espressione logica del *Fault Tree*:

$$\textbf{TLE} = (A1 * A2) + (B1 * B2) + INS_A * [DOP_A + (AHRS_A * AHRS_B * AHRS_B)] + (P_AB * P_AB) + (T_A * T_B)$$
$$= (A1 * A2) + (B1 * B2) + (INS_A * DOP_A) + (INS_A * AHRS_A * AHRS_B * AHRS_B) + (P_AB * P_AB) + (T_A * T_B)$$

I *minimal cutsets* sono per definizione il minimo set di eventi di base che portano al fallimento del sistema (TLE = 1). In questo caso sono stati ricavati dall'espressione logica del Fault Tree, la quale è stata espressa mediante somme di prodotti. Tali prodotti rappresentano proprio i minimal cutsets.
Dunque il sistema fallisce, se falliscono congiuntamente:

- *Le due repliche del bus A (A1 e A2)*,
- *Le due repliche del bus B (B1 e B2)*,
- *L'INS e il Doppler*, in quanto non sarebbe possibile replicare il sistema di navigazione,
- *L'INS e i tre AHRS*, per lo stesso motivo di prima,
- *Le due repliche della Processing Unit*,
- *Le due repliche del Remote Terminal.*

Ovviamente queste sono solo le minime combinazioni di eventi che portano al fallimento del sistema, ma non le uniche.

6.5.3 Punto C

Per ogni blocco la legge descrive la reilability è di tipo esponenziale e vale:

$$R_c = e^{-\lambda_c t} \quad con \quad \lambda_c = \frac{1}{MTTF_c}$$

in cui $MTTF_c$ rappresenta il Mean Time To Failure del singolo componente c, descritto nella tabella.
A partire dalla Fig.6.5.1 si notano immediatamente componenti in serie e in parallelo. In particolare ci sono 5 blocchi in serie, per ognuno dei quali si può calcolare la reilability.

Bus A

Il primo blocco ha in parallelo le due componenti replicate del bus.

$$\begin{aligned} R_{BusA} &= 1 - (1 - R_A)^2 \\ &= 1 - (1 - e^{-\frac{t}{60000}})^2 \\ &= 2e^{-\frac{t}{60000}} - e^{-2\frac{t}{60000}} \end{aligned}$$

Bus B

Analogamente avviene per il bus B.

$$\begin{aligned} R_{BusB} &= 1 - (1 - R_B)^2 \\ &= 2e^{-\frac{t}{60000}} - e^{-2\frac{t}{60000}} \end{aligned}$$

Sistema di Navigazione

Il ramo superiore del parallelo rappresenta l'alternativa quando il componente INS si guasta. La sua reilability vale:

$$\begin{aligned} R_{secondario} &= R_{DOP} * [1 - (1 - R_{AHRS})^3] \\ &= e^{-\frac{t}{500}}[1 - (1 - e^{-\frac{t}{2000}})^3] \\ &= e^{-\frac{t}{500}}[1 - (1 - 3e^{-\frac{t}{2000}} + 3e^{-2\frac{t}{2000}} - e^{-3\frac{t}{2000}})] \\ &= e^{-\frac{t}{500}}[3e^{-\frac{t}{2000}} - 3e^{-2\frac{t}{2000}} + e^{-3\frac{t}{2000}})] \\ &= 3e^{-\frac{t}{400}} - 3e^{-3\frac{t}{1000}} + e^{-7\frac{t}{2000}} \end{aligned}$$

Il ramo inferiore rappresenta il componente principale del sistema di navigazione.

$$R_{principale} = R_{INS} = e^{-\frac{t}{2000}}$$

In definitiva:

$$\begin{aligned} R_{NAV} &= 1 - (1 - R_{secondario})(1 - R_{principale}) \\ &= 1 - \{1 - R_{DOP}[1 - (1 - R_{AHRS})^3]\}\{1 - R_{INS}\} \\ &= 1 - (1 - 3e^{-\frac{t}{400}} + 3e^{-3\frac{t}{1000}} - e^{-7\frac{t}{2000}})(1 - e^{-\frac{t}{2000}}) \end{aligned}$$

Processing Unit

L'unità di calcolo ha una struttura duale al bus, per cui:

$$\begin{aligned} R_{CPU} &= 1 - (1 - R_{pu})^2 \\ &= 2e^{-\frac{t}{10000}} - e^{-2\frac{t}{10000}} \end{aligned}$$

Remote Terminal

Analogo discorso vale per il sistema dei terminali.

$$\begin{aligned} R_{TERM} &= 1 - (1 - R_t erm)^2 \\ &= 2e^{-\frac{t}{4500}} - e^{-2\frac{t}{4500}} \end{aligned}$$

La reilability complessiva vale come il prodotto delle reilability dei singoli blocchi, essendo tutti collegati in serie.

$$R_{sys} = R_{BusA} * R_{BusB} * R_{NAV} * R_{CPU} * R_{TERM} \tag{6.3}$$

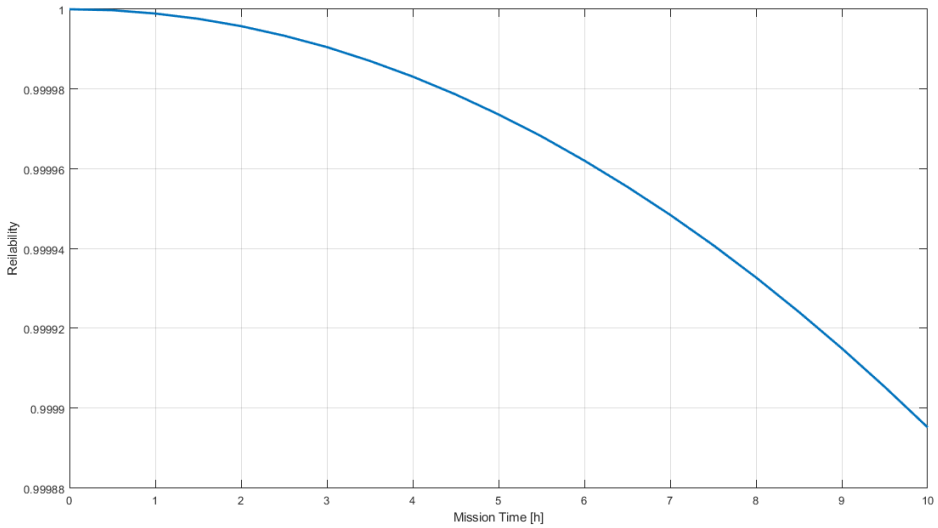


Figura 6.21: *Reilability del sistema con un intervallo di 8h*

In un'ora di volo la reilability vale:

$$R_{sys}(1) \approx 0.9999989413$$

6.5.4 Punto D

Fin ora si è supposto che tutti i fallimenti di tutti i componenti possono essere rilevati con probabilità del 100%, ma ciò non accade nella realtà. Ad esempio dal blocco della CPU si evince che se fallisce una CPU, si può utilizzare tranquillamente l'altra. Questo nell'ipotesi che il fallimento viene rilevato. Se il fallimento non viene rilevato allora si continua ad utilizzare una CPU guasta, portando ad un fallimento totale del sistema.
Per ovviare a questo problema (nel caso della CPU) si aggiunge un fattore di *fault coverage* **c** che indica la probabilità che venga rilevato un fallimento.

$$R_{CPU} = 1 - (1 - R_{pu})^2 \quad \Rightarrow \quad R_{CPU} = R_{pu} + c(1 - R_{pu})R_{pu} \tag{6.4}$$

Essa rappresenta la reilability del parallelo tra le CPU con l'aggiunta della *fault coverage*. Sostituendo la (6.4) nella (6.3) si ottiene:

$$R_{sys} = R_{BusA} * R_{BusB} * R_{NAV} * [R_{pu} + c(1 - R_{pu})R_{pu}] * R_{TERM}$$

Il valore della *fault coverage* deve essere ricavato considerando che la reilability (in un'ora) vale 0.99999.

$$c \geq \frac{\frac{R_{sys}}{R_{BusA}R_{BusB}R_{NAV}R_{TERM}} - R_{pu}}{R_{pu}(1 - R_{pu})} = 0.91057 \tag{6.5}$$

Capitolo 7

FFDA

L'obiettivo dell'homework è quello di realizzare una log-based FFDA (Field Failure Data Analysis) di sistemi complessi a larga scala. Questo tipo di analisi si realizza su dati di errori e fallimenti, collezionati monitorando il sistema durande la sua normale esecuzione. I dati, in questo caso, sono stati collezionati dai supercalcolatori **Mercury** e **Blue-Gene**.

La FFDA prevede quattro fasi di sviluppo:

1. Data Logging & Collection. È una fase che prevede il campionamento dei dati. I dati sono stati già collezionati in automatico sotto forma di file di log (file di testo).
2. Filtering. Spesso i file di log sono di grosse dimensioni, devono essere opportunamente filtrati per ricavare solo gli eventi interessati. Anche questa fase è già stata effettuata avendo a disposizione gli errori già filtrati.
3. Manipulation. I dati filtrati devono essere manipolati cercando di individuare e rimuovere gli errori sintomatici della stessa causa.
4. Data Analisys. Consiste in analisi statistiche sulle informazioni a disposizione con il fine di valutare misure quantitative.

Come già descritto la *fase 1* e la *fase 2* sono già state effettuate.

7.1 Architetture

7.1.1 Mercury cluster

L'architettura del sistema Mercury è realizzata su tre livelli:

- **"login" nodes: tg-loginx**,
- **"computation" nodes: tg-cX**,
- **"storage" nodes: tg-sX**

Il tutto è gestito da un nodo d'interfaccia: il nodo **tg-master**.

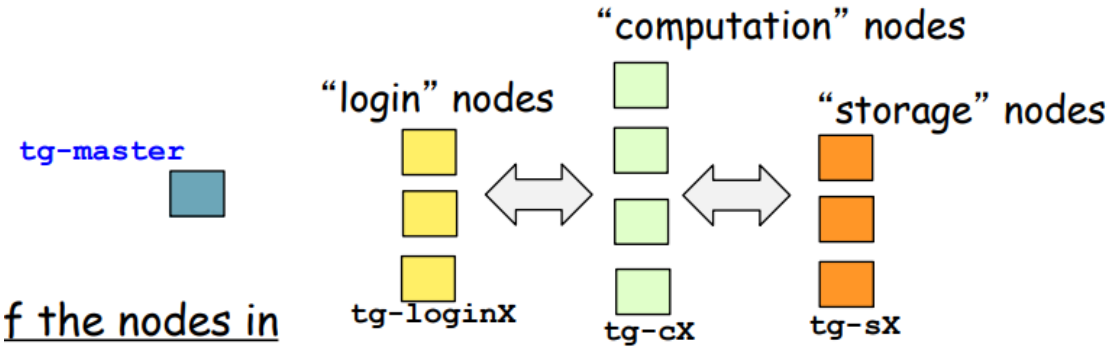


Figura 7.1: *Architettura Mercury Cluster*

Mercury Event Log

Gli eventi sono stati collezionati attraverso il *syslog daemon*. Il log fornitoci *MercuryErrorLog.txt*, è già filtrato ed è costituito da 80.854 entries. Sono stati memorizzati esclusivamente i "fatal error".
Ogni entry presenta il seguente formato:

- **1171436995 tg-c401 DEV Component Info: Vendor Id =x x, Device Id =x x, Class Code =x x, Seg Bus Dev Func =x x x x x,**
- **1168094507 tg-login3 I-O x unknown partition table**

che può essere generalizzato come segue:

- **timestamp**,
- **nodo-origine**,
- **categoria-sottosistema**,
- **breve descrizione dell'errore**

Le possibili categorie d'errore sono 5:

- **DEV**,
- **MEM**,
- **NET**,
- **I-O**,
- **PRO**

7.1.2 Blue-Gene Sample Diagram

La struttura del sistema è rappresentata dalla seguente immagine:

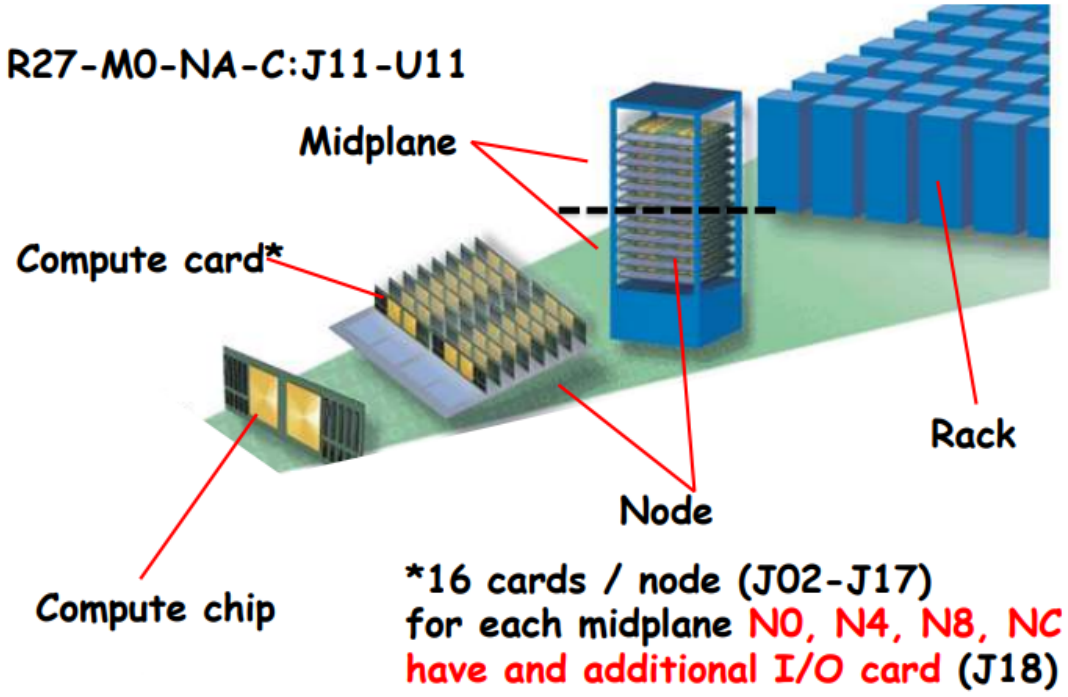


Figura 7.2: *Architettura Blue-Gene*

Esso è dunque formato da dei blocchi numerati, i *Rack*, i quali a sua volta sono suddivisi in *Midplane*, formati da più *nodi*. Su ciascun nodo sono presenti 16 *compute card* realizzate tramite dei chip.

Blue-Gene Event Log

Anche in tal caso il log fornitoci *BGLErrorLog.txt* è stato già precedentemente filtrato e contiene esclusivamente gli errori fatali.
Esso è composto da 125.624 entries, aventi il seguente formato:

- **1133835036 R52-M1-NA J03-U01 memory manager / command manager address parity..0**,
- **1133847441 R12-M0-N0 J12-U01 round toward -infinity.....0**

Generalizzando avremo:

- **timestamp**,
- **nodo origine**,
- **card origine**,
- **breve descrizione dell'errore**

7.2 Manipolazione

La manipolazione deve essere effettuata per entrambi i supercalcolatori, per cui si ha a disposizione *MercuryErrorLog.txt* (file dei log filtrato di *Mercury*) e *BGLErrorLog.txt* (file dei log filtrato di *BG/L*).
La tecnica utilizzata è la **coalescenza spaziale**. Essa prevede di raggruppare entries preveniente da più nodi diversi, cercando di capire se si sono verificate nello stesso intervallo temporale. Per farlo si utilizza una **finestra di coalescenza** W la quale raggruppa tutte le righe il cui timestamp appartiene a W. Dato che questo valore non è unico, bisogna fare delle analisi solo per sceglierlo.

7.2.1 Finestra Temporale

La finestra di coalescenza viene scelta provando a calcolare il numero di raggruppamenti, che prendono il nome di **tuple**, per diversi valori di W. Si effettua un grafico e si cerca di identificare un "knee", in modo da scegliere quel valore.
I tentativi sono stati fatti con i seguenti valori:

$$W = \begin{bmatrix} 10 & 50 & 150 & 180 & 200 & 220 \\ 230 & 240 & 250 & 290 & 390 & 800 \end{bmatrix}$$

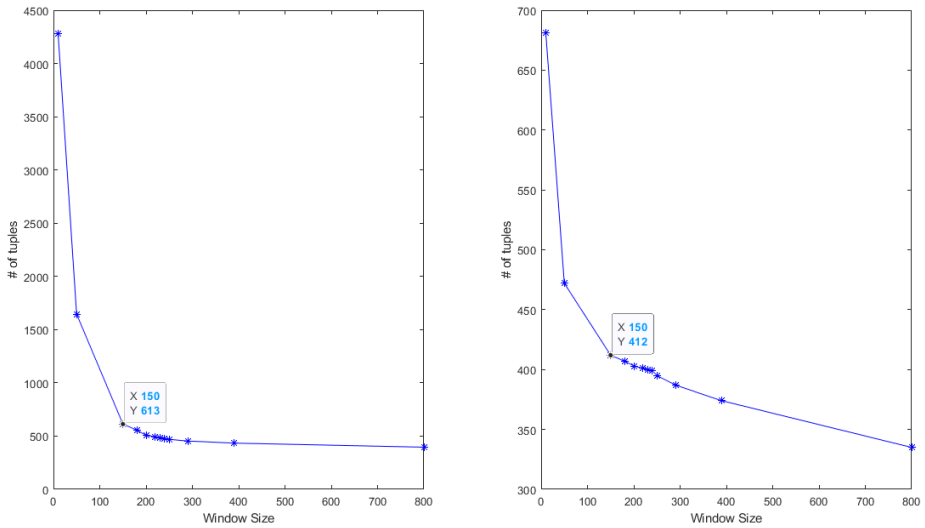


Figura 7.3: *Calcolo della Finestra Temporale*

Dai grafici si nota che il ginocchio si trova, per entrambi, in corrispondenza di una finestra temporale di *150 secondi*.
Dopo aver deciso W per entrambi i log, si possono dividere fisicamente le tuple in file diversi, ognuno quindi può essere analizzato in modo isolato. Per farlo si può utilizzare lo script **bash** messo a disposizione nel materiale *tupling_ with_ Cwin.sh* il quale prende il ingresso il percorso del file e la finestra W.
Questo tipo di approccio presenta diverse problematiche, come ad esempio:

- **Troncamento**. L'avvenimento di un *fault* può generare diversi errori che vengono poi collezionati all'interno del file di log. La suddivisione in tuple potrebbe troncare le righe corrispondenti ad uno stesso fault, andandole a dividere, e inserire, in tuple diverse.

- **Collisione**. Analogamente in una stessa tupla potrebbero comparire eventi che in realtà corrispondono a fault differenti.

7.2.2 Analisi del Troncamento

Un modo semplice per individuare la presenza di possibili troncamenti, è quello di calcolare l'intervallo temporale tra l'ultimo elemento di una tupla e il primo della tupla successiva. Se infatti il tempo di distacco è relativamente basso allora si può pensare che c'è stato un troncamento.
Tale informazione temporale viene memorizzata all'interno di un file *interarrivals.txt*, generato dopo la creazione delle tuple con lo stesso script usato precedentemente.
Un modo per visualizzare graficamente i possibili troncamenti è quello di porre sull'asse delle x le tuple che si vanno a considerare, e sull'asse delle y il numero di troncamenti conteggiati fino a quella tupla. Per conteggiare un troncamento si può porre un limite di differenza temporale tra una tupla e un altra.
Il tutto è meglio descritto con il seguente script MATLAB:

```
limit = 2*150;
interarrivi_mercury = load("tupling_MercuryErrorLog-150/interarrivals.txt");
interarrivi_bgl = load("tupling_BGLErrorLog-150/interarrivals.txt");

%% Mercury
j = 1;
hold on;
for i=1:length(interarrivi_mercury)
        if(interarrivi_mercury(i) <= limit)
                tronc_mercury(j) = i;
                plot(i,j, '-*b');
                j = j+1;
        end
end
grid;
xlabel("# di tuple");
ylabel("Numero di troncamenti");

%% BGL
figure;
j=1;
hold on;
for i=1:length(interarrivi_bgl)
        if(interarrivi_bgl(i) <= limit)
                tronc_bgl(j) = i;
                plot(i,j, '-*b');
                j = j+1;
        end
end
grid;
xlabel("# di tuple");
ylabel("Numero di troncamenti");
```

In cui il limite è posto pari al doppio della finestra temporale calcolata. Quindi si suppone che se una tupla si distacca dalla successiva di un valore inferiore a questo limite, allora è possibile che c'è stato un troncamento.

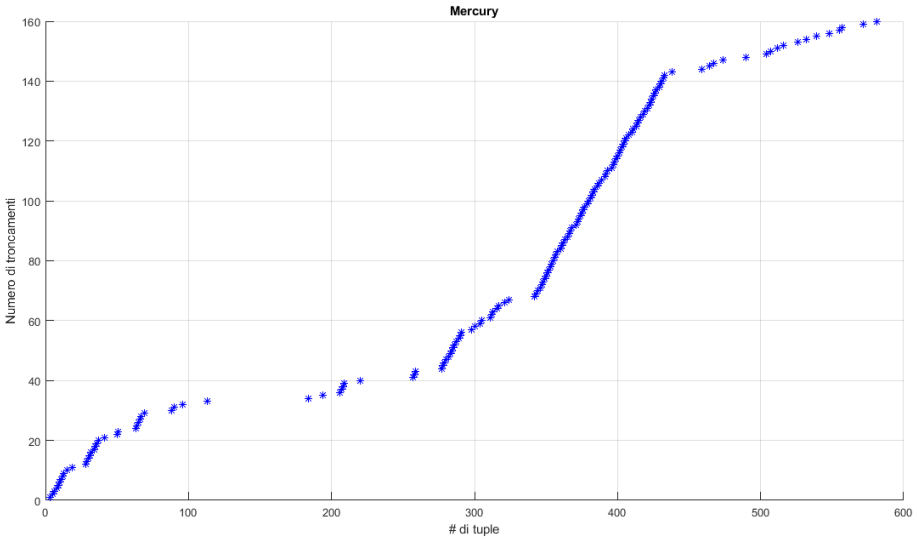


Figura 7.4: *Numero di troncamenti per tuple*

Nel caso Mercury si può notare come tra le tuple 100 e 200 circa, esse si discostano di un valore superiore al limite, portando a pensare che non ci siano stati troncamenti. Al contrario tra le tuple 350 e 400 circa, il numero di troncamenti aumenta velocemente, portando a pensare che in quella finestra temporale i log si riferivano ad uno stesso fault.

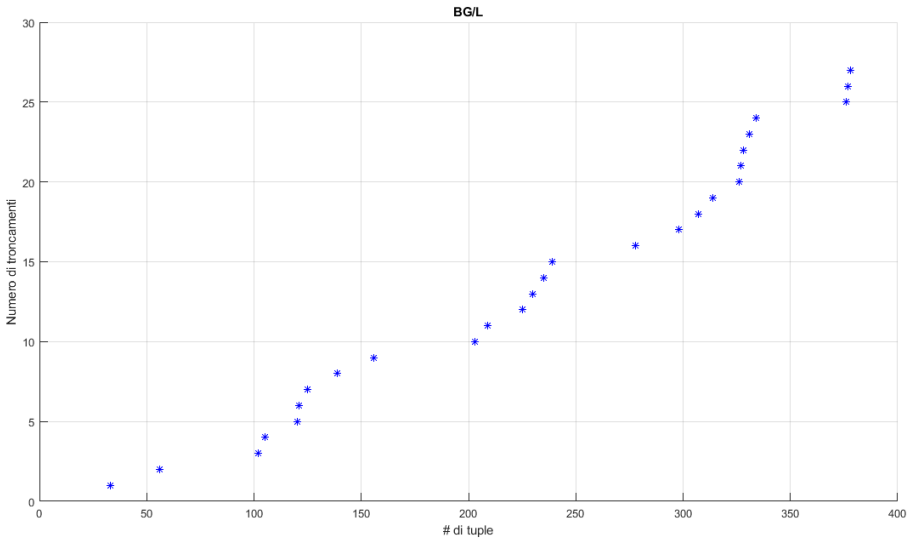


Figura 7.5: *Numero di troncamenti per tuple*

Nel caso di BG/L, invece, il numero di troncamenti sembra essere molto minore; il grafico appare meno denso rispetto a quello di Mercury.

7.2.3 Analisi delle Collisioni

Le collisioni riguardano fault che innescano errori in più nodi. Bisogna capire dunque se gli errori dei diversi nodi sono stati generati da uno stesso fallimento. Per farlo si analizzano le tuple che presentano errori collezionati da diversi nodi e si cerca di capire se effettivamente essi derivano da uno stesso fault.
Un modo per vedere graficamente la presenza di qualche collisione consiste in:

1. Scartare le tuple che hanno all'interno un solo tipo di nodo
2. Selezionare una tupla che contiene log riferiti a più nodi
3. Memorizzare tutti i timestamp della tupla in un vettore
4. Associare un valore numerico ad ogni possibile nodo del supercalcolatore in esame
5. Effettuare un grafico tra timestamp e nodo corrispettivo

Quello che ci si aspetta è che ad ogni timestamp viene associato un numero, che corrisponde al nodo corrispettivo. In MATLAB si può realizzare un algoritmo che descrive i passaggi precedenti.

```
%% Mercury
N = length(load("tupling_MercuryErrorLog-150/interarrivals.txt")) + 1;
base_path = 'tupling_MercuryErrorLog-150/tuple_';
node_list = [""];
nodes_mercury = [""];
j=1;
k=1;
o=1;
for i=1:N
        path = strcat(base_path,num2str(i, '%d'));
        current_file = fopen(path, 'r');
        while feof(current_file)==0
                row = fgetl(current_file);
                row_splitted = split(row);
                node = row_splitted(2);
                if contains(node_list,
↪   convertCharsToStrings(cell2mat(node)))==0
                        node_list(j) = convertCharsToStrings(cell2mat(node));
                        j = j+1;
                end
                if contains(nodes_mercury,
↪   convertCharsToStrings(cell2mat(node)))==0
                        nodes_mercury(o) =
↪   convertCharsToStrings(cell2mat(node));
                        o = o+1;
                end
        end
        if length(node_list) > 1
                file_list_mercury(k) = i;
                k = k+1;
        end
        node_list = [""];
        j=1;
        closeresult = fclose(current_file);
end
```

Lo script precedente calcola tutti i nodi che compaiono all'interno del log per Mercury (può essere facilmente adattato per BG/L) e calcola inoltre la lista di file (e quindi le tuple) che contengono più di un nodo.
Con tali informazioni basta creare una *map* che associa ad ogni nodo un valore, e effettuare il grafico precedentemente descritto.

```
%% Mercury
tupla = 21;

values = 1:length(nodes_mercury);
map_mercury = containers.Map(nodes_mercury,values);

base_path = 'tupling_MercuryErrorLog-150/tuple_';
path = strcat(base_path,num2str(tupla, '%d'));
hold on;
current_file = fopen(path, 'r');
while feof(current_file)==0
        row = fgetl(current_file);
        row_splitted = split(row);
        timestamp =
↪   str2num(convertCharsToStrings(cell2mat(row_splitted(1))));
        current_node =
↪   map_mercury(convertCharsToStrings(cell2mat(row_splitted(2))));
        plot(timestamp, current_node, '-*b');
end
grid;
xlabel("Timestamp");
ylabel("Nodo");
title("Mercury (tupla 21)");
```

Il cui risultato è il seguente.

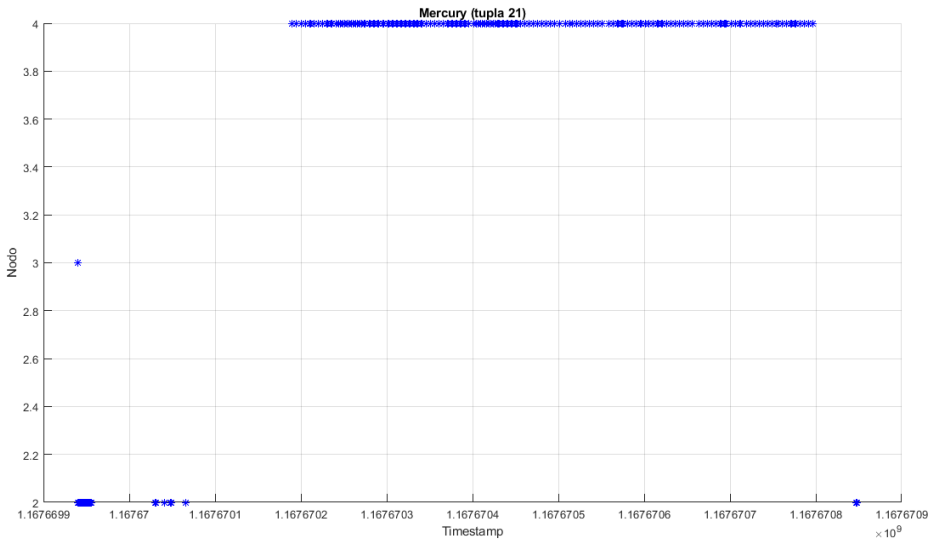


Figura 7.6: *Nodo per timestamp nella tupla 21*

Per la tupla in esame, leggendo il grafico, i nodi non si sovrappongono durante la scrittura del log. Questo potrebbe far pensare che non c'è stato un fault che si è propagato tra i nodi.
Per BG/L si può effettuare la stessa tecnica.

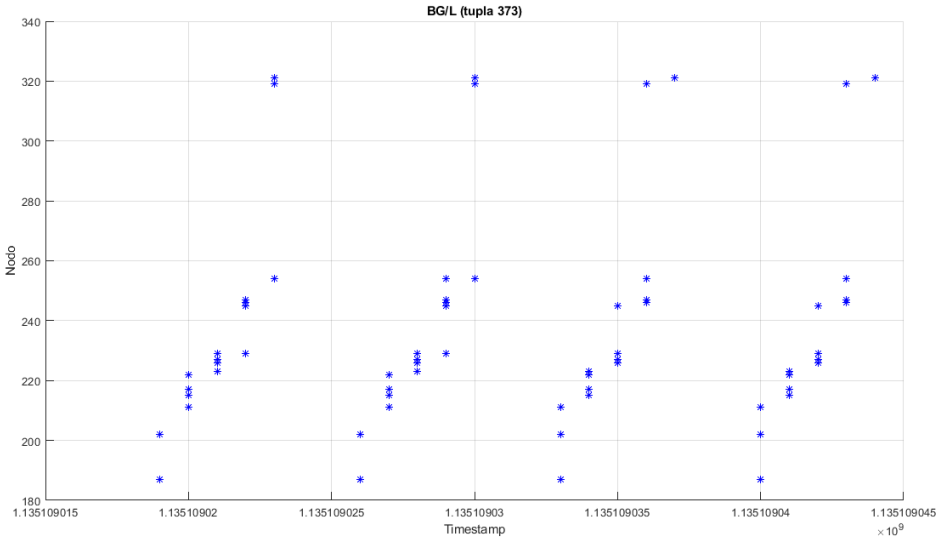


Figura 7.7: *Nodo per timestamp nella tupla 373*

Per BG/L si ha un comportamento totalmente diverso. Diversi nodi hanno elaborato un log in un arco di tempo molto breve, andandosi a sovrapporsi tra loro. Inoltre si vede proprio che temporalmente i log sono in successione, portando a pensare che uno stesso fault abbia causato errori nei nodi rappresentati.

7.2.4 Domanda 1

La stessa finestra di coalescenza può essere usata per diversi nodi (fare sia per Mercury che BG) e categorie d'errore (solo Mercury)?

Tra il materiale messo a disposizione c'è il file *filter.sh* il quale prende in ingresso un file di log e lo filtra per nodo o categoria d'errore. Quello che si deve fare è effettuare un filtering tre volte:

1. Mercury filtrato per Nodo. Sono stati scelti in esame tre nodi:
   - **tg-c401**. Nodo di calcolo
   - **tg-login3**. Nodo di login
   - **tg-master**
2. BG/L Filtrato per Nodo. Sono stati scelti anche in questo caso tre nodi:
   - **R12-M0-N0**
   - **R63-M0-N2**
   - **R71-M0-N4**
3. Mercury Filtrato per Categoria di Errore. Anche in questo caso sono stati scelte tre categorie:
   - **DEV**
   - **MEM**

- **NET**

La stessa analisi fatta nella ricerca della finestra di coalescenza deve essere fatta per questi nove nuovi file.
Il risultato per i file di log filtrati per nodo è il seguente:

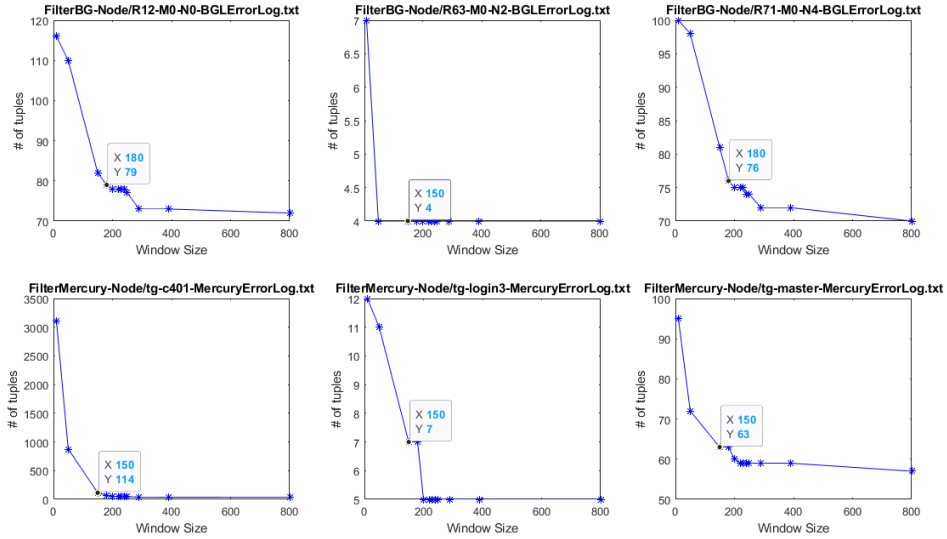


Figura 7.8: *Calcolo della Finestra Temporale per Nodo*

Come si nota dalla figura in grosso modo le finestre temporali sono le stesse del valore W scelto per i file di log completi.
Il risultato per i file di log filtrati per categoria di errore è il seguente.

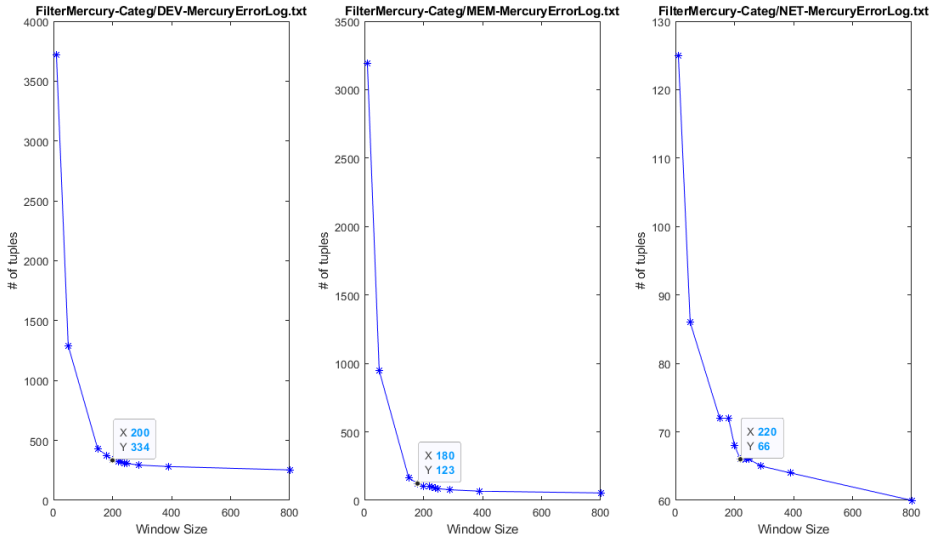


Figura 7.9: *Calcolo della Finestra Temporale per Errore*

Anche se nella figura le finestre scelte sono diverse, esse si discostano poco tra loro e soprattutto si discostano poco dal valore W scelto nella fase iniziale.

7.3 Reliability Modeling

Una volta ottenuti i dati manipolati è necessario procedere ad una loro analisi. In particolare, si parte dagli *interarrivi* (file interarrivals.txt, ottenuto alla creazione delle tuple), i quali rappresentano la "distanza temporale" tra due tuple consecutive, ovvero il TTF - Time To Failure del sistema complessivo.
L'obiettivo è valutare le distribuzioni empiriche di:

- *Tempi di Fallimento - Unreliability*,
- *Reliability*

Con il seguente script MATLAB è stata calcolata la CDF del TTF, e a partire da essa la Reliability empirica del sistema:

```
load interarrivals.txt;
[y,t] = cdfcalc(interarrivals); %ne calcolo la CDF = unreliability
empTTF = y(2:size(y,1));        %scarto la prima riga (?)
empRel = 1 - empTTF;            %Reliability
plot(t,empTTF,'-*b');
hold on;
plot(t,empRel,'-+r');
xlabel('time[s]');
ylabel('p');
legend('empTTF','empRel');
```

EmpTTF e EmpRel - Mercury

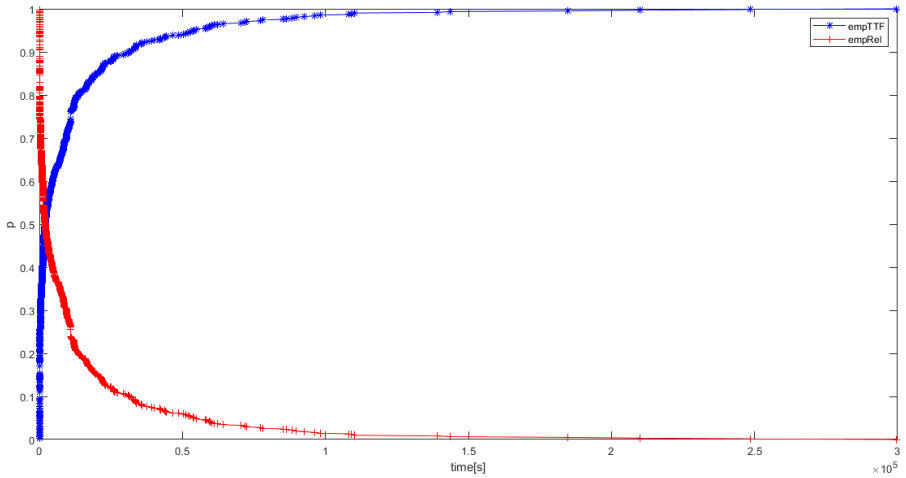


Figura 7.10: *Distribuzioni Empiriche Mercury*

EmpTTF e EmpRel - BlueGene

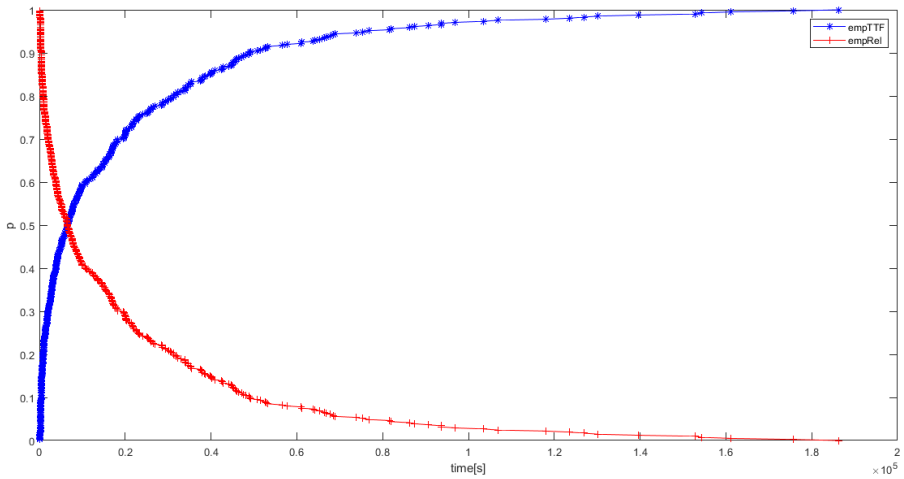


Figura 7.11: *Distribuzioni Empiriche Blue-Gene*

7.3.1 Fitting di EmpRel

Di tali distribuzioni opportunamente ottenute, bisogna effettuarne un *Curve Fitting* in modo da ricavarne un modello statistico (così facendo ho informazioni anche sul probabile andamento futuro della curva). Tale operazione risulta essere fondamentale, in quanto una data distribuzione può essere sintomatica di un particolare guasto.
A tale scopo è stato utilizzato il tool MATLAB *Curve Fitting*.

Fitting EmpRel - Mercury

Il fitting è stato eseguito con una distribuzione Esponenziale a 2 termini, descritta dalla seguente equazione:

$$f(x) = a * e^{bx} + c * e^{dx}$$

i cui coefficienti sono stati determinati tramite algoritmi interni del tool.

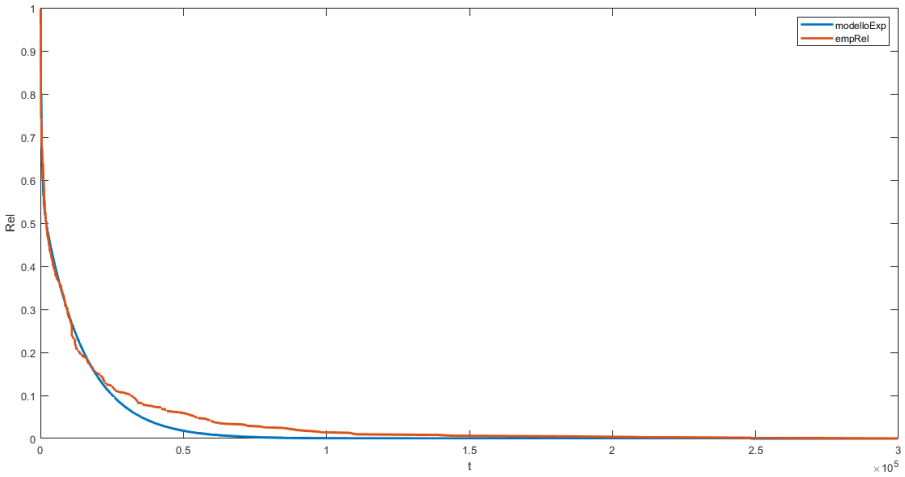


Figura 7.12: *Fitting empRel Mercury con distribuzione Esponenziale*

Tale analisi ha prodotto i seguenti risultati:

$$SSE = 0,4685$$
$$Rsquare = 0,9872$$

L' *Rsquare* è un valore abbastanza elevato, a dimostrazione del fatto che la distribuzione esponenziale selezionata spiega gran parte della variazione di quella oggetto del fitting. Come ulteriore dimostrazione della *GOF - Goodness of Fit* è stato eseguito il già precedentemente citato *Kolmogorov-Sminrov test*, il quale ha prodotto come risultato h = 0, verificando l'ipotesi nulla.

```
h = kstest2(empRel,fittedmodel(t));
```

Fitting EmpRel - BlueGene

Lo stesso procedimento descritto in precedenza è stato iterato per il supercalcolatore Blue-Gene. Anche in questo caso è stata selezionata una distribuzione Esponenziale a 2 termini, come modello che meglio approssima la Reliability empirica del sistema.

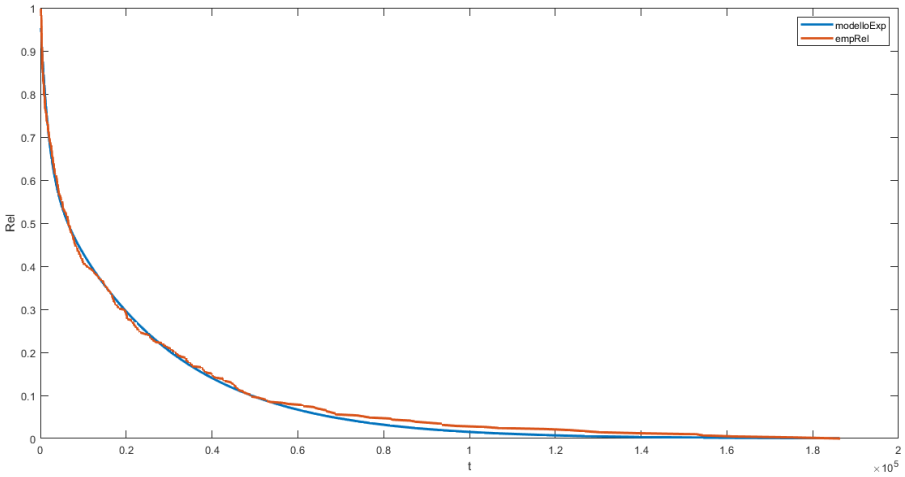


Figura 7.13: *Fitting empRel Blue-Gene con distribuzione Esponenziale*

I risultati prodotti dall'analisi sono i seguenti:

$$SSE = 0,08606$$
$$Rsquare = 0,9974$$

Anche in questo caso il kstest ha fornito output h=0, confermando la bontà del fitting eseguito.
La distribuzione esponenziale è associata a fenomeni degenerativi di tipo hardware (boh vedi meglio ste conclusioni - non sono convita che mercury sia esponenziale SSE non piccolo).

7.3.2 Analisi Reliability per Sottocomponenti

Sono stati eseguiti dei confronti tra Reliability dell'intero sistema e quella relativa ad alcuni dei suoi sottocomponenti, suddivisi per tipologia di nodo.

Mercury

Per ogni categoria di nodo (master, login, computation e storage), sono stati selezionati quelli in cui, in base al report, è stato riscontrato il numero più elevato di fallimenti, effettuando quindi dei confronti "al limite":

- **tg-master**,
- **tg-login3**,
- **tg-c401**,
- **tg-s044**

Le entries associate a ciascuno di questi nodi sono state isolate in log specifici, i quali sono stati manipolati con le tecniche già descritte per i log principali. Per tutti i nodi è stata selezionata una finestra temporale di 150. Dopodichè, dagli interarrivi relativi ai

singoli nodi sono state calcolate le Reliability Empiriche.

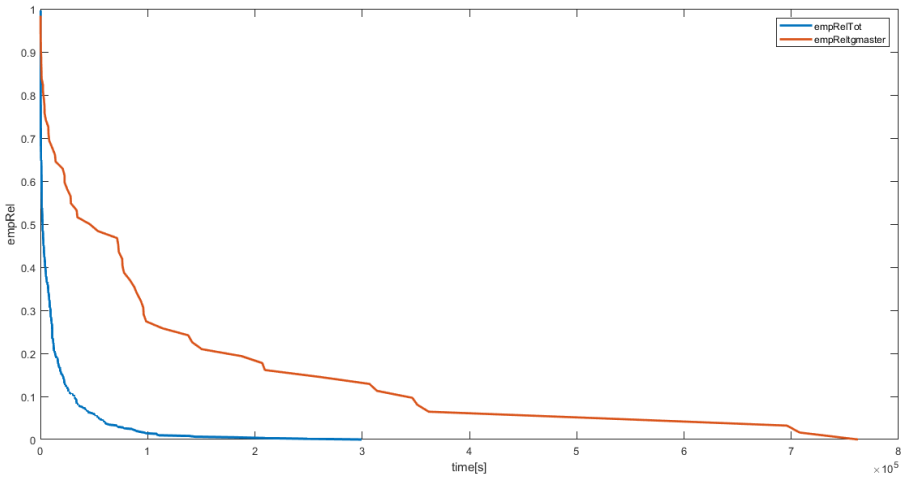


Figura 7.14: *Confronto Reliability Totale - Reliability tg-master*

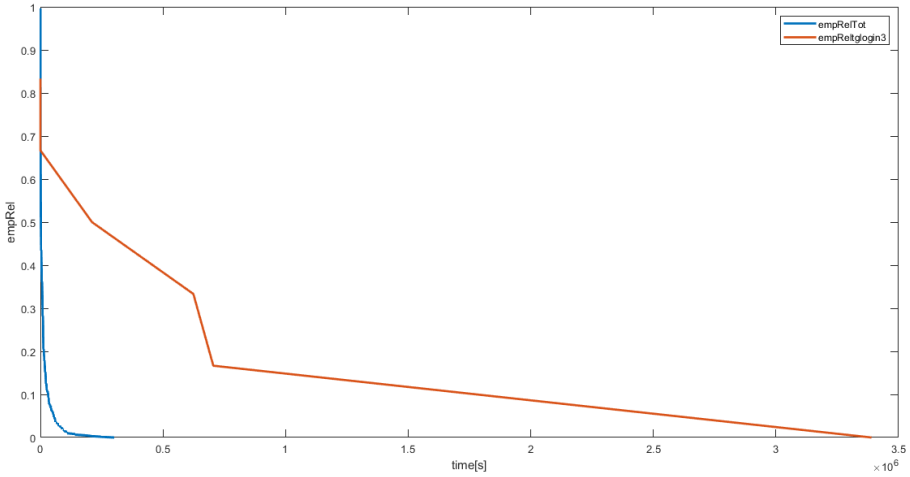


Figura 7.15: *Confronto Reliability Totale - Reliability tg-login3*

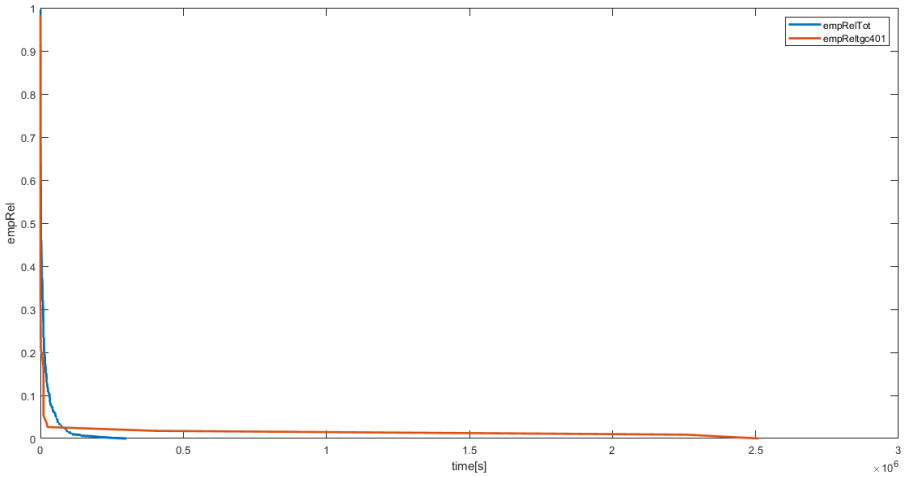


Figura 7.16: *Confronto Reliability Totale - Reliability tg-c401*

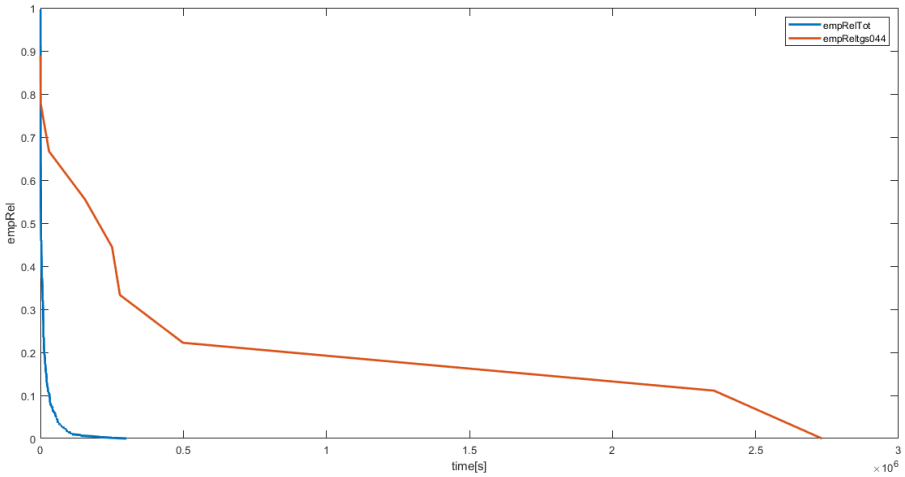


Figura 7.17: *Confronto Reliability Totale - Reliability tg-s044*

Quindi tutti i nodi eccetto il tg-c401, hanno una reliability migliore rispetto a quella totale del sistema.
Il nodo di computation **tg-c401** per tempi brevi in confronto ai totali, risulta essere il meno reliable, addirittura anche rispetto al sistema. Ciò significa che esso potrebbe essere un probabile collo di bottiglia per il supercalcolatore Mercury. Quest'ultima affermazione è molto probabile in quanto tale nodo è quello caratterizzato dal maggior numero di fallimenti.
In seguito è stato riportato un grafico che confronta tra di loro le reliability dei diversi nodi selezionati:

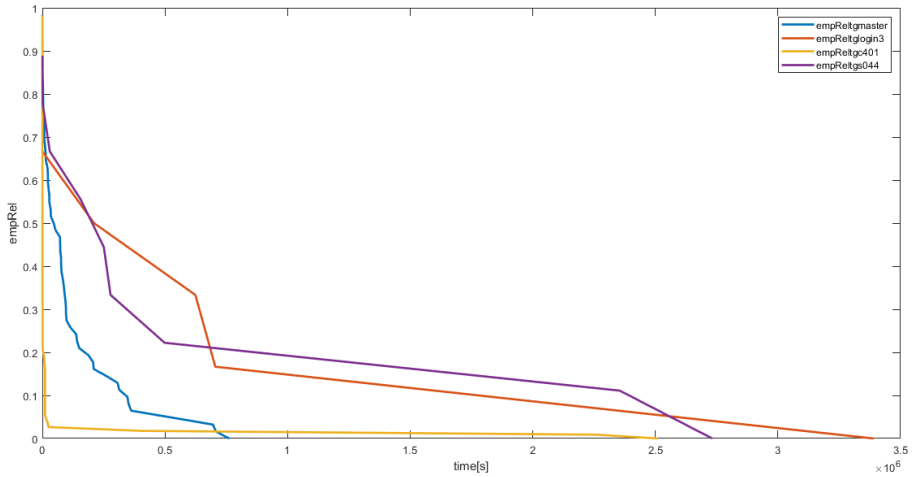


Figura 7.18: *Confronto Reliability Nodi Mercury*

Da come si può notare, i nodi con maggiore reliability sono quelli di storage e login. La reliablity del nodo master, per tempi brevi rispetto ai totali, risulta superare quella del nodo di computation, anche se a lungo andare tende più velocemente a zero se confrontata con quella di quest'ultimo.

Un altro confronto è stato effettuato sulla reliability di nodi funzionalmente simili tra di loro.
Per tale analisi sono stati selezionati tre nodi computation:

- **tg-c401**, il nodo con il numero maggiore di fallimenti,
- **tg-c238**,
- **tg-c242**.

Gli ultimi due presentano un numero di fallimenti molto simile tra di loro.

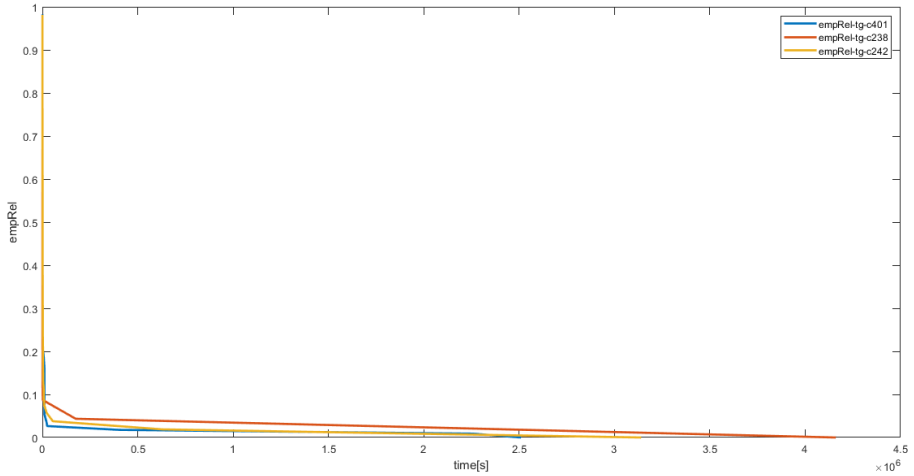


Figura 7.19: *Confronto Reliability Nodi Computation Mercury*

La reliability del nodo 401 risulta essere sempre peggiore delle altre, anche se nonostante ciò gli andamenti risultano essere tutti molto simili tra di loro (rispettivamente 1273 e 1067). Tutti i nodi computation analizzati hanno dunque una reliability molto bassa.

Blue-Gene

L'analisi è stata ripetuta anche per i primi 3 nodi con più entries di fallimenti di Blue-Gene. Essi, ordinati per numero di entries, sono:

- **R71-M0-N4**,
- **R12-M0-N0**,
- **R63-M0-N2**

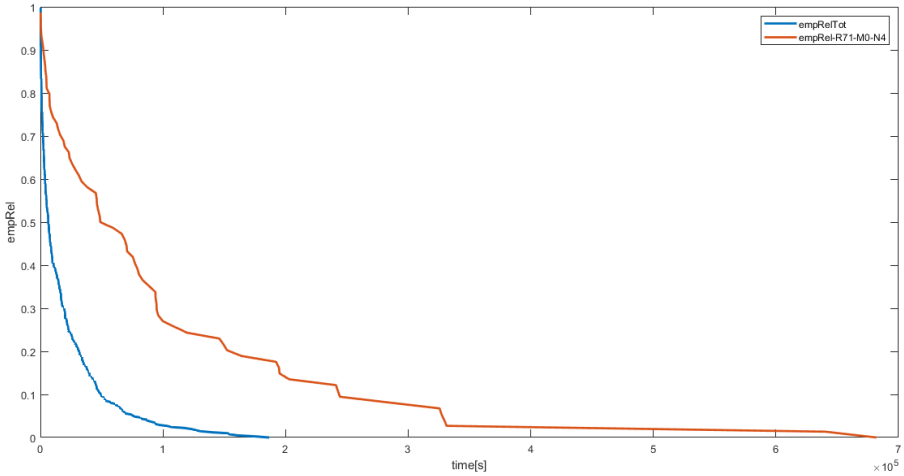


Figura 7.20: *Confronto Reliability Totale - Reliability R71-M0-N4*

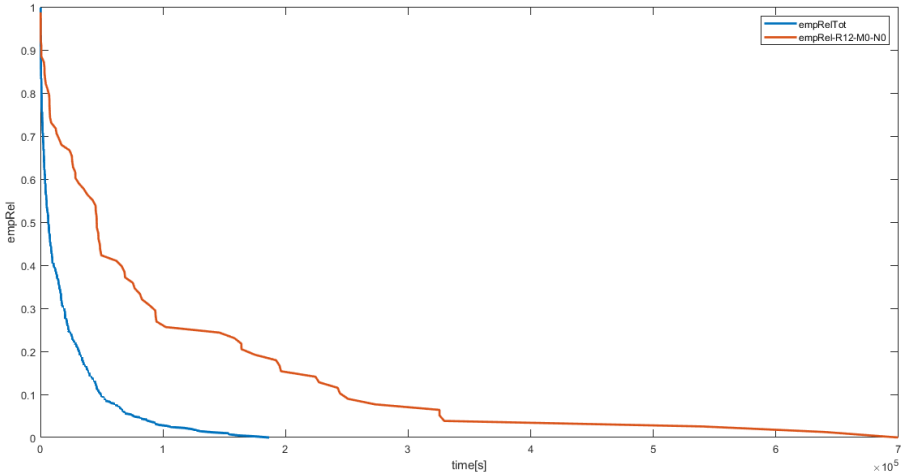


Figura 7.21: *Confronto Reliability Totale - Reliability R12-M0-N0*

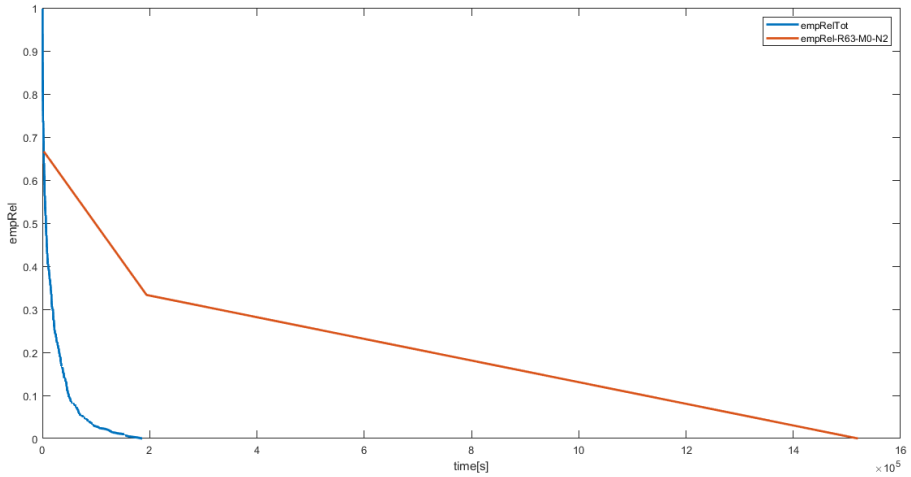


Figura 7.22: *Confronto Reliability Totale - Reliability R63-M0-N2*

In tal caso non risultano esserci evidenti colli di bottiglia, e la reliability del singolo nodo risulta essere sempre maggiore di quella totale del sistema.
Anche per Blue-Gene si riporta un grafico in cui vengono messe a confronto le reliability dei 3 differenti nodi:

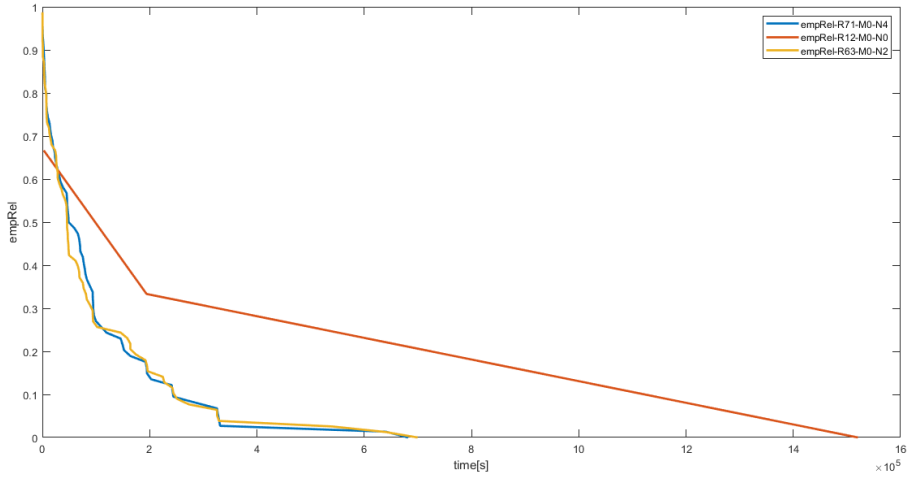


Figura 7.23: *Confronto Reliability Nodi Blue-Gene*

In tal caso i nodi confrontati sono tutti funzionalmente simili. Come si evince dal grafico la reliability del nodo **R71-M0-N4** e **R63-M0-N2** risulta avere un andamento molto simile, a differenza del terzo nodo il quale è più reliable dei precedenti.

7.4 Analisi relazione tipologia errore-nodo (Mercury)

Infine, esclusivamente per il supercalcolatore Mercury, si è voluta dimostrare la relazione tra tipologia d'errore e nodo. Si è subito notato che su nodi funzionalmente simili si

presentavano principalmente le stesse categorie d'errore. In particolare, per dimostrare ciò, sono stati considerati 4 nodi *computation*:

- **tg-c894**,
- **tg-c401**,
- **tg-c117**,
- **tg-c572**

e 4 nodi *login*:

- **tg-login1**,
- **tg-login2**,
- **tg-login3**,
- **tg-login4**

Dopo aver filtrato le entries associate a ciascuno di questi nodi, su di esse è stato applicata la seguente funzione MATLAB:

```
function [device_list] = search(file)
fid = fopen(file,'r');
device_list = [""];
device_mercury = [""];
j=1;
k=1;
o=1;
if (fid==-1)
disp('Unable to open the file')
else
i=1;
while feof(fid)==0
%Read one line into a string file
linea=fgetl(fid);
aline=split(linea);
device(i)=convertCharsToStrings(cell2mat(aline(3)));
if contains(device_list, device(i))==0
device_list(j) = device(i);
j = j+1;
end
if contains(device_mercury, device(i))==0
device_mercury(o) = device(i);
o = o+1;
end
i=i+1;
end
closeresult=fclose(fid);
if closeresult~=0
disp('Unable to close the file')
end
end
end
```

il cui obiettivo è quello di isolare il vettore *device(i)*, in cui sono memorizzate le differenti tipologie di errore relative a ciascuna entry del particolare nodo selezionato. Per ogni categoria d'errore riscontrata, ne è stata, in seguito, memorizzata la frequenza d'occorrenza:

```
function [] = istogramma(device_list,device)
[d1,d2] = size(device);
[d3,d4] = size(device_list);
num = zeros(1,d4);

for i = 1:d2
for j = 1:d4
if device(i) == device_list(j)
num(j) = num(j) + 1;
end
end
end
histogram('Categories',device_list,'BinCounts',num);
end
```

Il tutto è stato graficato tramite degli istogrammi, i quali mostrano per ogni categoria d'errore riscontrata, la frequenza con cui esso si verifica.

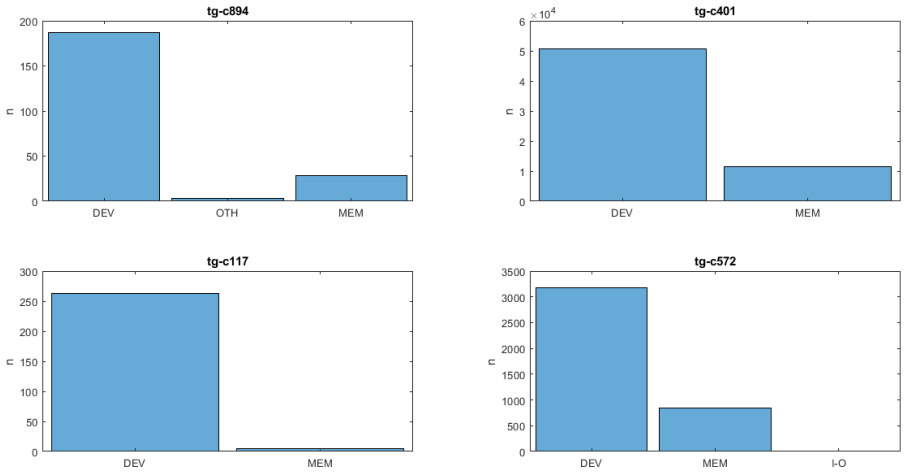


Figura 7.24: *Categorie d'errore per nodi computation*

Come si può notare nei nodi computation, la categoria d'errore più frequente è quella relativa ai device fisici, seguita da quelli sulla memoria.

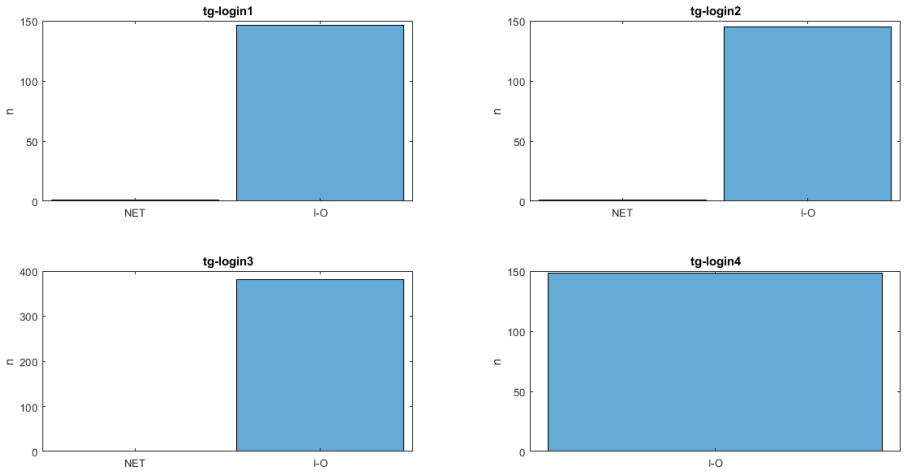


Figura 7.25: *Categorie d'errore per nodi login*

Le categorie d'errori più frequenti nei nodi di login, come atteso, sono quelle sull' Input/Output (connessioni dall'esterno - fisiche) e quelle network (connection down). Gli errori di I/O sono di gran lunga più frequenti rispetto a quelli di rete.