

**Corso di Laurea Magistrale
in
Ingegneria Informatica**

**Impianti di Elaborazione
Elaborato d'esame**

prof. Domenico Cotroneo

M63000989 Fabio d'Andrea
M63000986 Guido Di Chiara



Università degli Studi di Napoli Federico II
SCUOLA POLITECNICA E DELLE SCIENZE DI BASE
Anno Accademico 2020/2021
Primo Semestre

Indice

1 Benchmark	1
1.1 Traccia	1
1.2 Soluzione	1
1.2.1 Sistema 1	2
1.2.2 Sistema 2	5
1.2.3 Confronto	8
2 Workload Characterization	13
2.1 Traccia	13
2.2 Workload characterization	13
2.2.1 Preprocessing	13
2.2.2 Principal Component Analysis (PCA)	16
2.2.3 Clustering	18
2.2.4 Analisi della devianza	20
2.3 Trade-off	23
2.3.1 Devianza persa	23
2.3.2 Dimensione workload sintetico	24
2.3.3 Devianza persa vs. Dimensione del workload sintetico	25
3 Web Server Performance Analysis	26
3.1 Traccia	26
3.2 Introduzione	26
3.3 Workload characterization	28
3.3.1 Generazione workload reale	28

3.3.2	Caratterizzazione del workload reale	32
3.3.3	Applicazione del workload sintetico di alto livello	37
3.3.4	Caratterizzazione del workload sintetico	37
3.3.5	Validazione del workload sintetico di alto livello	38
3.4	Capacity test	40
3.4.1	Risorse Small	41
3.4.2	Risorse Large	42
3.4.3	Risorse Random	43
3.5	Design of Experiment (DoE)	44
3.5.1	Effetti dei livelli dei fattori	45
3.5.2	Errori	47
3.5.3	Importanza	47
3.5.4	Significatività	48
4	Reliability	53
4.1	Esercizio 1	53
4.1.1	Traccia	53
4.1.2	Soluzione	53
4.2	Esercizio 2	58
4.2.1	Traccia	58
4.2.2	Soluzione	59
4.3	Esercizio 3	61
4.3.1	Traccia	61
4.3.2	Soluzione	62
4.4	Esercizio 4	64
4.4.1	Traccia	64
4.4.2	Soluzione	65
4.5	Esercizio 5	69
4.5.1	Traccia	69
4.5.2	Soluzione	70

5 Field Failure Data Analysis	75
5.1 Traccia	75
5.2 Field Failure Data Analysis (FFDA)	75
5.3 Mercury	76
5.3.1 Sistema	76
5.3.2 Sottosistemi	85
5.3.3 Tipologie di nodi	89
5.3.4 Nodi critici	93
5.3.5 Bottleneck	98
5.3.6 Ulteriori analisi	100
5.4 Blue Gene/L	101
5.4.1 Sistema	102
5.4.2 Rack critici	108
5.4.3 Tipologie di card	117
5.4.4 Nodi critici	122
5.5 Confronto tra Mercury e Blue Gene/L	127

Capitolo 1

Benchmark

1.1 Traccia

Confrontare due sistemi differenti utilizzando il benchmark *Intel LINPACK Benchmark*. Realizzare il confronto per le seguenti dimensioni: 1000, 5000, 10000 e 20000.

1.2 Soluzione

Come richiesto dalla traccia sono stati confrontati due sistemi diversi, eseguendo su ciascuno di essi il benchmark *LINPACK* fornito da Intel, per le diverse dimensioni indicate.

Tale benchmark consiste nella risoluzione di un sistema di equazioni lineari, la cui matrice associata è una matrice quadrata con dimensione pari alla dimensione specificata. La risoluzione di sistemi lineari comporta l'utilizzo intensivo della floating point unit del processore e consente quindi di stimarne le prestazioni. In particolare, il benchmark restituisce alcuni parametri di interesse fra cui:

- *Time(s)*, tempo di esecuzione del benchmark espresso in secondi;
- *GFlops* raggiunti nella risoluzione del sistema lineare;

Nel caso specifico, il benchmark è stato utilizzato per comparare due sistemi dotati di due processori Intel differenti:

- il primo sistema possiede un processore *Intel Core i7-6500U*, caratterizzato da 2 core e 4 thread;
- il secondo sistema possiede invece un processore *Intel Core i7-7700HQ*, caratterizzato da 4 core e 8 thread.

Dalle specifiche dei due processori, ci si aspetta che il secondo, ovvero l'*i7-7700HQ*, abbia prestazioni migliori rispetto al primo.

Per ogni dimensione fissata del benchmark sono state raccolte 5 osservazioni, eseguendo un singolo test e riavviando ogni volta il sistema. Tale procedimento assicura che le

5 osservazioni raccolte per ogni dimensione siano **i.i.d.**¹ (indipendenti ed identicamente distribuite). In questo modo sono verificate le ipotesi del **teorema del limite centrale**, il quale garantisce che al tendere ad infinito della dimensione di un campione, la media campionaria tende ad una distribuzione normale. Di conseguenza, essendo valida l'ipotesi di normalità è possibile applicare test parametrici.

Le 5 osservazioni raccolte per ogni dimensione costituiscono il **precampione**, utilizzato per stimare la deviazione standard della popolazione. Ottenuta tale informazione è possibile calcolare la **dimensione campionaria** tale per cui l'errore stimato sulla media della popolazione non sia superiore a 5, con un livello di confidenza del 95%. Per ogni dimensione e per ogni sistema, è stata calcolata quindi la dimensione campionaria come:

$$n = \left(\frac{t_{\frac{\alpha}{2}, 4} \sigma}{E} \right)^2 \quad (1.1)$$

dove:

- $E = 5$ è l'errore massimo commesso nella stima della media della popolazione;
- $\alpha = 0.05$ è il livello di significatività (1-livello di confidenza);
- $t_{\frac{\alpha}{2}, 4} = 2.776$ è l' α -quantile della distribuzione *t-Student* con quattro gradi di libertà.

Una volta determinata la dimensione campionaria opportuna, sono state eventualmente raccolte ulteriori osservazioni, in modo da ottenere un **campione** per ogni dimensione e per ogni sistema.

1.2.1 Sistema 1

A valle della raccolta delle prime 5 osservazioni costituenti il precampione, per ogni dimensione sono stati ottenuti i seguenti valori:

	Size	Time(s)	GFlops	Proc
0	1000	0.013	50.4464	i7-6500U
1	1000	0.016	41.0971	i7-6500U
2	1000	0.016	40.9790	i7-6500U
3	1000	0.017	39.3252	i7-6500U
4	1000	0.018	38.0824	i7-6500U

Tabella 1.1: Precampione dimensione 1000 sistema 1

¹Nella teoria della probabilità, una sequenza di variabili casuali è detta *indipendente e identicamente distribuita* se:

- le variabili hanno tutte la stessa distribuzione di probabilità;
- le variabili sono tutte statisticamente indipendenti.

	Size	Time(s)	GFlops	Proc
0	5000	1.478	56.4206	i7-6500U
1	5000	1.251	66.6291	i7-6500U
2	5000	1.249	66.7740	i7-6500U
3	5000	1.280	65.1569	i7-6500U
4	5000	1.233	68.1825	i7-6500U

Tabella 1.2: Precampione dimensione 5000 sistema 1

	Size	Time(s)	GFlops	Proc
0	10000	10.469	63.7019	i7-6500U
1	10000	9.776	68.2178	i7-6500U
2	10000	10.142	65.7537	i7-6500U
3	10000	9.201	72.4739	i7-6500U
4	10000	10.982	60.7227	i7-6500U

Tabella 1.3: Precampione dimensione 10000 sistema 1

	Size	Time(s)	GFlops	Proc
0	20000	75.549	70.6052	i7-6500U
1	20000	81.923	65.1113	i7-6500U
2	20000	80.515	66.2502	i7-6500U
3	20000	81.325	65.5904	i7-6500U
4	20000	80.218	66.4951	i7-6500U

Tabella 1.4: Precampione dimensione 20000 sistema 1

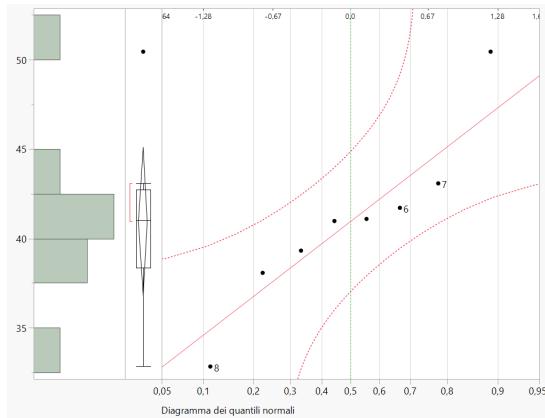
Applicando l'Equazione 1.1 per ogni dimensione campionaria, si ottengono i seguenti risultati:

Size	n	Sample size
1000.0	7.374850	8.0
5000.0	6.849652	7.0
10000.0	6.154958	7.0
20000.0	1.478745	2.0

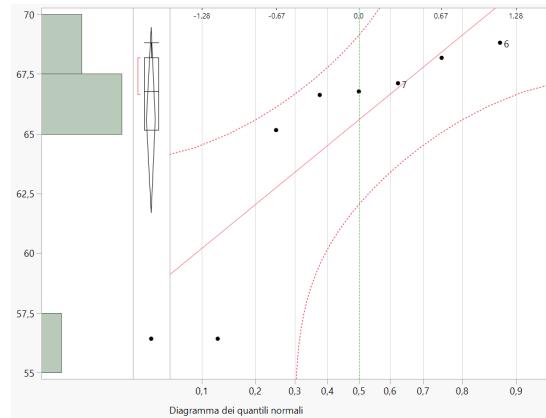
Tabella 1.5: Dimensioni campionarie sistema 1

Per le prime tre dimensioni del benchmark sono state raccolte altre osservazioni, in modo da ottenere campioni della dimensione richiesta.

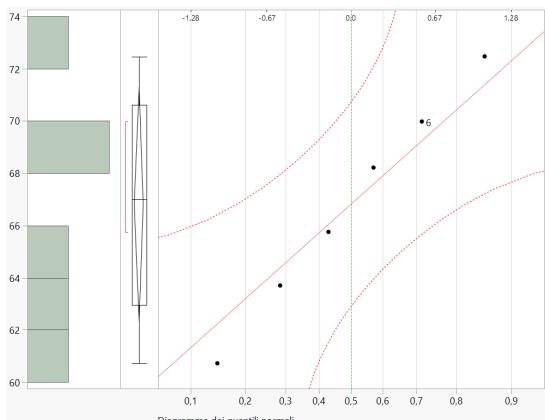
Una volta ottenuti i campioni, per avere un'ulteriore conferma dell'ipotesi di normalità, è stato realizzato per ognuno di essi un **QQ-plot** (plot quantile-quantile), utilizzando *JMP* [1].



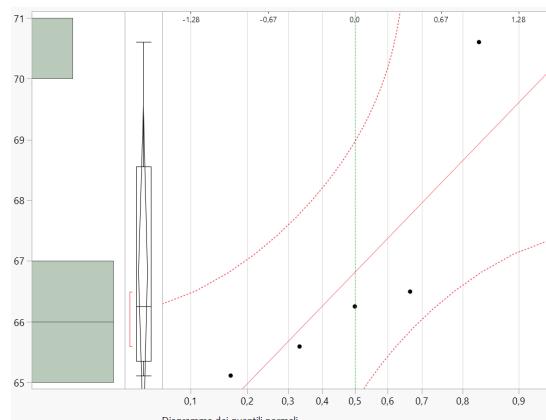
(a) Dimensione 1000



(b) Dimensione 5000



(c) Dimensione 10000



(d) Dimensione 20000

Figura 1.1: QQ-plot campioni sistema 1

Infine, è stata verificata la simmetria, quindi la normalità, della distribuzione dei campioni osservando che media e mediana di ognuno di essi non siano troppo distanti.

Quantili			Quantili		
100.0%	massimo	50,4464	100.0%	massimo	68,8152
99.5%		50,4464	99.5%		68,8152
97.5%		50,4464	97.5%		68,8152
90.0%		50,4464	90.0%		68,8152
75.0%	quartile	42,741725	75.0%	quartile	68,1825
50.0%	mediana	41,03805	50.0%	mediana	66,774
25.0%	quartile	38,3931	25.0%	quartile	65,1569
10.0%		32,8457	10.0%		56,4206
2.5%		32,8457	2.5%		56,4206
0.5%		32,8457	0.5%		56,4206
0.0%	minimo	32,8457	0.0%	minimo	56,4206

Statistiche di riepilogo			Statistiche di riepilogo		
Media	40,947063		Media	65,585371	
Dev std	4,9553165		Dev std	4,2073079	
Errore std della media	1,751969		Errore std della media	1,5902129	
Media superiore al 95%	45,089811		Media superiore al 95%	69,476482	
Media inferiore al 95%	36,804314		Media inferiore al 95%	61,694261	
N	8		N	7	

Quantili			Quantili		
100.0%	massimo	72,4739	100.0%	massimo	70,6052
99.5%		72,4739	99.5%		70,6052
97.5%		72,4739	97.5%		70,6052
90.0%		72,4739	90.0%		70,6052
75.0%	quartile	70,60355	75.0%	quartile	68,55015
50.0%	mediana	66,98575	50.0%	mediana	66,2502
25.0%	quartile	62,9571	25.0%	quartile	65,35085
10.0%		60,7227	10.0%		65,1113
2.5%		60,7227	2.5%		65,1113
0.5%		60,7227	0.5%		65,1113
0.0%	minimo	60,7227	0.0%	minimo	65,1113

Statistiche di riepilogo			Statistiche di riepilogo		
Media	66,80835		Media	66,81044	
Dev std	4,2881805		Dev std	2,1902674	
Errore std della media	1,7506423		Errore std della media	0,9795174	
Media superiore al 95%	71,308519		Media superiore al 95%	69,530016	
Media inferiore al 95%	62,308181		Media inferiore al 95%	64,090864	
N	6		N	5	

(a) Dimensione 1000

(b) Dimensione 5000

(c) Dimensione 10000

(d) Dimensione 20000

Figura 1.2: Statistiche campioni sistema 1

1.2.2 Sistema 2

A valle della raccolta delle prime 5 osservazioni costituenti il precampione, per ogni dimensione sono stati ottenuti i seguenti valori:

	Size	Time(s)	GFlops	Proc
0	1000	0.007	102.5468	i7-7700HQ
1	1000	0.007	100.6210	i7-7700HQ
2	1000	0.007	96.7064	i7-7700HQ
3	1000	0.006	105.5112	i7-7700HQ
4	1000	0.007	93.8769	i7-7700HQ

Tabella 1.6: Precampione dimensione 1000 sistema 2

	Size	Time(s)	GFlops	Proc
0	5000	0.591	140.9704	i7-7700HQ
1	5000	0.587	142.0023	i7-7700HQ
2	5000	0.598	139.3975	i7-7700HQ
3	5000	0.627	133.0295	i7-7700HQ
4	5000	0.587	142.0075	i7-7700HQ

Tabella 1.7: Precampione dimensione 5000 sistema 2

	Time(s)	GFlops	Proc
0	4.473	149.0706	i7-7700HQ
1	4.508	147.9199	i7-7700HQ
2	4.508	156.7314	i7-7700HQ
3	4.585	145.4375	i7-7700HQ
4	4.508	147.9286	i7-7700HQ

Tabella 1.8: Precampione dimensione 10000 sistema 2

	Size	Time(s)	GFlops	Proc
0	20000	37.251	143.1947	i7-7700HQ
1	20000	36.145	147.5765	i7-7700HQ
2	20000	37.699	141.4934	i7-7700HQ
3	20000	36.632	145.6136	i7-7700HQ
4	20000	36.343	146.7725	i7-7700HQ

Tabella 1.9: Precampione dimensione 20000 sistema 2

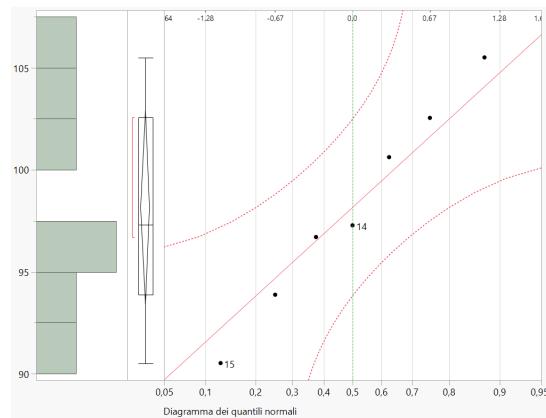
Applicando l'Equazione 1.1 per ogni dimensione campionaria, si ottengono i seguenti risultati:

Size	n	Sample size
1000.0	6.586967	7.0
5000.0	4.360716	5.0
10000.0	5.695905	6.0
20000.0	1.979529	2.0

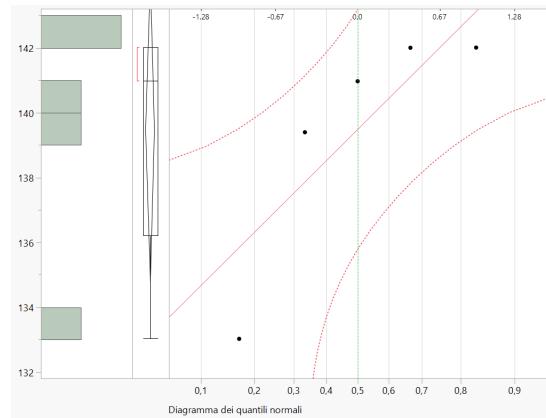
Tabella 1.10: Dimensioni campionarie sistema 2

Per le prime tre dimensioni del benchmark sono state raccolte ulteriori osservazioni, in modo da ottenere campioni della dimensione richiesta.

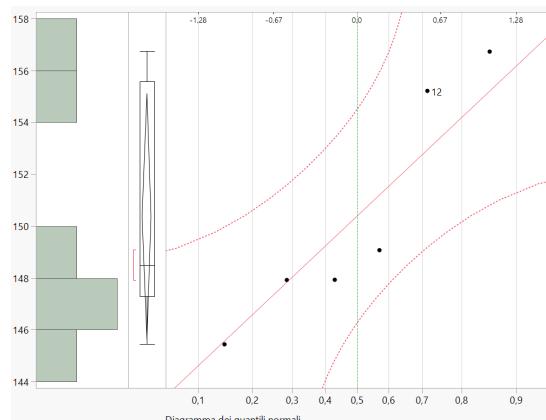
Anche in questo caso, per avere un'ulteriore conferma dell'ipotesi di normalità, è stato realizzato per ogni campione ottenuto un QQ-plot.



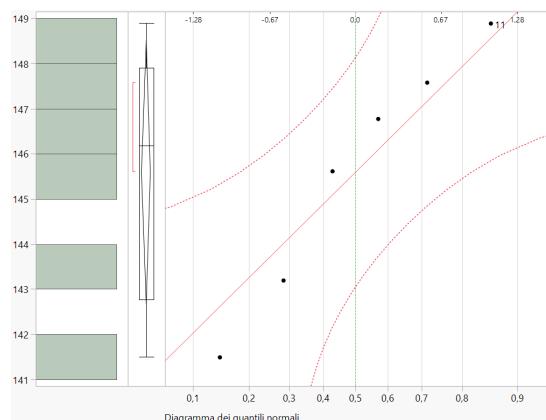
(a) Dimensione 1000



(b) Dimensione 5000



(c) Dimensione 10000



(d) Dimensione 20000

Figura 1.3: QQ-plot campioni sistema 2

Infine, è stata verificata la simmetria, quindi la normalità, della distribuzione dei campioni osservando che media e mediana di ognuno di essi non siano troppo distanti.

Quantili			Quantili		
100.0%	massimo	156,7314	100.0%	massimo	142,0075
99.5%		156,7314	99.5%		142,0075
97.5%		156,7314	97.5%		142,0075
90.0%		156,7314	90.0%		142,0075
75.0%	quartile	155,5953	75.0%	quartile	142,0049
50.0%	mediana	148,4996	50.0%	mediana	140,9704
25.0%	quartile	147,2993	25.0%	quartile	136,2135
10.0%		145,4375	10.0%		133,0295
2.5%		145,4375	2.5%		133,0295
0.5%		145,4375	0.5%		133,0295
0.0%	minimo	145,4375	0.0%	minimo	133,0295

Statistiche di riepilogo			Statistiche di riepilogo		
Media	150,3841		Media	139,48144	
Dev std	4,5152446		Dev std	3,7612263	
Errore std della media	1,8433409		Errore std della media	1,6820715	
Media superiore al 95%	155,12256		Media superiore al 95%	144,15162	
Media inferiore al 95%	145,64564		Media inferiore al 95%	134,81126	
N	6		N	5	

Quantili			Quantili		
100.0%	massimo	156,7314	100.0%	massimo	148,8856
99.5%		156,7314	99.5%		148,8856
97.5%		156,7314	97.5%		148,8856
90.0%		156,7314	90.0%		148,8856
75.0%	quartile	155,5953	75.0%	quartile	147,903775
50.0%	mediana	148,4996	50.0%	mediana	146,19305
25.0%	quartile	147,2993	25.0%	quartile	142,769375
10.0%		145,4375	10.0%		141,4934
2.5%		145,4375	2.5%		141,4934
0.5%		145,4375	0.5%		141,4934
0.0%	minimo	145,4375	0.0%	minimo	141,4934

Statistiche di riepilogo			Statistiche di riepilogo		
Media	150,3841		Media	145,58938	
Dev std	4,5152446		Dev std	2,7830062	
Errore std della media	1,8433409		Errore std della media	1,1361575	
Media superiore al 95%	155,12256		Media superiore al 95%	148,50997	
Media inferiore al 95%	145,64564		Media inferiore al 95%	142,6688	
N	6		N	6	

(a) Dimensione 1000

(b) Dimensione 5000

(c) Dimensione 10000

(d) Dimensione 20000

Figura 1.4: Statistiche campioni sistema 2

1.2.3 Confronto

Ottenuti campioni statisticamente significativi, sono state confrontate le prestazioni dei due sistemi.

Si è scelto per prima cosa di effettuare un **test grafico**, realizzato riportando per ogni dimensione del benchmark l'intervallo di confidenza della media del campione di ciascun sistema. Come è possibile osservare in Figura 1.5, per tutte le dimensioni gli intervalli di confidenza dei due campioni non si sovrappongono. Da questo primo test è possibile

già concludere che le differenze tra i due sistemi sono significative e che il secondo sistema presenta prestazioni superiori rispetto al primo.

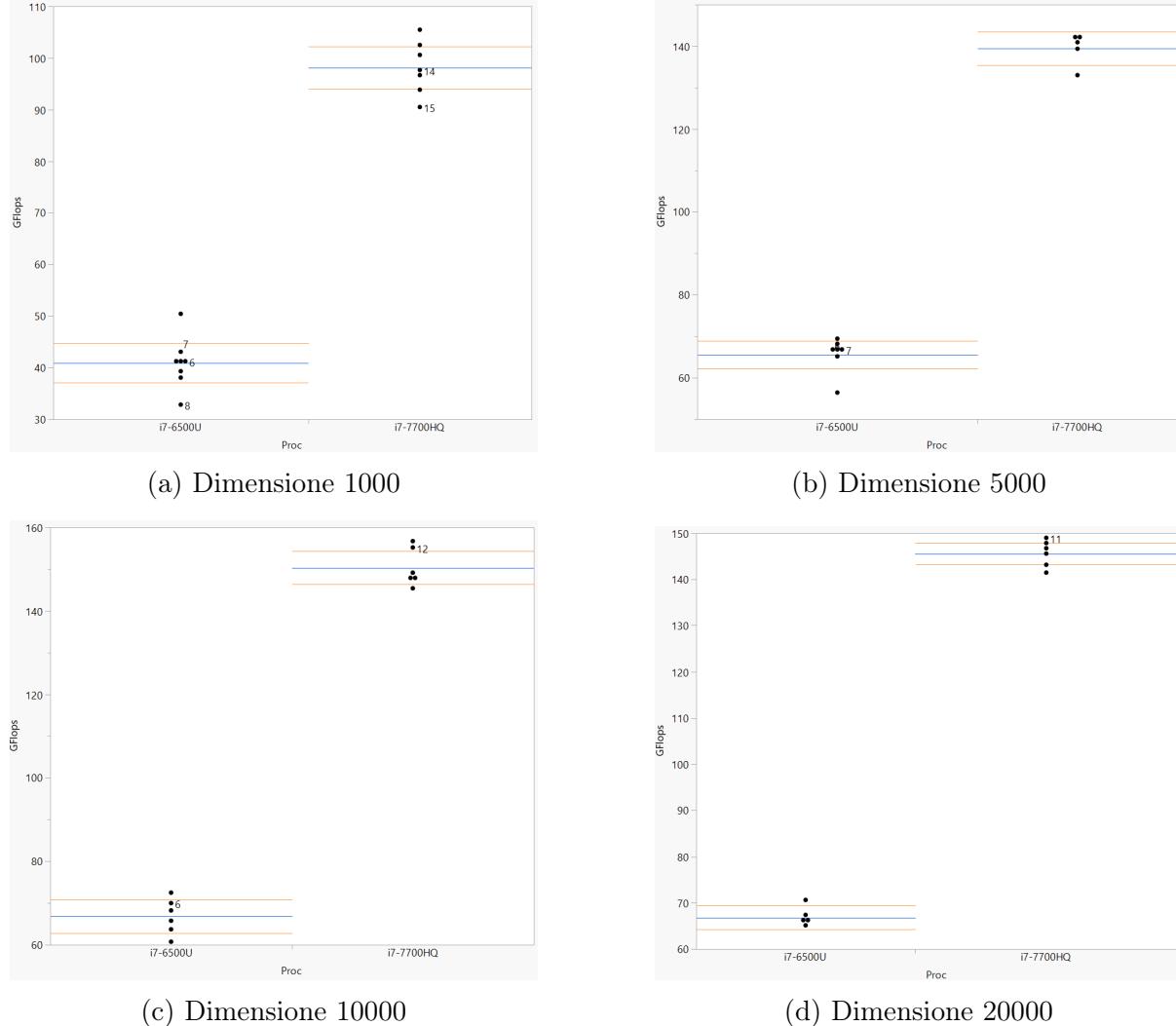


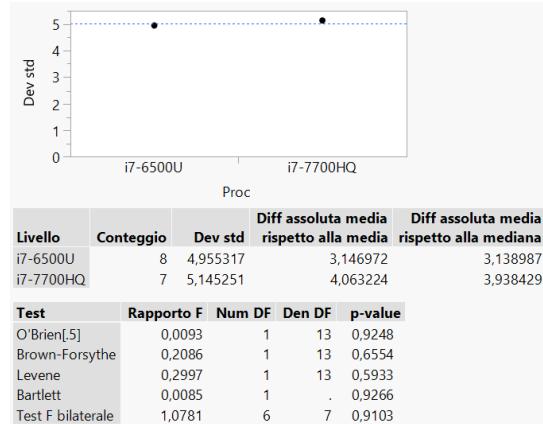
Figura 1.5: Intervalli di confidenza

Per avere un'ulteriore conferma dei risultati ottenuti, è stato effettuato un **test di ipotesi sulla differenza delle medie**, il quale permette di determinare se la differenza tra due campioni è statisticamente significativa, in altre parole, se due campioni provengono o meno dalla stessa popolazione.

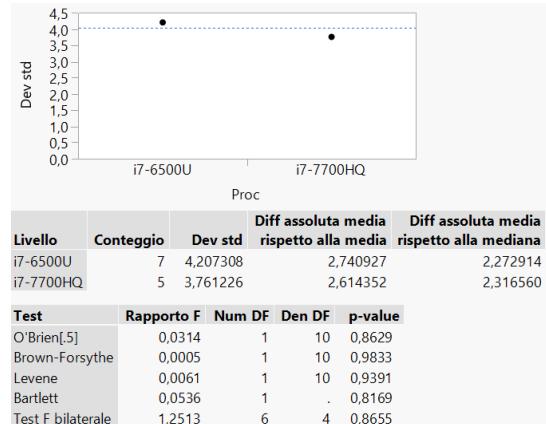
Essendo valide le ipotesi del teorema del limite centrale, è possibile applicare un test parametrico. Dato che le deviazioni standard delle popolazioni non sono note, ma stimate con le deviazioni campionarie, e che le dimensioni campionarie sono sempre inferiori a 30, è necessario applicare un **t-test** (non uno z-test).

Dal momento che i sistemi considerati sono indipendenti, non esiste alcuna corrispondenza tra le osservazioni delle coppie di campioni, i quali in alcuni casi presentano perfino dimensioni diverse. Per questo motivo, per ogni coppia di campioni relativi ad una certa dimensione del benchmark è stato effettuato un **unpaired t-test**.

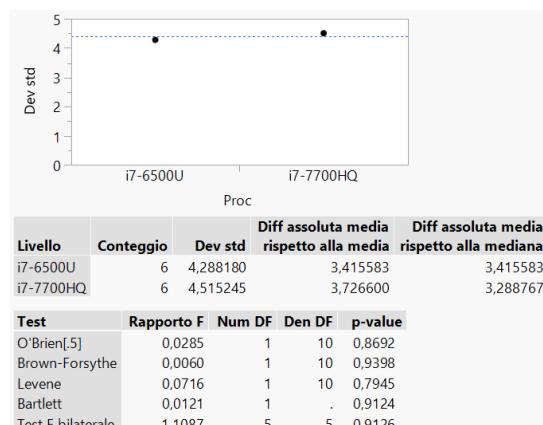
Prima di eseguire il test, è necessario verificare se le varianze dei due campioni sono simili o meno. A seconda di questo si deve eseguire un test differente. Per effettuare questa verifica è stato eseguito per ogni coppia di campioni un **test delle varianze ineguali** su JMP. Come mostrano i risultati in Figura 1.6, l'ipotesi nulla, ovvero che le varianze dei due campioni siano uguali, viene accettata per ogni dimensione del benchmark.



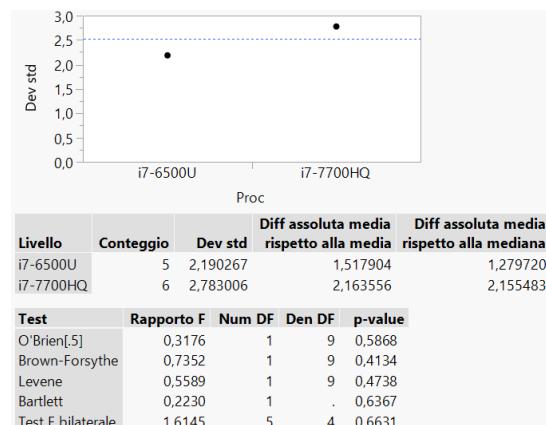
(a) Dimensione 1000



(b) Dimensione 5000



(c) Dimensione 10000



(d) Dimensione 20000

Figura 1.6: Test sulle varianze

Ottenuto questo risultato, per ogni dimensione del benchmark è stato applicato un **t-test aggregato**, in cui la statistica t è ottenuta come:

$$t = \frac{\bar{X}_1 - \bar{X}_2}{s_p \sqrt{\frac{1}{n_1} + \frac{1}{n_2}}}$$

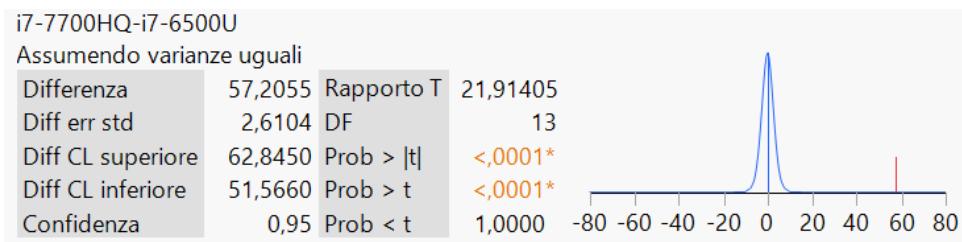
dove:

- \bar{X}_1 e \bar{X}_2 sono le medie campionarie dei due campioni;
- n_1 ed n_2 sono le dimensioni campionare dei due campioni;

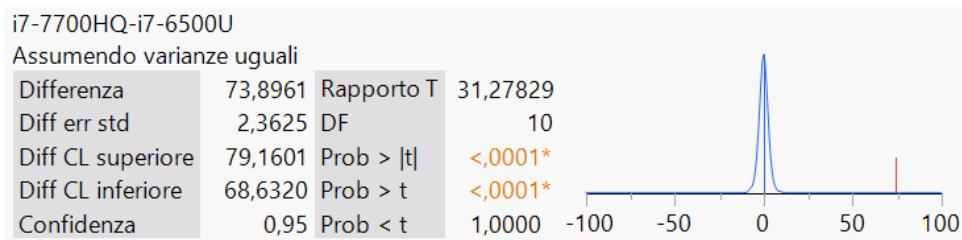
- $s_p = \sqrt{\frac{(n_1-1)s^2 X_1 + (n_2-1)s^2 X_2}{n_1+n_2-2}}$ è uno stimatore della *pooled standard deviation*² dei due campioni;
- s_{X_1} e s_{X_2} sono le deviazioni standard dei due campioni;
- $n_1 + n_2 - 2$ rappresenta il numero di gradi di libertà.

Di fatto il test consiste nel calcolare la statistica t per poi accedere in base ad essa e al numero di gradi di libertà in una tabella contenente i valori del *p-value*. Individuato il valore p opportuno, lo si confronta con il livello di significatività α , fissato in questo caso a 0.05. Se p è inferiore ad α allora viene rigettata l'ipotesi nulla, ovvero che i due campioni sono estratti dalla stessa popolazione.

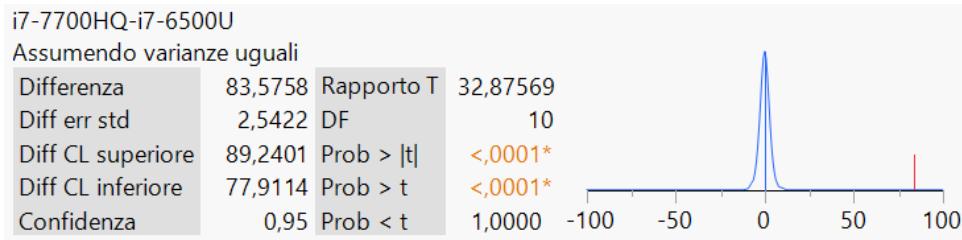
Il t-test aggregato è stato effettuato ancora una volta utilizzando JMP. In Figura 1.7 si riportano i risultati ottenuti. Per tutte le dimensioni del benchmark viene rigettata l'ipotesi nulla, per cui è sempre possibile affermare che le differenze tra i due sistemi sono significative.



(a) Dimensione 1000



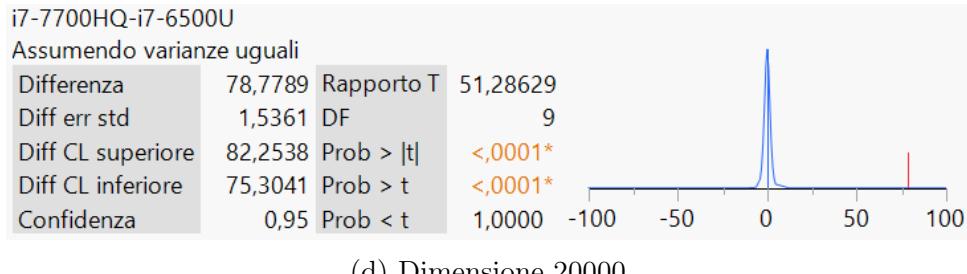
(b) Dimensione 5000



(c) Dimensione 10000

Figura 1.7: T-test

²La *pooled standard deviation* è un metodo per stimare la deviazione standard di popolazioni differenti aventi media diversa ma stessa varianza.



(d) Dimensione 20000

Figura 1.7: T-test (cont.)

In conclusione, è possibile osservare che i risultati attesi inizialmente sono stati confermati: il secondo sistema, dotato del processore *Intel Core i7-7700HQ* presenta prestazioni superiori rispetto al primo, dotato di processore *Intel Core i7-6500U*, in quanto le differenze riscontrate non sono dovute al caso.

Capitolo 2

Workload Characterization

2.1 Traccia

Realizzare la *workload characterization* di un sistema Linux applicando le tecniche di *Principal Component Analysis* (PCA) e *clustering*. Inoltre è richiesto di trovare il miglior trade-off tra devianza persa e riduzione del dataset.

2.2 Workload characterization

La **workload characterization** è il processo di creazione di un **workload sintetico**, rappresentativo di un workload reale. In altre parole, si vuole estrarre un modello statistico del workload reale che ne rappresenti la variabilità.

Il workload da caratterizzare è costituito da 24 componenti (colonne) e 3000 osservazioni (righe), per un totale di 72000 valori.

2.2.1 Preprocessing

Prima di procedere con la caratterizzazione è necessario eseguire alcuni step di preprocessing.

2.2.1.1 Componenti costanti

Per prima cosa, si verifica la presenza di componenti costanti all'interno del workload.

Individuazione delle componenti costanti

```
df_const = df_rough[df_rough.columns[df_rough.nunique() == 1]]  
df_const.head()
```

In Tabella 2.1 è possibile osservare un estratto del dataset che mostra le colonne costanti.

	Active	AnonPages	avgLatency	Errors
0	0	8159224	0	2
1	0	8159224	0	2
2	0	8159224	0	2
3	0	8159224	0	2
4	0	8159224	0	2

Tabella 2.1: Esempio di colonne costanti

Dalla verifica si evince che 4 delle 24 componenti del workload hanno valori costante e sono: *Active*, *AnonPages*, *avgLatency*, *Errors*. Dal momento che queste componenti non contribuiscono alla varianza del workload è bene eliminarle.

Rimozione delle componenti costanti

```
df_temp = df_rough[df_rough.columns[df_rough.nunique() != 1]]
```

2.2.1.2 Componenti perfettamente correlate

Un altro aspetto da considerare è l'esistenza di componenti perfettamente correlate. Anche in questo caso, componenti di questo tipo non contribuiscono alla varianza del workload.

Per individuare tali componenti è possibile calcolare la **matrice di correlazione** C delle colonne del dataset. L'elemento c_{ij} della matrice rappresenta la correlazione tra le componenti i e j . Se c_{ij} è pari ad 1 significa che le componenti i e j sono perfettamente correlate, ovvero linearmente dipendenti.

Calcolo della matrice di correlazione

```
df_corr = df_temp.corr()
```

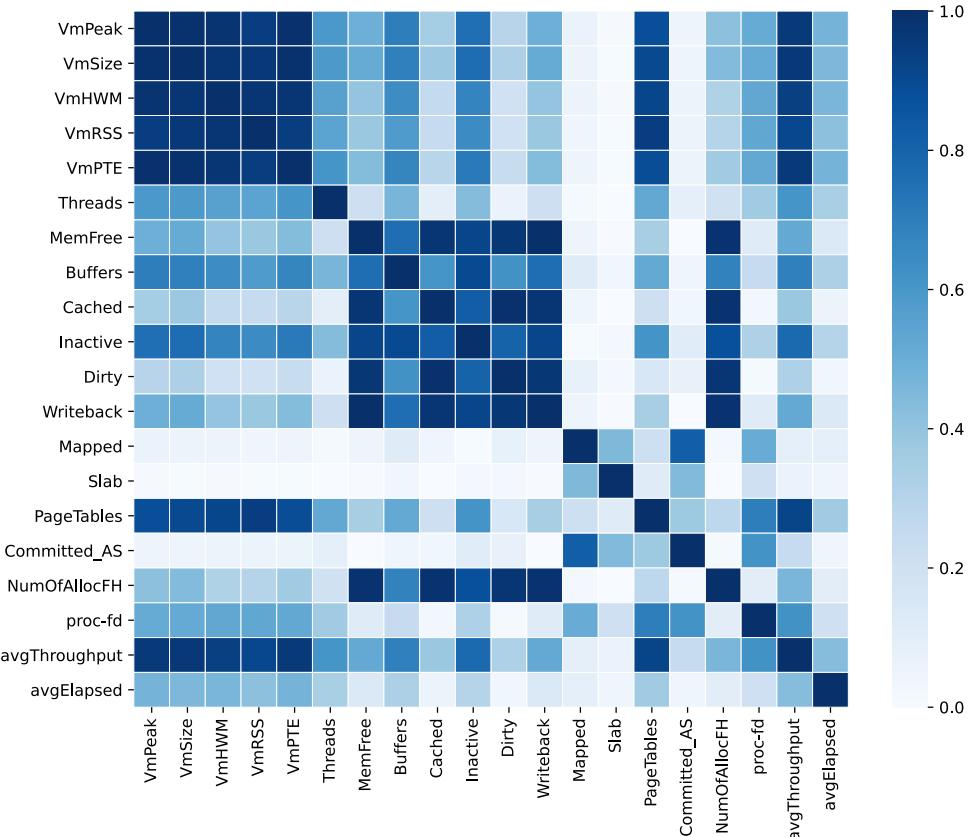


Figura 2.1: Matrice di correlazione

Dal momento che ciascuna componente è perfettamente correlata a se stessa, si verifica la presenza di 1 al di fuori della diagonale principale della matrice di correlazione.

Individuazione delle componenti perfettamente correlate

```
mask = np.ones(df_corr.shape, dtype='bool')
mask[np.triu_indices(len(df_corr))] = False

corr_comp = (df_corr == 1 & mask).values.nonzero()
corr_comp = [val[0] for val in corr_comp]

df_temp.columns[corr_comp]
```

Da questa verifica si evince che due componenti, *Writeback* e *MemFree*, sono linearmente dipendenti. È sufficiente considerare solo una delle due, ad esempio *MemFree*, mentre l'altra può essere scartata dal dataset.

Individuazione delle componenti perfettamente correlate

```
df = df_temp.drop(df_temp.columns[corr_comp[0]], axis=1)
```

Al termine della fase di preprocessing, il workload è stato ridotto a 19 componenti e 3000 osservazioni, per un totale di 57000 valori.

2.2.1.3 Normalizzazione

L'ultimo step da eseguire consiste nel normalizzare i valori del dataset. Tale operazione è necessaria dato che il valore assoluto degli attributi influenza il risultato della PCA. Nello specifico, si è scelto di normalizzare applicando la definizione di **z-score**:

$$z = \frac{x - \bar{x}}{\sigma_x}$$

In questo modo, le colonne del dataset avranno media nulla e varianza unitaria.

Normalizzazione dei dati

```
x = df.values
x_scaled = StandardScaler().fit_transform(x)
```

2.2.2 Principal Component Analysis (PCA)

La prima tecnica usata per eseguire la workload characterization è la **principal component analysis** (PCA), la quale consiste nel calcolare le componenti principali del dataset per poi usarle per effettuare un cambiamento di base.

Le **componenti principali** di un insieme di punti sono dei vettori direzione ortogonali che rappresentano una base ortonormale in cui le differenti dimensioni dei dati sono linearmente incorrelate. Inoltre, le componenti principali sono ordinate in base alla varianza spiegata dei dati: la prima componente è la direzione che massimizza la varianza dei dati, la seconda componente è la seconda direziona che massimizza la varianza dei dati, e così via.

La PCA è utilizzata per proiettare ogni punto del dataset in uno spazio costituito solo da alcune delle prime componenti principali, in modo da ottenere un dataset caratterizzato da una minore dimensionalità, preservando allo stesso tempo quanta più varianza possibile. Tale processo è noto come **dimensionality reduction**.

2.2.2.1 Esecuzione PCA

Per calcolare le componenti principali del workload è stata utilizzata la classe PCA, fornita dalla libreria *scikit-learn* [2]. A tal fine, è sufficiente richiamare il metodo `fit`, fornendo in ingresso i dati normalizzati.

Calcolo delle componenti principali

```
pca = PCA()
princ_comp = pca.fit(df_scaled.values)
```

Una volta calcolate le componenti, è possibile proiettare i dati nel nuovo spazio di stato, richiamando il metodo `transform`.

Proiezione dei dati

```
d = pca.transform(df_scaled.values)
```

2.2.2.2 Varianza spiegata

Una volta eseguita la PCA è possibile osservare la varianza spiegata da ognuna delle componenti principali. Tali informazioni sono riportate in Tabella 2.2. In particolare, la prima colonna della tabella rappresenta il valore assoluto della varianza spiegata da ogni componente, la seconda rappresenta il rapporto tra varianza spiegata da una componente e varianza totale del dataset e infine la terza rappresenta la varianza cumulativa spiegata dalle componenti rispetto a quella totale.

	explained variance	explained variance ratio	cumulative explained variance ratio
0	9.861920	0.518875	0.518875
1	3.667308	0.192952	0.711827
2	2.459855	0.129423	0.841250
3	0.781238	0.041104	0.882354
4	0.706996	0.037198	0.919552
5	0.625316	0.032900	0.952453
6	0.339288	0.017851	0.970304
7	0.265791	0.013984	0.984288
8	0.161845	0.008515	0.992804
9	0.090384	0.004755	0.997559
10	1.777932e-02	9.354420e-04	0.998494
11	1.139004e-02	5.992757e-04	0.999094
12	7.844429e-03	4.127271e-04	0.999506
13	5.749318e-03	3.024948e-04	0.999809
14	1.813531e-03	9.541720e-05	0.999904
15	1.138868e-03	5.992046e-05	0.999964
16	6.755696e-04	3.554444e-05	1.000000
17	2.875732e-06	1.513039e-07	1.000000
18	1.428357e-07	7.515164e-09	1.000000

Tabella 2.2: Varianza spiegata dalle componenti principali

Le stesse informazioni sono riportate in Figura 2.2.

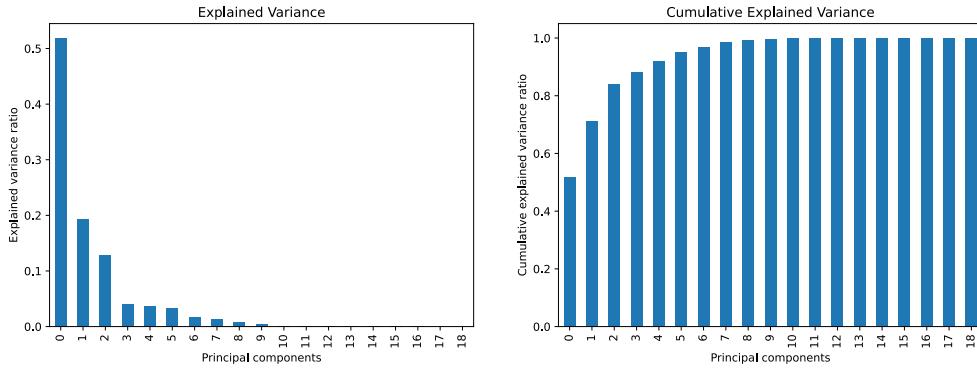


Figura 2.2: Grafici della varianza delle componenti principali

2.2.3 Clustering

Dopo aver ridotto la dimensionalità del workload (numero di colonne) si vuole ridurre anche la sua dimensione (numero di righe). A tal fine è possibile utilizzare la tecnica del **clustering**, la quale consiste nel raggruppare un insieme di elementi in modo che gli elementi nello stesso gruppo sono più simili tra loro rispetto agli elementi di altri gruppi. Una volta trovati i gruppi di elementi, detti anche **cluster**, è possibile costruire un nuovo workload (sintetico) costituito da un elemento di ogni cluster.

Tra le diverse tecniche di clustering si è scelto di utilizzare un *approccio agglomerativo* detto **metodo di Ward**, il quale è un metodo di *clustering gerarchico* che minimizza la *devianza intra-cluster*. Minimizzando tale quantità si minimizza anche la **varianza intra-cluster**, che rappresenta proprio la porzione di varianza persa applicando il clustering rispetto alla varianza totale del dataset.

Secondo il metodo di Ward, inizialmente ogni elemento del dataset rappresenta un cluster diverso. Vengono poi costruiti nuovi cluster in maniera iterativa, unendo ad ogni iterazione i due cluster più vicini. Per parlare di vicinanza tra cluster è necessario definire una metrica di distanza tra cluster. La distanza tra due cluster P e Q è misurata attraverso una metrica detta *distanza di Ward*, definita come:

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

dove $|Q|$ è la cardinalità del cluster Q e \bar{x}_Q è il suo **centroide**, definito come:

$$\bar{x}_Q = \frac{1}{|Q|} \sum_{i=1}^{|Q|} x_i$$

con x_i i-esimo elemento del cluster Q . Lo stesso vale chiaramente per il cluster P .

2.2.3.1 Esecuzione clustering

Per prima cosa è necessario definire i dati da clusterizzare. Come detto precedentemente, è possibile considerare solo alcune delle prime componenti principali. Ad esempio,

si considerano le prime 3 componenti principali, che spiegano l'84% della varianza del dataset.

Esecuzione del clustering

```
n_comp = 3  
df_clustering = df_princ_comp.iloc[:, 0:n_comp]
```

Eseguire il clustering dopo la PCA porta due vantaggi fondamentali:

- Il clustering è applicato ad un dataset avente un numero minore di componenti. Eseguendo la PCA, il dataset viene infatti ridotto a 3 componenti e 3000 osservazioni, per un totale di 9000 valori.
- Si prevengono errori nel clustering, essendo le componenti principali incorrelate.

Dopo aver fissato il numero di componenti principali è necessario stabilire il numero di cluster¹. Ad esempio, si suppone di voler dividere il dataset in 10 gruppi.

Per eseguire il clustering è stata utilizzata la classe `AgglomerativeClustering`, fornita da *scikit-learn*. Di default la metrica di distanza utilizzata dall'algoritmo è la distanza di Ward. Per applicare l'algoritmo, è sufficiente richiamare il metodo `fit`, fornendo in ingresso i dati ottenuti dalla PCA.

Esecuzione del clustering

```
n_clusters = 10  
clustering = AgglomerativeClustering(n_clusters=n_clusters)  
clustering = clustering.fit(df_clustering.values)
```

Una volta eseguito il clustering, si ha un'etichetta per ogni osservazione che ne indica il cluster di appartenenza. Sebbene il clustering sia eseguito nello spazio definito dalle prime 3 componenti principali, le etichette dei cluster sono associate anche alle rispettive osservazioni nello spazio originale.

2.2.3.2 Workload sintetico

Evidentemente, il metodo di Ward è l'algoritmo di clustering più appropriato per eseguire la caratterizzazione di un workload: da un lato ne riduce la dimensione, dall'altro ne preserva quanta più varianza possibile.

In questo caso, partendo da un workload di 3000 (72000 valori) osservazioni, è stato ottenuto un workload sintetico costituito da sole 10 osservazioni (240 valori), ognuna delle quali è un campione estratto dal relativo cluster.

¹La scelta del numero di componenti e del numero di cluster è molto critica e deve essere effettuata tenendo conto della percentuale di varianza del dataset che si vuole preservare. Tale questione viene approfondita nelle sezioni successive, in quella corrente ci si limita a fissare dei valori per mostrare il processo completo della caratterizzazione di un workload.

Creazione del workload sintetico

```
df_synth = df_clustered.groupby('cluster').apply(lambda x: x.sample())
df_synth = df_synth.reset_index(level=0, col_level=2, drop=True)
```

Se si considerano i dati ottenuti nello spazio trasformato ridotto, la riduzione è ancora più significativa, in quanto si ottiene un dataset costituito da 10 punti in uno spazio a 3 dimensioni, per un totale di 30 valori.

In conclusione, si riporta la Tabella 2.3 riassuntiva che evidenza come si riducono le dimensioni del dataset durante il processo della workload characterization.

	osservazioni	componenti	valori
workload reale	3000	24	72000
preprocessing	3000	19	57000
PCA	3000	3	9000
clustering	10	3	30
workload sintetico	10	24	240

Tabella 2.3: Riduzione dimensioni workload

2.2.4 Analisi della devianza

L'obiettivo della caratterizzazione è quello di definire un workload sintetico che sia rappresentativo di quello reale, in termini di variabilità dei dati. Per questo motivo, è fondamentale osservare come entrambe le tecniche usate nel processo di caratterizzazione vanno ad impattare sulla varianza del workload. Più precisamente, piuttosto che considerare la varianza si preferisce studiare la devianza, per motivi legati alla scelta della tecnica di clustering.

2.2.4.1 Devianza totale

Per prima cosa, si calcola la devianza totale del workload reale, ottenuta come:

$$d_{tot} = \sum_{i=1}^N \|x_i - \bar{x}\|^2 = 57000.00000000045 \quad (2.1)$$

dove:

- N è la dimensione del dataset;
- x_i è l'i-esima osservazione del dataset;
- \bar{x} è la media delle osservazioni del dataset.

2.2.4.2 Devianza PCA

Successivamente, si calcola la devianza preservata dalla PCA, ovvero la devianza dei dati proiettati nello spazio definito dalle prime 3 componenti principali:

$$d_{PCA} = \sum_{i=1}^N \|x_{PCA_i} - \bar{x}_{PCA}\|^2 = 47951.256746660554 \quad (2.2)$$

dove:

- N è la dimensione del dataset;
- x_{PCA} è l'i-esima osservazione del dataset proiettato nello spazio trasformato ridotto;
- \bar{x}_{PCA} è la media delle osservazioni del dataset proiettato nello spazio trasformato ridotto.

Calcolando il rapporto tra d_{PCA} e d_{tot} si ottiene la porzione di devianza preservata dalla PCA rispetto al totale:

$$d_{PCA_{ratio}} = \frac{d_{PCA}}{d_{tot}} = 0.8412501183624592 \quad (2.3)$$

Evidentemente, il valore coincide con il risultato riportato in Tabella 2.2 in corrispondenza di 3 componenti principali.

2.2.4.3 Devianza Clustering

Infine, si deve considerare come il clustering influisce sulla devianza dei dati. Per prima cosa, si osserva che la devianza totale dei dati da clusterizzare, ovvero la devianza dopo aver applicato la PCA, può essere scritta come:

$$d_{PCA} = d_{intra} + d_{inter}$$

dove d_{intra} e d_{inter} sono chiamate rispettivamente **devianza intra-cluster** e **devianza inter-cluster**. Dal momento che per creare il workload sintetico si considera un campione casuale per ogni cluster, d_{intra} rappresenta la devianza persa a causa del clustering, mentre d_{inter} rappresenta la devianza preservata.

La devianza intra-cluster è definita come la somma delle devianze di ogni cluster:

$$d_{intra} = \sum_{k=1}^K \sum_{i=1}^{n_k} \|x_i - \bar{x}_k\|^2 = 993.5895196217942$$

dove:

- K è il numero di cluster;
- n_k è il numero di elementi del cluster k ;
- x_i è l'i-esimo elemento del cluster k

- \bar{x}_k è il centroide del cluster k

Calcolando il rapporto tra d_{intra} e d_{PCA} si ottiene la porzione di **devianza persa** a causa del clustering rispetto alla varianza preservata dalla PCA:

$$d_{intra_ratio} = \frac{d_{intra}}{d_{PCA}} = 0.020720823332560315 \quad (2.4)$$

La devianza inter-cluster è definita come:

$$d_{inter} = \sum_{k=1}^K n_k \|\bar{x}_k - \bar{x}\|^2 = 46957.66722703876$$

dove:

- K è il numero di cluster;
- n_k è il numero di elementi del cluster k ;
- \bar{x}_k è il centroide del cluster k
- $\bar{x} = \frac{1}{n} \sum_{i=1}^n x_i$ è la media di tutti gli elementi del dataset proiettato nello spazio trasformato ridotto.

Calcolando il rapporto tra d_{inter} e d_{PCA} si ottiene la porzione di **devianza preservata** dal clustering rispetto alla varianza preservata dalla PCA:

$$d_{intra_ratio} = \frac{d_{intra}}{d_{PCA}} = 0.9792791766674396 \quad (2.5)$$

Banalmente, si può verificare che la somma tra devianza inter ed intra-cluster è uguale alla devianza preservata dalla PCA.

2.2.4.4 Devianza persa

Infine, è possibile calcolare la devianza totale persa a causa di PCA e clustering seguendo diversi approcci:

1. La porzione di devianza persa può essere vista come la somma tra la devianza persa a causa della PCA e quella persa a causa del clustering (devianza intra-cluster), che è relativa alla devianza preservata dalla PCA:

$$d_{lost_ratio} = (1 - d_{PCA_ratio}) + d_{intra_ratio} \cdot d_{PCA_ratio} = 0.17618127671862474$$

2. La porzione di devianza persa può essere vista come la devianza totale meno la devianza preservata, che a sua volta è data dal prodotto tra la devianza preservata dalla PCA e quella preservata dal clustering (devianza inter-cluster):

$$d_{lost_ratio} = 1 - d_{PCA_ratio} * d_{inter_ratio} = 0.17618127671862482$$

In conclusione, eseguendo la workload characterization considerando 3 componenti principali e fissando il numero di cluster pari a 10, si perde circa il 17% della varianza del workload reale.

2.3 Trade-off

Se da un lato la riduzione della dimensione del workload dipende dal numero di clusters, dall'altro la perdita di devianza causata dalla PCA e dal clustering dipende non solo dal numero di clusters ma anche dal numero di componenti principali. Per tale ragione, è necessario trovare il numero di componenti principali e clusters che assicura il miglior trade-off tra devianza persa e riduzione della dimensione del workload.

2.3.1 Devianza persa

Per valutare la quantità di devianza persa al variare di questi parametri è sufficiente eseguire gli step riportati nella Paragrafo 2.2 per diversi valori di numero di componenti principali e clusters.

In particolare, si è scelto di far variare il numero di componenti principali da 3 (84% di varianza spiegata) a 6 (95% di varianza spiegata) ed il numero di cluster da 1 a 20.

Calcolo devianza persa al variare del numero di componenti principali e clusters

```

nmin_comp = 3
nmax_comp = 6
n_comp = nmax_comp-nmin_comp+1

nmax_cluster = 20

array_dev_lost = np.zeros(shape=(nmax_cluster,n_comp))

for i in range(nmin_comp,nmax_comp+1):
    df_clustering = df_princ_comp.iloc[:,0:i]

    for j in range(1,nmax_cluster+1):
        # clustering
        clustering = AgglomerativeClustering(n_clusters=j)
        clustering = clustering.fit(df_clustering.values)
        df_clustering.loc[:, 'cluster'] = clustering.labels_

        # deviance PCA
        df_pca = df_princ_comp.iloc[:,0:i]
        dev_pca = df_pca.sub(df_pca.mean()).pow(2).sum().sum()

        # deviance clustering
        n_elem = df_clustering.groupby('cluster').count().values[:,0]
        centroids = df_clustering.groupby('cluster').mean()

        # intra-cluster deviance
        arr = np.zeros(shape=(j,1))
        for k in range(j):
            elem_k = df_clustering.loc[df_clustering['cluster'] == k]
            .drop(columns='cluster')
            arr[k] = elem_k.sub(centroids.loc[k,:]).pow(2).sum().sum()
        dev_intra = arr.sum()

        # inter-cluster deviance
        dev_inter = centroids.sub(df_clustering.mean()).pow(2).mul(n_elem, axis=0)
        .sum().sum()
        dev_inter

        # deviance lost
        dev_pca_ratio = dev_pca/dev_tot
        dev_inter_ratio = dev_inter/dev_pca
        dev_lost = 1 - dev_inter_ratio * dev_pca_ratio

        array_dev_lost[j-1][i-n_comp+1] = dev_lost
    
```

A questo punto è possibile osservare come varia la devianza persa in funzione del numero di cluster, per diversi valori di componenti principali come mostrato in Figura 2.3.

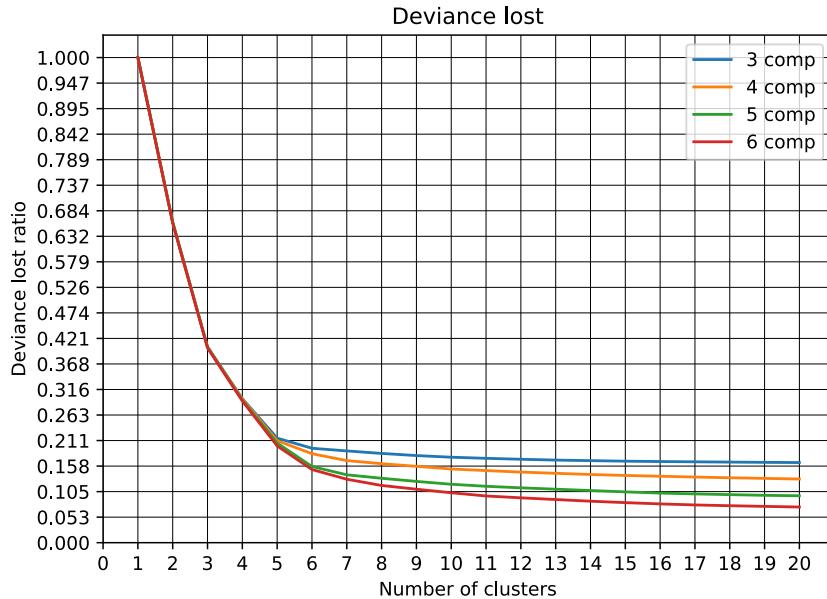


Figura 2.3: Varianza persa in funzione del numero di clusters e componenti principali

Si nota quindi che la perdita di devianza presenta un andamento monotono decrescente con l'aumentare del numero di clusters, indipendentemente dal numero di componenti principali. Tale risultato è ragionevole dal momento che aumentando il numero di clusters si vanno a considerare più osservazioni e di conseguenza si preserva una maggiore quantità di devianza. Tuttavia, la devianza non decresce linearmente. Si può osservare infatti come la curva si stabilizzi una volta raggiunto un numero di cluster pari a 6.

Inoltre, per un numero di clusters strettamente minore di 6, la quantità di devianza persa non dipende molto dal numero di componenti principali considerate. Al contrario, se il numero di cluster è maggiore o uguale a 6, la perdita di devianza diventa minore all'aumentare del numero di componenti principali. Ad esempio, con 6 clusters, scegliere 3 o 6 componenti principali comporta una differenza del 5% in termini di devianza persa.

2.3.2 Dimensione workload sintetico

La riduzione della dimensione del workload può essere misurata considerando il rapporto tra il numero di valori (numero di colonne per numero di righe) del workload sintetico ed il numero di valori del workload reale. Tale rapporto assume valore massimo pari ad 1 quando il workload sintetico ha la stessa dimensione del workload reale, e valore minimo quanto il workload sintetico ha dimensione unitaria, ovvero nel caso in cui si consideri un unico cluster.

Calcolo della riduzione della dimensione del workload

```
dim_synt = np.arange(1,nmax_cluster+1)*df_rough.shape[1]
dim_real = df_rough.size
array_dim = dim_synt/dim_real
```

Come già detto, la riduzione dipende in particolar modo dal numero di clusters, dal momento che la dimensione del workload sintetico cresce linearmente con il numero di clusters.

2.3.3 Devianza persa vs. Dimensione del workload sintetico

Idealmente, si vorrebbe che sia la devianza persa che la dimensione del workload sintetico siano quanto più bassi possibili. Tuttavia, la devianza decresce con l'aumentare del numero di clusters (e di componenti principali) mentre la dimensione del workload sintetico aumenta con l'aumentare degli stessi parametri.

Per questo motivo, è possibile considerare il prodotto tra devianza persa e dimensione del workload sintetico ed osservare quali sono i valori di cluster e componenti principali che minimizzano tale quantità. In Figura 2.4 è riportato l'andamento della funzione prodotto.

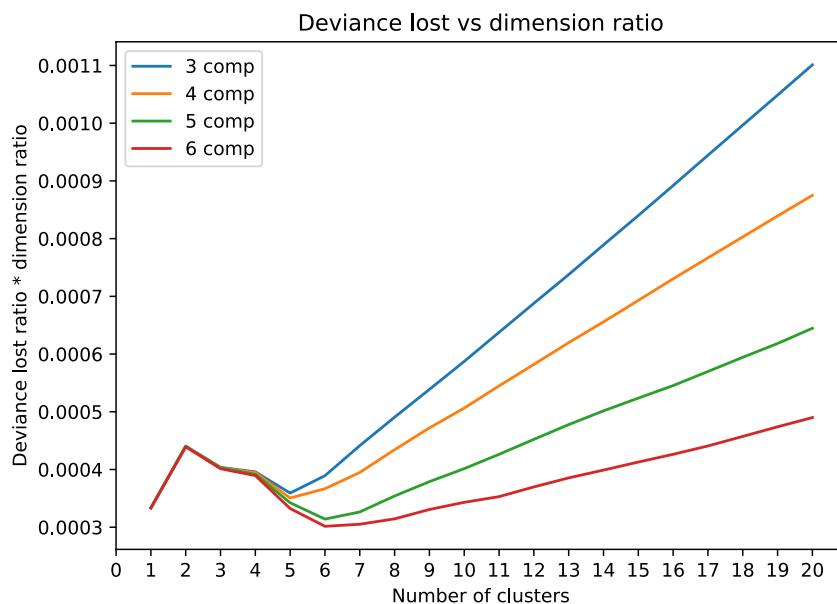


Figura 2.4: Varianza persa in funzione del numero di clusters e componenti principali

Osservando il grafico è possibile concludere che:

- Se si vuole il minimo numero di componenti principali che preservi un buon quantitativo di devianza, si dovrebbero scegliere 3 componenti e 5 clusters;
- Se ci si può permettere di utilizzare un maggior numero di componenti principali, si dovrebbero scegliere 5 componenti principali e 6 clusters.

In ogni caso, è importante evidenziare che il numero di componenti principali e di cluster scelti dipende in maniera significativa dalle risorse a disposizione per i processi di caratterizzazione del workload e di performance analysis.

Capitolo 3

Web Server Performance Analysis

3.1 Traccia

Analizzare le prestazioni di un web server, eseguendo in particolare:

- *Workload characterization*: caratterizzare il workload del sistema, in termini di parametri di basso e alto livello. Verificare inoltre che il workload sintetico sia rappresentativo del workload reale.
- *Capacity test*: valutare throughput e tempo di risposta al variare del carico.
- *Design of Experiment*: studiare l'influenza di diversi fattori sul tempo di risposta.

3.2 Introduzione

La prima fase del processo di performance analysis consiste nel definire il sistema da analizzare.

Il **System Under Test** (SUT) è un server Linux eseguito su una macchina virtuale Oracle VM Virtual Box, le cui caratteristiche sono riportate in Tabella 3.1.

	Server
CPU	i7-7700HQ
Core	1
RAM	2 GB
SO	Ubuntu 20.4.1 LTS 64-bit

Tabella 3.1: Specifiche server

In particolare, si vuole studiare un componente specifico del sistema, ovvero un web server Apache 2.4 che rappresenta il **Component Under Study** (CUS). In Tabella 3.2 si riportano le caratteristiche del server.

	Web server
StartServers	2
MinSpareThreads	25
MaxSpareThreads	75
ThreadLimit	64
ThreadsPerChild	25
MaxRequest Worker	50
MaxConnectionsPerChild	0

Tabella 3.2: Specifiche web server

Il web server espone una serie di risorse ad un generico client accessibili tramite richieste http. Nello specifico si è scelto di utilizzare tre tipi di risorse, classificate sulla base della loro dimensione. Per ogni tipologia sono presenti cinque file distinti. In Tabella 3.3 si riportano le caratteristiche di ogni risorsa.

Tipo	Risorsa	Dimensione
Small	WikipediaHome.html	71.2 KB
	WikipediaPag1.html	158.3 KB
	WikipediaPag2.html	131.4 KB
	WikipediaPag3.html	134.8 KB
	WikipediaPag4.html	202.2 KB
Medium	RepubblicaHome.html	2.0 MB
	RepubblicaPag1.html	1.5 MB
	RepubblicaPag2.html	1.5 MB
	RepubblicaPag3.html	1.5 MB
	RepubblicaPag4.html	1.5 MB
Large	YouTubeHome.html	4.2 MB
	YouTubePag1.html	5.1 MB
	YouTubePag2.html	4.0 MB
	YouTubePag3.html	4.1 MB
	YouTubePag4.html	4.1 MB

Tabella 3.3: Specifiche risorse

Per analizzare il web server, le richieste sono effettuate tramite il tool Apache JMeter, eseguito su un client Windows avente le caratteristiche riportate in Tabella 3.4.

	Client
CPU	i7-7700HQ
Core	4
RAM	16 GB
SO	Windows 10 Home 64-bit

Tabella 3.4: Specifiche client

3.3 Workload characterization

In questa sezione viene descritto il processo di caratterizzazione del workload, il cui scopo, come ampiamente spiegato nel Paragrafo 2.2, è generare un workload sintetico a partire da un workload reale. A tal fine, sono state effettuate le seguenti operazioni:

- Generazione del workload reale
 - Raccolta dei parametri di alto livello (HL)
 - Raccolta dei parametri di basso livello (LL)
- Caratterizzazione del workload reale
 - Caratterizzazione dei parametri di alto livello (HL_c)
 - Caratterizzazione dei parametri di basso livello (LL_c)
- Applicazione del workload sintetico di alto livello
 - Raccolta dei parametri di basso livello (LL')
- Caratterizzazione del workload sintetico¹
 - Caratterizzazione dei parametri di basso livello (LL'_c)
- Validazione del workload sintetico di alto livello
 - Test di ipotesi sulla differenza delle medie dei parametri di basso livello caratterizzati (LL_c ed LL'_c)

3.3.1 Generazione workload reale

Al fine di simulare l'imposizione di un carico reale al sistema si è cercato di diversificare quanto più possibile le richieste effettuate dal client.

Si è deciso quindi di definire quattro **Thread Group**, ognuno dei quali effettua casualmente, attraverso un **Random Controller**, una delle tre tipologie di richieste. Più precisamente, ad ogni attivazione un thread sceglie casualmente una tipologia di richieste e, attraverso un **Simple Controller**, richiede tutte le risorse della data tipologia.

I quattro Thread Group si differenziano sulla base del numero di utenti e del numero di richieste al minuto effettuate dal singolo utente, imposto tramite un **Constant Throughput Timer**. Inoltre, in ogni Thread Group viene avviato un nuovo thread ogni 0.5 secondi. In Tabella 3.5 sono riassunte le configurazioni di ciascun Thread Group.

¹Per caratterizzazione del workload sintetico si intende la caratterizzazione dei parametri di basso livello LL' raccolti durante l'applicazione del workload sintetico di alto livello HL_c al web server.

	Throughput	Utenti	Ramp-up period
TG Slow High	10	20	10
TG Slow Low	10	10	5
TG Fast High	20	20	10
TG Fast Low	20	10	5

Tabella 3.5: Configurazione Apache JMeter

Definita la configurazione di JMeter, è stato eseguito un test della durata di cinque minuti.

3.3.1.1 Raccolta dei parametri di alto livello (*HL*)

Una volta eseguito il test, JMeter fornisce i parametri di alto livello in un file .jmx. In particolare, il workload di alto livello (*HL*) è costituito da 4464 entry e 17 parametri, tra cui vale la pena menzionare:

- *Elapsed*: tempo trascorso tra l'invio della richiesta del client e la ricezione dell'ultimo pacchetto della risposta del server;
- *ThreadName*: nome del Thread Group a cui appartiene il thread che ha effettuato la richiesta;
- *Success*: esito della richiesta;
- *Bytes*: bytes ricevuti dal client;
- *AllThreads*: numero di threads attivi in tutti i Thread Group al momento della richiesta;
- *URL*: URL della risorsa richiesta al server;
- *Latency*: tempo trascorso tra l'invio della richiesta del client e la ricezione del primo pacchetto della risposta del server;
- *Connect*: tempo necessario per instaurare la connessione con il server.

È bene osservare che il client è stato configurato, in termini di throughput e numero di utenti, in modo da garantire che nessuna richiesta vada in errore.

3.3.1.1.1 Elapsed vs. Latency In Figura 3.1 si riportano le distribuzioni dei due parametri *Elapsed* e *Latency*.

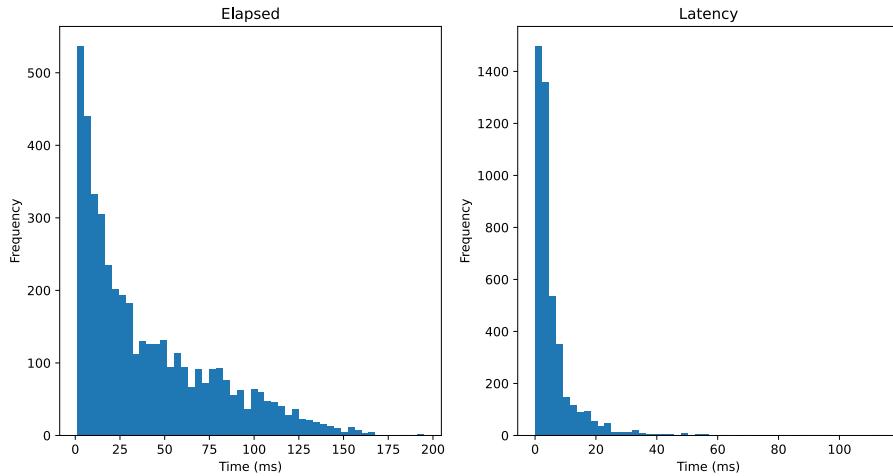


Figura 3.1: Distribuzioni parametri Elapsed e Latency

È interessante osservare le distribuzioni dei due parametri al variare del tipo di risorsa. Dal grafico in Figura 3.2 si evince che *Elapsed* dipende dalla tipologia di risorsa, mentre *Latency* è del tutto indipendente da questo fattore. Inoltre, il primo parametro cresce all'aumentare della dimensione delle risorse.

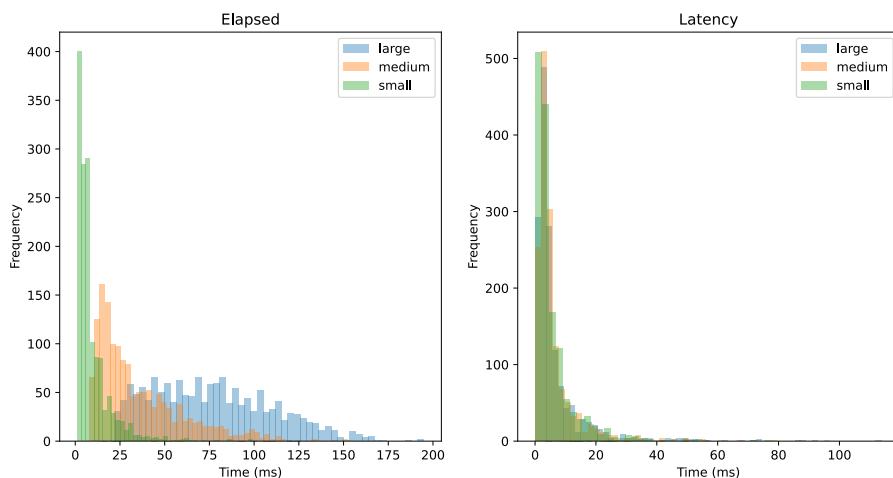


Figura 3.2: Distribuzioni parametri Elapsed e Latency per tipologia

Un'ulteriore conferma di quanto detto è data dagli scatter plot in Figura 3.3, in cui sono riportati i valori di *Elapsed* e *Latency* in funzione del tempo, al variare della tipologia di risorsa. Evidentemente, a risorse di dimensione maggiore sono associati maggiori valori di *Elapsed*.

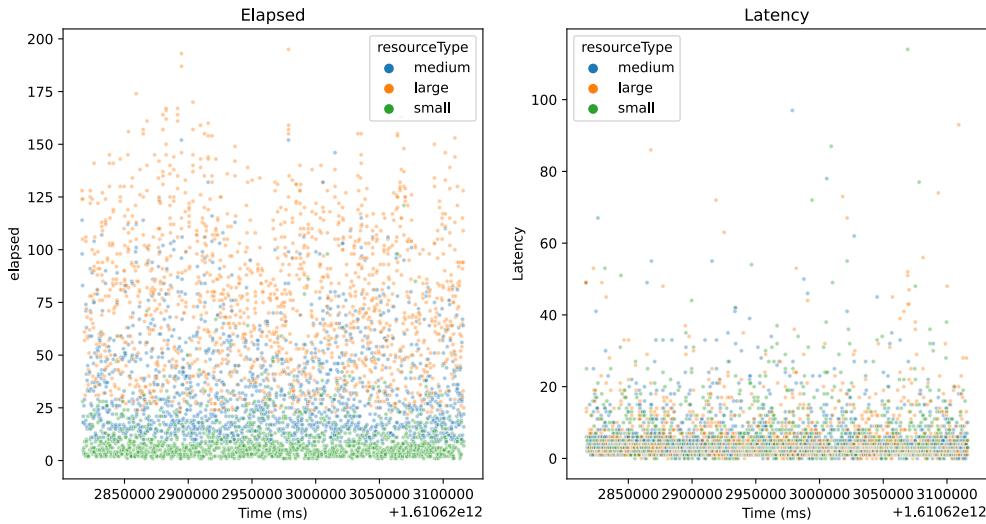


Figura 3.3: Scatter plot parametri Elapsed e Latency

Infine, in Figura 3.4 è possibile osservare per ogni tipologia di risorsa i valori medi di *Elapsed* e *Latency* ed i relativi intervalli di confidenza. Dal momento che per il parametro *Elapsed* gli intervalli di confidenza delle diverse tipologie non sono sovrapposti, è possibile concludere che le differenze tra le medie sono statisticamente significative. Non è possibile affermare lo stesso per il parametro *Latency*, essendo gli intervalli sovrapposti.

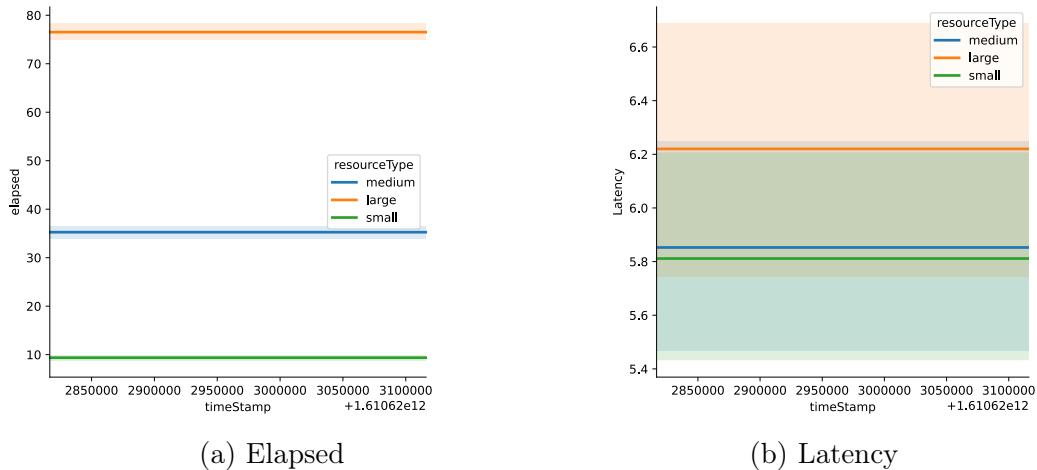


Figura 3.4: Valori medi e CI per tipologia di richiesta

3.3.1.2 Raccolta dei parametri di basso livello (*LL*)

Durante l'esecuzione del test sono stati raccolti diversi parametri di basso livello attraverso il comando: `vmstat -n 1 310 > report_server.csv`. Tale comando ha permesso di ricavare il workload di basso livello (*LL*), contenente informazioni relative

allo stato del server. Nello specifico, per la durata di 310 secondi, ogni secondo sono stati raccolti 17 parametri, tra cui vale la pena menzionare:

- *Memory*
 - *free*: memoria libera;
 - *buff*: memoria usata come buffer;
 - *cache*: memoria usata come cache;
- *System*:
 - *in*: numero di interruzioni al secondo;
 - *cs*: numero di context-switch al secondo;
- *CPU*:
 - *us*: percentuale di tempo speso in esecuzione di codice di livello utente;
 - *sy*: percentuale di tempo speso in esecuzione di codice di livello kernel;
 - *id*: percentuale di tempo in idle;
 - *wa*: percentuale di tempo speso in attesa di operazioni di I/O.

3.3.2 Caratterizzazione del workload reale

Per caratterizzare il workload reale, per ogni livello di parametri sono state effettuate le seguenti operazioni:

- *Preprocessing*
 - rimozione delle colonne nominali
 - rimozione delle colonne costanti
 - rimozione delle colonne perfettamente correlate
 - normalizzazione dei dati
- *Trade-off*: si eseguono PCA e clustering al variare delle componenti principali e del numero di cluster, al fine di individuare il miglior compromesso tra riduzione della dimensione del workload e varianza persa.
- *PCA*
- *Clustering*
- *Generazione workload sintetico*: si costruisce il workload sintetico estraendo una entry da ogni cluster.

3.3.2.1 Caratterizzazione dei parametri di alto livello (HL_c)

Secondo le operazioni precedentemente descritte, è stata applicata la PCA al dataset costituito dai seguenti parametri di alto livello: *Elapsed*, *Byte*, *sentByte*, *grpThreads*, *allThreads*, *Latency*, *IdleTime*, *Connect*.

Successivamente, è stato applicato il metodo di Ward per clusterizzare il dataset costituito dalle prime 5 componenti principali, selezionando 8 cluster.

In Figura 3.5 è possibile osservare l'andamento della devianza persa in funzione del numero di cluster, al variare delle componenti principali. Con i parametri scelti (5 componenti principali e 8 cluster) viene preservato circa il 70% di devianza.

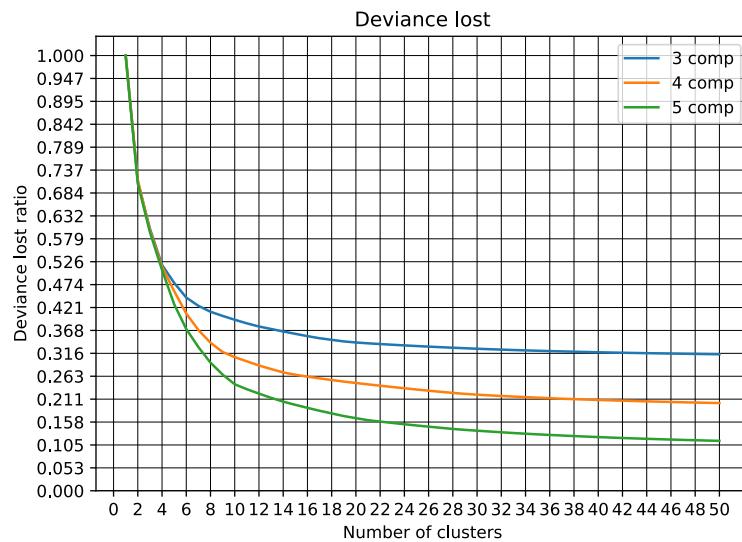


Figura 3.5: Tradeoff devianza persa

Prima di costruire il workload sintetico, sono state studiate alcune caratteristiche dei cluster ottenuti.

In Figura 3.6 è riportato il numero di elementi contenuti in ogni cluster. Evidentemente solo i primi 4 cluster presentano un numero di elementi significativo.

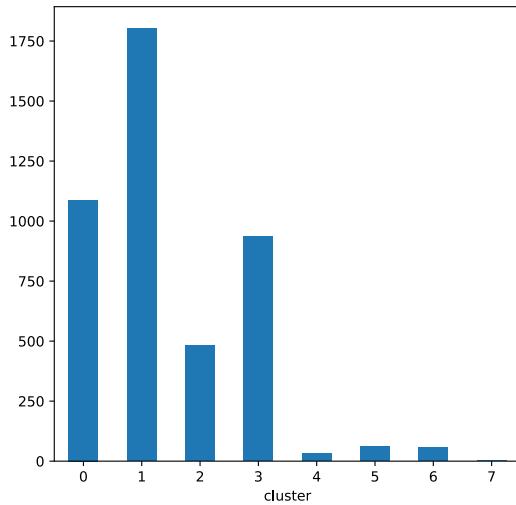


Figura 3.6: Dimensioni dei cluster

In Figura 3.7 è possibile osservare il numero di richieste per diverse tipologie di risorsa contenute in ogni cluster. Questa informazione sarà considerata nell'estrazione di un elemento rappresentativo di ogni cluster.

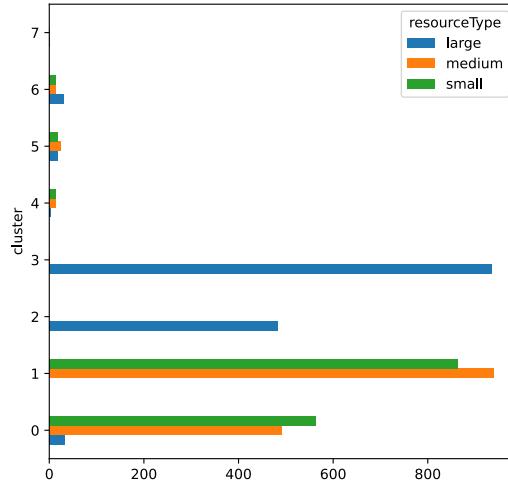


Figura 3.7: Numero di richieste per tipo di risorsa

Dai grafici precedenti si evince che gli ultimi 4 cluster hanno una dimensione notevolmente inferiore rispetto agli altri. Per verificarne la significatività sono stati calcolati i valori medi dei parametri degli elementi di ogni cluster. In Figura 3.8 è possibile osservare

che gli ultimi 4 cluster presentano valori anomali di *Latency*, *Connect* e *allThreads*, per cui è possibile concludere che gli elementi contenuti in tali cluster sono outlier. Considerando la natura dei due parametri, questi outlier non sono stati ritenuti significativi, di conseguenza i relativi cluster sono stati scartati

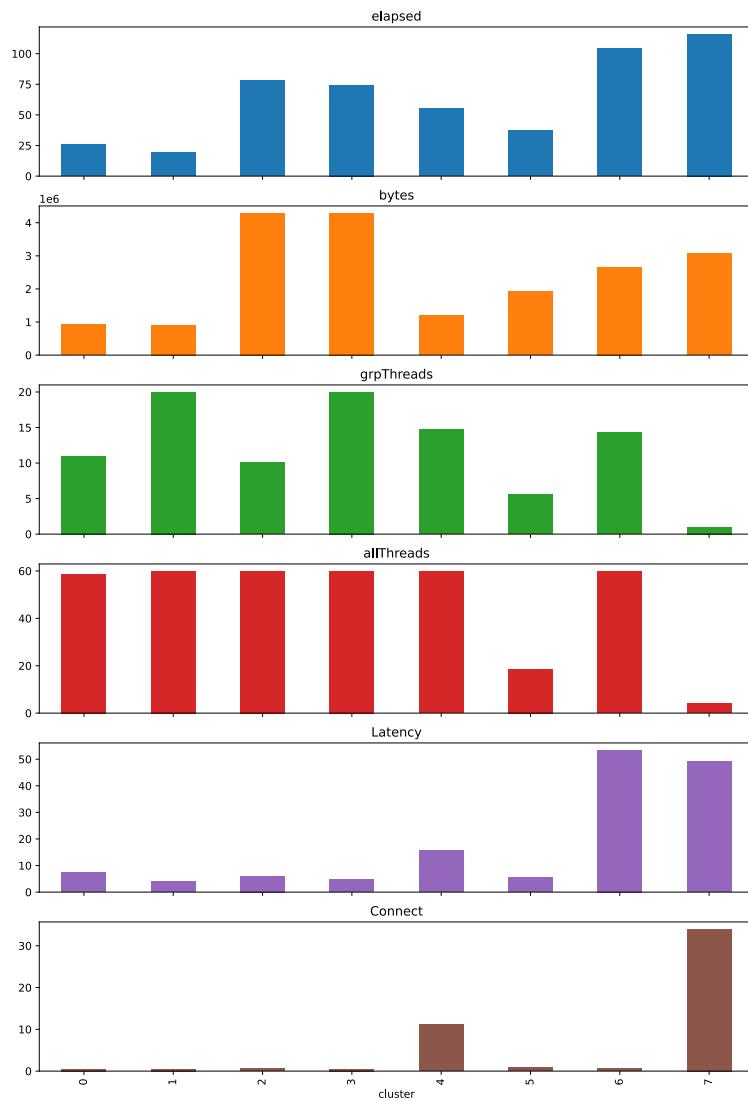


Figura 3.8: Numero di richieste per tipo di risorsa

Come ultimo step è stato costruito il workload sintetico, estraendo un elemento da

ognuno dei primi 4 cluster. L'estrazione non è stata casuale, ma si è ragionato sulle caratteristiche dei cluster. In particolare, per ogni cluster è stata individuata la tipologia di risorsa più frequente. Successivamente è stata estratta la richiesta associata alla risorsa più frequente della data tipologia. Individuata la richiesta si è tenuto conto anche del Thread Group a cui appartiene il thread che l'ha generata.

In conclusione, il workload sintetico di alto livello (HL_c) è costituito dalle richieste riportate in Tabella 3.6.

Cluster	Tipo di richiesta	Risorsa
0	fast - low - small	Wikipedia Pag2
1	fast - high - medium	Repubblica Pag1
2	fast - low - large	YouTube Home
3	fast - high - large	YouTube Pag1

Tabella 3.6: Richieste costituenti il workload sintetico

3.3.2.2 Caratterizzazione dei parametri di basso livello (LL_c)

Secondo le operazioni precedentemente descritte, è stata applicata la PCA al dataset costituito dai seguenti parametri di basso livello: *procs_r*, *procs_b*, *memory_free*, *memory_buff*, *memory_cache*, *swap_si*, *swap_so*, *io.bi*, *io.bo*, *system_in*, *system_cs*, *cpu_us*, *cpu_sy*, *cpu_id*, *cpu_wa*

Successivamente, è stato applicato il metodo di Ward per clusterizzare il dataset costituito dalle prime 8 componenti principali, selezionando 20 cluster.

In Figura 3.9 è possibile osservare l'andamento della devianza persa in funzione del numero di cluster, al variare delle componenti principali. Con i parametri scelti (8 componenti principali e 20 cluster) viene preservato circa il 90% di devianza.

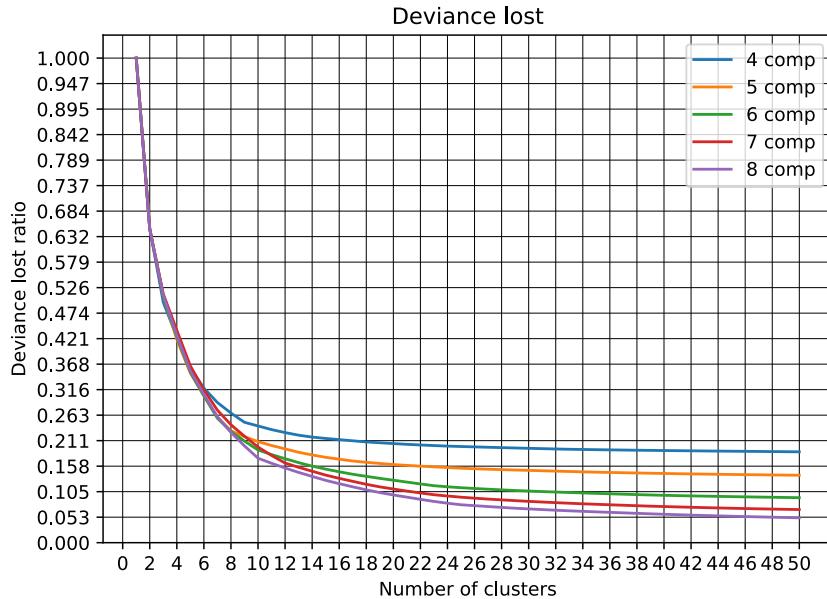


Figura 3.9: Tradeoff devianza persa

Infine, è stato costruito il workload sintetico di basso livello (LL_c) estraendo casualmente un campione da ogni cluster.

3.3.3 Applicazione del workload sintetico di alto livello

Ottenuto il workload sintetico di alto livello (HL_c), è stato effettuato un nuovo test, configurando il client in modo da effettuare solo le richieste riportate in Tabella 3.6.

3.3.3.1 Raccolta dei parametri di basso livello (LL')

Durante l'esecuzione del test sono stati raccolti nuovamente i parametri di basso livello attraverso il comando: `vmstat -n 1 310 > report_server_synth.csv`.

3.3.4 Caratterizzazione del workload sintetico

3.3.4.1 Caratterizzazione dei parametri di basso livello (LL'_c)

Sono stati caratterizzati i parametri di basso livello ottenuti applicando il workload sintetico al web server, seguendo difatti le stesse operazioni descritte nel Paragrafo 3.3.2.2.

In Figura 3.10 è possibile osservare l'andamento della devianza persa in funzione del numero di cluster, al variare delle componenti principali. Con i parametri scelti (8 componenti principali e 20 cluster) viene preservato circa il 80% di devianza.

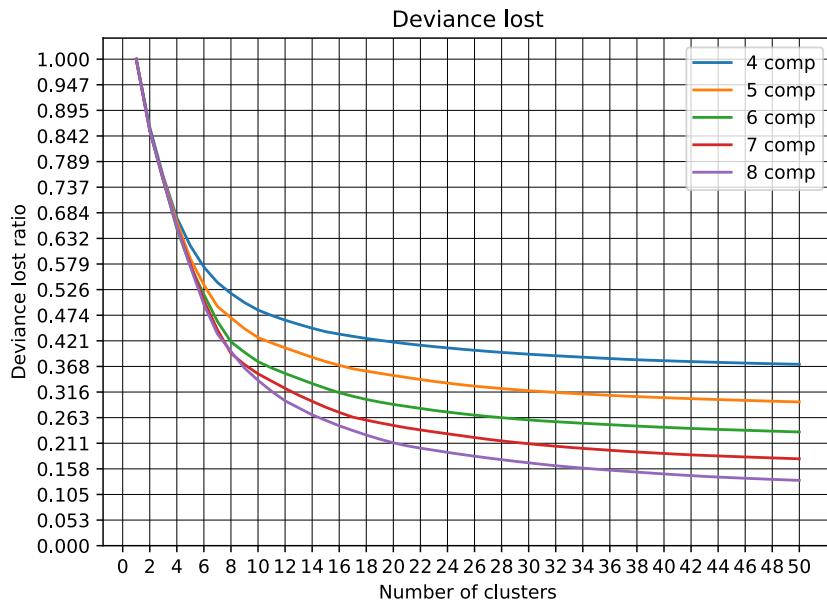


Figura 3.10: Tradeoff devianza persa

Infine, è stato costruito il nuovo workload sintetico di basso livello (LL'_c) estraendo casualmente un campione da ogni cluster.

3.3.5 Validazione del workload sintetico di alto livello

Per validare il workload sintetico di alto livello (HL_c) ottenuto a partire da quello reale (HL), sono stati confrontati i due workload sintetici di basso livello, uno (LL_c) relativo all'applicazione del workload reale di alto livello, l'altro (LL'_c) relativo all'applicazione di quello sintetico di alto livello.

Il confronto di LL_c e LL'_c è stato effettuato mediante un **test di ipotesi sulla differenza delle medie**. In particolare, si è preferito ragionare sulle prime 8 componenti principali dei due workload, ottenute durante il processo di caratterizzazione.

Per prima cosa, per entrambi i campioni è stata verificata la normalità della distribuzione dei valori di ogni componente principale. A tal fine, tramite JMP per ogni componente è stato tracciato un **QQ-plot** ed è stato effettuato il **test di Shapiro-Wilk**.

In Figura 3.11 è riportato un esempio di test di normalità effettuato per un parametro del workload.

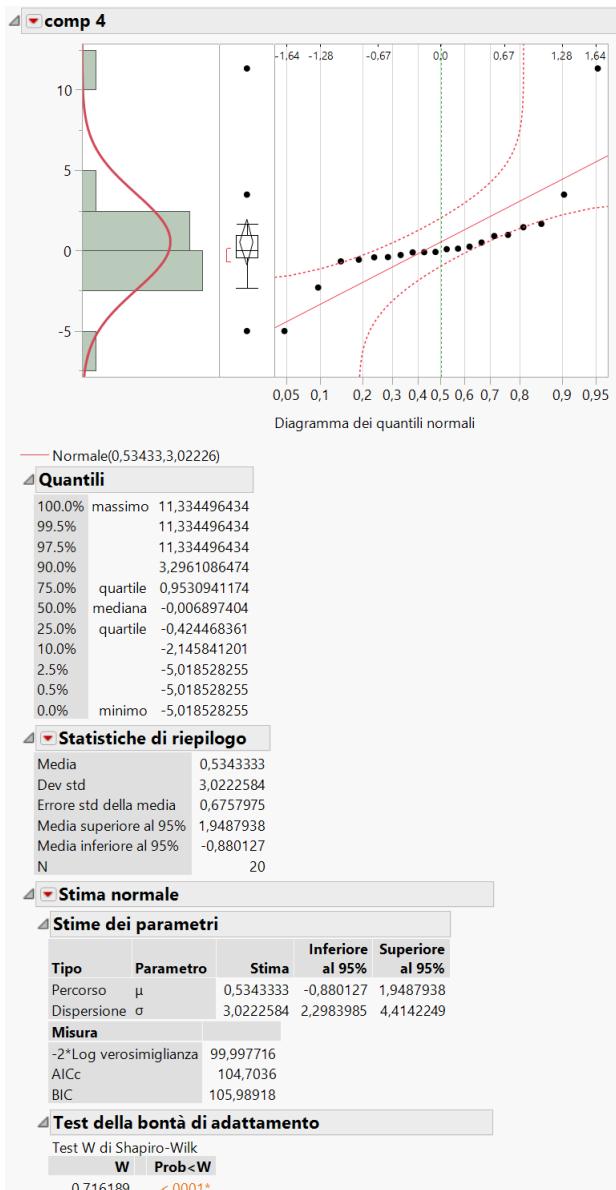


Figura 3.11: Esempio di test di normalità

Dal momento che in alcuni casi la distribuzione dei valori non è risultata normale, si è dovuto optare per un test di ipotesi non parametrico. Nello specifico, è stato applicato il **test dei ranghi con segno di Wilcoxon**. Per tutte le colonne considerate, il test ha restituito p-value maggiori del livello di significatività fissato ($\alpha = 0.05$), per cui l'ipotesi nulla è stata accettata in tutti i casi: i dati provengono dalla stessa distribuzione.

In Figura 3.12 è riportato un esempio di test sulle medie effettuato per un parametro del workload.

Test di Wilcoxon/Kruskal-Wallis (somme dei ranghi)						
Livello	Conteggio	Somma degli score	Score atteso	Media degli score	(Media-Media0)/Std0	
LL_c	20	387,000	410,000	19,3500	-0,609	
LL'_c	20	433,000	410,000	21,6500	0,609	
Test a due campioni, approssimazione normale						
S	Z	Prob> Z				
433	0,60863	0,5428				

Figura 3.12: Esempio di test sulle medie

In conclusione, è possibile affermare che non esistono differenze significative tra i due workload sintetici di basso livello, per cui la caratterizzazione del workload di alto livello può essere ritenuta corretta.

3.4 Capacity test

Nella sezione corrente sono riportate le operazioni effettuate per eseguire il capacity test del sistema, il cui scopo è quello di studiarne le performance al variare del carico imposto, al fine di dimensionarne opportunamente le risorse. In particolare, le metriche di performance valutate sono:

- **throughput**, inteso come numero di richieste al secondo servite dal web server, ovvero numero di richieste al secondo inviate dal client che non sono andate in errore.
- **tempo di risposta**, inteso come tempo trascorso tra l'invio di una richiesta del client e la ricezione dell'ultimo pacchetto della risposta del server (*Elapsed*).

Il capacity test è stato effettuato per diverse configurazioni di richieste effettuate dal client:

- *Small*: il client richiede solo risorse di tipo small;
- *Large*: il client richiede solo risorse di tipo large;
- *Random*: il client richiede tutti i tipi di risorse, in maniera casuale.

Ai fini del capacity test è fondamentale aver effettuato precedentemente la caratterizzazione del workload. Questo perché avendo ottenuto un workload sintetico rappresentativo di un workload reale, è stato possibile considerare un numero limitato di risorse per ciascuna tipologia. Sono state infatti considerate esclusivamente le risorse del workload sintetico riportate in Tabella 3.6.

Di seguito si riportano le configurazioni di Apache JMeter adottate per realizzare i diversi capacity test:

- In tutti i casi, le richieste sono state effettuate da 100 threads appartenenti ad un unico **Thread Gruoup**.

- Nei primi due casi (Small e Large), ciascun thread esegue ad ogni attivazione tutte le richieste di una data tipologia, tra quelle appartenenti al workload sintetico, attraverso un **Simple Controller**.
- Nell'ultimo caso (Random), ciascun thread ad ogni attivazione sceglie casualmente, attraverso **Random Controller**, una tipologia di risorsa ed esegue tutte le richieste di quella tipologia contenute nel workload sintetico.
- In tutti i casi, il numero di richieste effettuate dal client viene imposto attraverso un **Costant Throughput Timer**.

Ogni capacity test ha previsto l'esecuzione di un certo numero di test, in cui il client applicava un determinato carico al server. Per ogni valore di carico fissato sono state effettuate **5 ripetizioni**, della **durata di 1 minuto** ciascuna. Di ogni ripetizione sono stati calcolati throughput e tempo di risposta medio.

Successivamente, per ogni test sono state calcolate le **medie** di throughput e tempo di risposta su tutte le ripetizioni. In questo modo, per ogni valore di carico è stata ottenuta una terna di valori: carico (*req/s*), throughput (*req/s*) e tempo di risposta (*ms*).

A partire da questi valori, per ogni capacity test sono stati tracciati due grafici in cui è riportato l'andamento di throughput e tempo di risposta al variare del carico. Attraverso tali grafici si possono determinare due valori significativi di capacità del server, noti come:

- **Knee capacity**: throughput in corrispondenza del punto in cui si ha il miglior compromesso tra throughput e response time. Tipicamente si ha in corrispondenza del gomito del tempo di risposta o del throughput.
- **Usable capacity**: throughput massimo raggiungibile dal sistema senza superare una determinata soglia di response time. Tipicamente si ha in corrispondenza del punto di saturazione del throughput.

Per facilitare l'individuazione di questi valori, per ogni capacity test è stato tracciato un grafico aggiuntivo, in cui viene riportato l'andamento della **potenza**, definita come rapporto tra throughput e tempo di risposta, al variare del carico. Tipicamente la knee capacity si ha infatti nei pressi del punto di massimo della curva, mentre la usable capacity si ha quando la curva si avvicina allo 0.

Infine, è bene osservare che in tutte e tre i capacity test l'aumento del carico è stato arrestato una volta raggiunta una buona percentuale di errore (10% - 20%)

3.4.1 Risorse Small

Il capacity test relativo alle risorse di tipo Small è stato effettuato facendo variare il carico da 166.66 *req/s* (10000 *req/min*) a 2000 *req/s* (120000 *req/min*), con un passo di 166.66 *req/s* (10000 *req/min*).

In Figura 3.13 si riporta l'andamento di tempo di risposta, throughput e potenza al variare del carico.

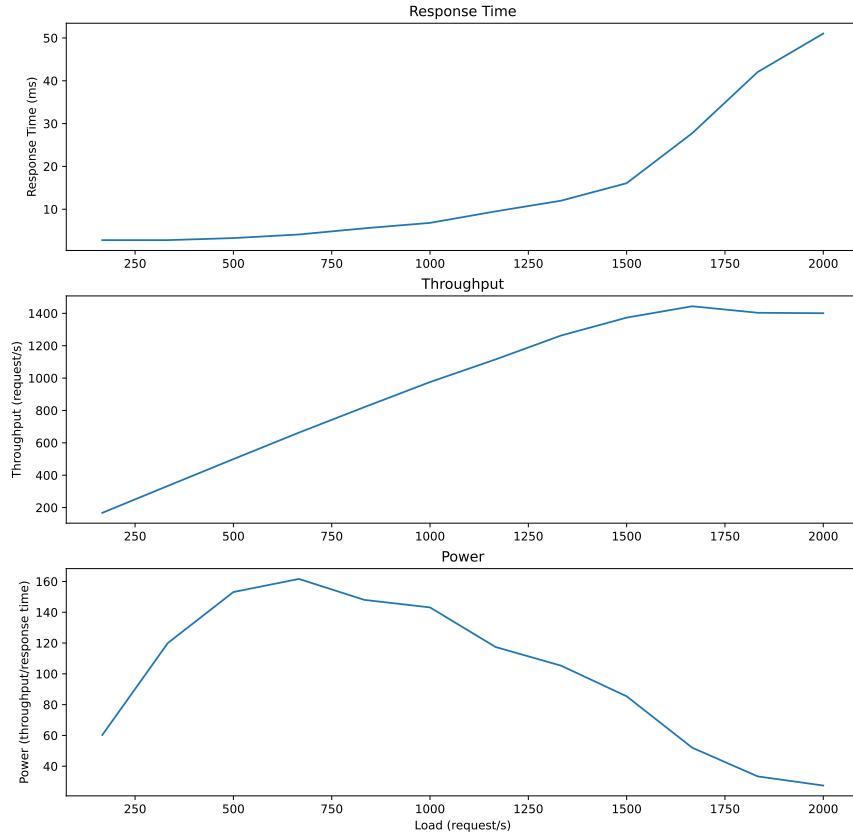


Figura 3.13: Capacity test risorse di tipo Small

Sulla base dei grafici precedenti, sono stati scelti i valori di knee e usable capacity riportati in Tabella 3.7.

Knee Capacity (req/s)	Usable Capacity (req/s)
666.666	1666.666

Tabella 3.7: Risultati capacity test risorse di tipo Small

3.4.2 Risorse Large

Il capacity test relativo alle risorse di tipo Large è stato effettuato facendo variare il carico da 33.33 req/s (2000 req/min) a 333.33 req/s (20000 req/min), con un passo di 33.33 req/s (2000 req/min).

In Figura 3.14 si riporta l'andamento di tempo di risposta, throughput e potenza al variare del carico.

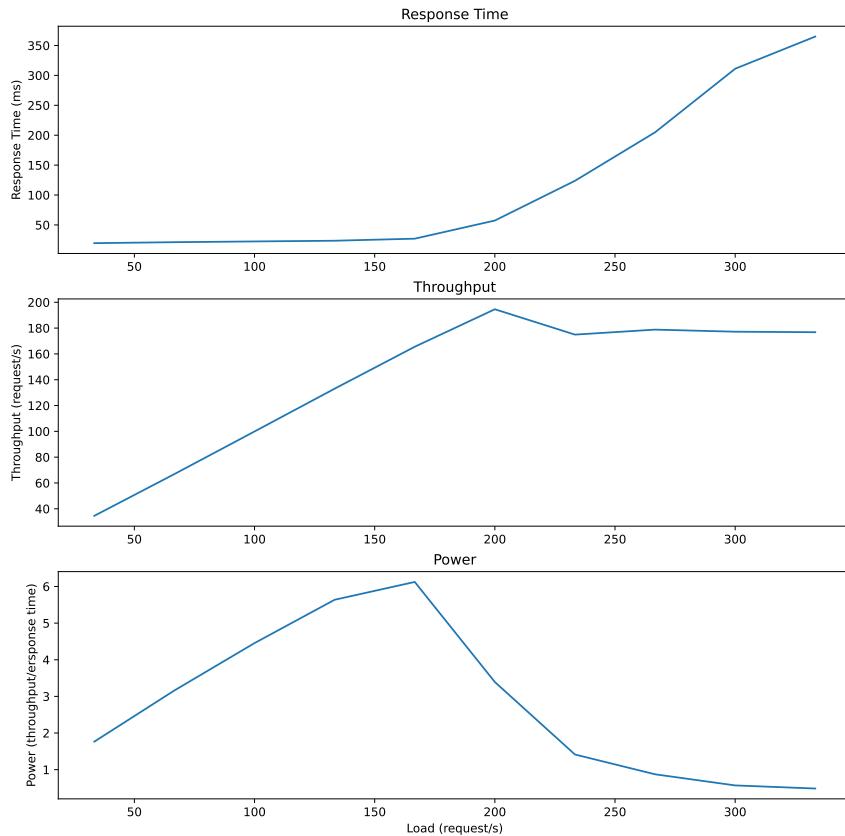


Figura 3.14: Capacity test risorse di tipo Large

Sulla base dei grafici precedenti, sono stati scelti i valori di knee e usable capacity riportati in Tabella 3.8. In questo caso si è preferito individuare il punto di knee capacity un attimo prima del punto di massimo della potenza, per garantire un maggiore distacco tra knee e usable capacity.

Knee Capacity (req/s)	Usable Capacity (req/s)
133.333	266.66

Tabella 3.8: Risultati capacity test risorse di tipo Large

3.4.3 Risorse Random

Il capacity test relativo alle risorse di tipo Random è stato effettuato facendo variare il carico da 83.33 req/s (5000 req/min) a 833.33 req/s (50000 req/min), con un passo di 83.33 req/s (5000 req/min).

In Figura 3.15 si riporta l'andamento di tempo di risposta, throughput e potenza al variare del carico.

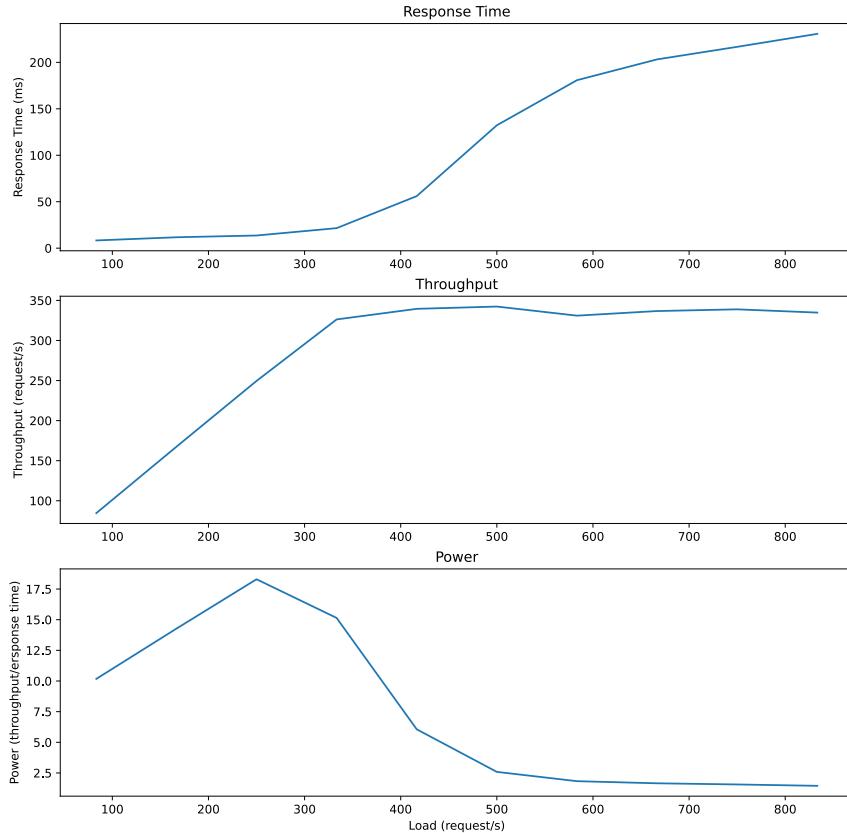


Figura 3.15: Capacity test risorse di tipo Random

Sulla base dei grafici precedenti, sono stati scelti i valori di knee e usable capacity riportati in Tabella 3.9.

Knee Capacity (req/s)	Usable Capacity (req/s)
200	583

Tabella 3.9: Risultati capacity test risorse di tipo Random

3.5 Design of Experiment (DoE)

In questa sezione si riporta l'ultima fase dell'analisi condotta sul web server, ovvero il design of experiment, il cui scopo è quello di determinare l'influenza di alcuni fattori sui

tempi di risposta del sistema. Nello specifico, sono stati considerati 2 fattori, ognuno dei quali presenta 3 livelli:

- **tasso di richieste** effettuate dal client. In questo caso, i possibili valori del fattore sono stati definiti come percentuale della usable capacity determinata in precedenza per le risorse Random:
 - *Low*: 25% della usable capacity (145req/s);
 - *Medium*: 50% della usable capacity (291req/s);
 - *High*: 75% della usable capacity (437req/s);
- **tipo di risorsa** richiesta dal client, il quale può assumere i valori *Small*, *Medium* e *Large*.

Essendo il numero di possibili combinazioni dei valori assunti dai fattori ragionevole, si è scelto di adottare un **full-factorial design**. Inoltre, per ogni combinazione sono state eseguite **5 ripetizioni**, per un totale di $9 \cdot 5 = 45$ esperimenti.

3.5.1 Effetti dei livelli dei fattori

Per prima cosa è necessario definire il modello della risposta del sistema.

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

dove:

- μ è la media della risposta su tutti gli esperimenti eseguiti;
- α_j è l'effetto del livello j del primo fattore, ovvero del tasso di richiesta;
- β_i è l'effetto del livello i del secondo fattore, ovvero del tipo di risorsa;
- γ_{ij} è l'effetto dell'interazione dei livelli i e j dei due fattori;
- e_{ijk} è l'errore commesso dal modello.

Definito il modello, è possibile determinare gli effetti dei livelli di ciascun fattore come segue:

$$\begin{aligned}\alpha_j &= \bar{y}_{.j.} - \bar{y}_{...} \\ \beta_i &= \bar{y}_{i..} - \bar{y}_{...} \\ \gamma_{ij} &= \bar{y}_{ij.} - \bar{y}_{i..} - \bar{y}_{.j.} - \bar{y}_{...}\end{aligned}$$

dove:

- $\bar{y}_{.j.}$ è la media della risposta ottenuta fissando il livello j del primo fattore e considerando tutte le ripetizioni di tutti i livelli del secondo fattore;

- $\bar{y}_{i..}$ è la media della risposta ottenuta fissando il livello i del secondo fattore e considerando tutte le ripetizioni di tutti i livelli del primo fattore;
- $\bar{y}_{ij..}$ è la media della risposta ottenuta fissando i livello j e i rispettivamente di primo e secondo fattore e considerando tutte le ripetizioni;
- $\bar{y}..._{..}$ è la media della risposta su tutti gli esperimenti eseguiti;

Si riporta in Figura 3.16 il modello utilizzato per stimare la risposta del sistema a partire dai livelli dei fattori, nel quale si ritrovano tutti gli effetti precedentemente definiti. Evidentemente, la risposta stimata dal modello è pari a:

$$\hat{y}_{ij} = \mu + \alpha_j + \beta_i + \gamma_{ij}$$

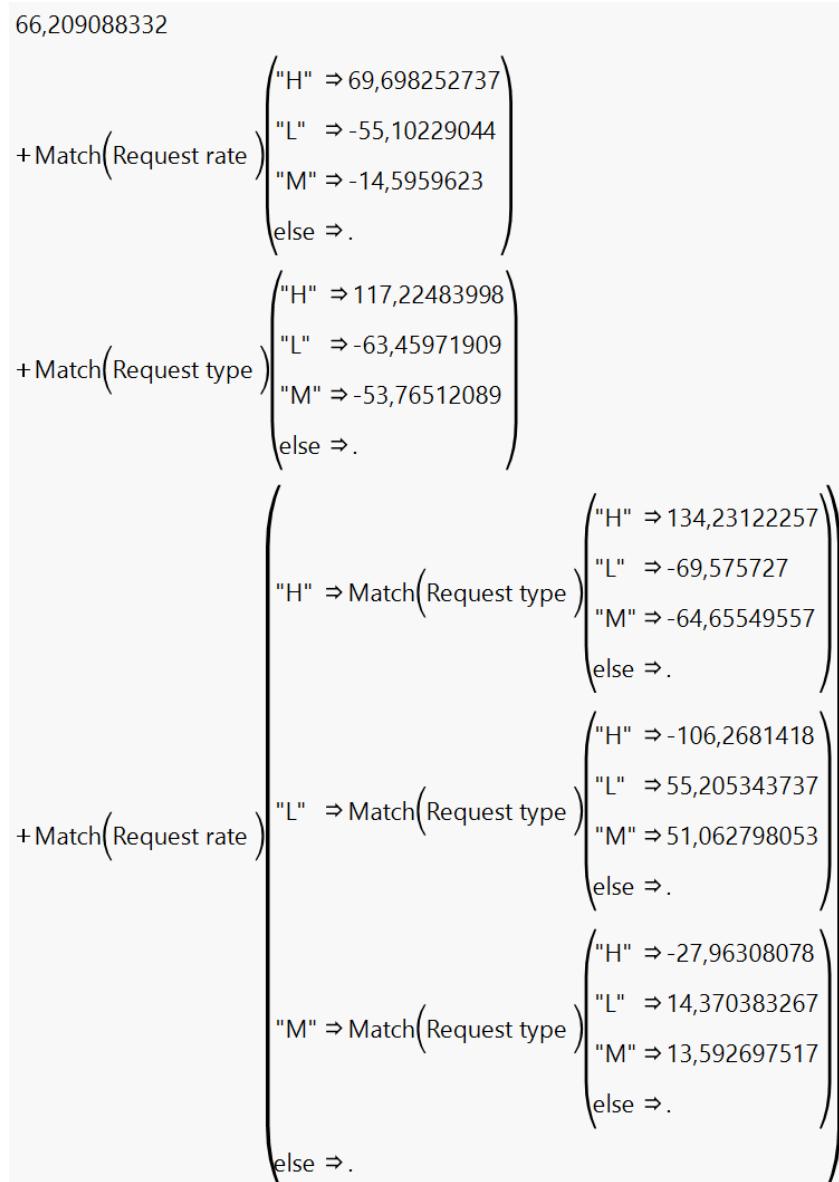


Figura 3.16: Modello della risposta

3.5.2 Errori

A partire dalla risposta stimata \hat{y}_{ij} e dalla risposta osservata y_{ijk} è possibile calcolare l'errore commesso nella stima come:

$$e_{ij} = y_{ijk} - \hat{y}_{ij}$$

3.5.3 Importanza

Noti gli effetti dei fattori e gli errori commessi è possibile calcolare l'importanza dei fattori, dell'interazione e dell'errore. Si definisce importanza di un fattore (o interazione o errore) il rapporto tra la devianza del fattore (o interazione o errore) e la devianza totale delle risposte osservate.

Le devianze utili per il calcolo dell'importanza si calcolano come:

$$\begin{aligned} SST &= \sum_{i=1}^3 \sum_{j=1}^3 \sum_{k=1}^5 (y_{ijk} - \bar{y}_{...}) \\ SSA &= 3 \cdot 5 \cdot \sum_{j=1}^3 \alpha_j^2 \\ SSB &= 3 \cdot 5 \cdot \sum_{i=1}^3 \beta_i^2 \\ SSAB &= 5 \cdot \sum_{i=1}^3 \sum_{j=1}^3 \gamma_{ij}^2 \\ SSE &= \sum_{i=1}^3 \sum_{j=1}^3 \sum_{k=1}^5 e_{ijk}^2 \end{aligned}$$

In Figura 3.17 sono riportati i valori delle devianze.

Riepilogo della stima				
	Riquadro	0,998906		
R-quadro corretto		0,998663		
Scarto quadratico medio		4,472145		
Media della risposta		66,20909		
Osservazioni (o somme pesate)		45		
Analisi della varianza				
Origine	DF	Somma dei quadrati	Media quadratica	Rapporto F
Modello	8	657301,06	82162,6	4108,115
Errore	36	720,00	20,0	Prob > F
C. totale	44	658021,06		<,0001*
Stime dei parametri				
Test degli effetti				
Origine	Nparm	DF	Somma dei quadrati	Rapporto F
Request rate	2	2	121607,26	3040,170 <,0001*
Request type	2	2	309892,31	7747,277 <,0001*
Request rate*Request type	4	4	225801,48	2822,507 <,0001*

Figura 3.17: Analisi della varianza

Ottenute le devianze si calcola l'importanza dei fattori, dell'interazione e dell'errore:

$$\frac{SSA}{SST} = 0.185$$

$$\frac{SSB}{SST} = 0.471$$

$$\frac{SSAB}{SST} = 0.343$$

$$\frac{SSE}{SST} = 1.095 \cdot 10^{-3}$$

In conclusione, è possibile affermare che fattori e interazione sono notevolmente più importanti dell'errore. Inoltre, il secondo fattore, relativo al tipo di risorsa, risulta essere il più importante.

3.5.4 Significatività

L'ultimo aspetto da valutare è la significatività dei fattori, la quale rappresenta la percentuale di variabilità spiegata da un fattore rispetto a quella spiegata dall'errore.

Per studiare la significatività dei fattori considerati si applica un metodo detto **ANOVA** (ANalysis Of VAriance). Tuttavia, il metodo preciso da applicare dipende da alcune caratteristiche della distribuzione degli errori, come indicato in Tabella 3.10.

Normalità	Omoschedasticità	ANOVA
verificata	verificata	F-test
non verificata	verificata	Kruskal-Wallis test
verificata	non verificata	Welch's test
non verificata	non verificata	Kruskal-Wallis o Friedman test

Tabella 3.10: ANOVA

Per capire quale metodo applicare è necessario quindi verificare la **normalità** e l'**omoschedasticità** degli errori.

Per verificare la normalità degli errori è possibile effettuare sia un test visuale, **QQ-plot**, che un test analitico, **test di Shapiro-Wilk**, entrambi riportati in Figura 3.18. I risultati di entrambi i test rigettano l'ipotesi che gli errori provengano da una popolazione con distribuzione normale.

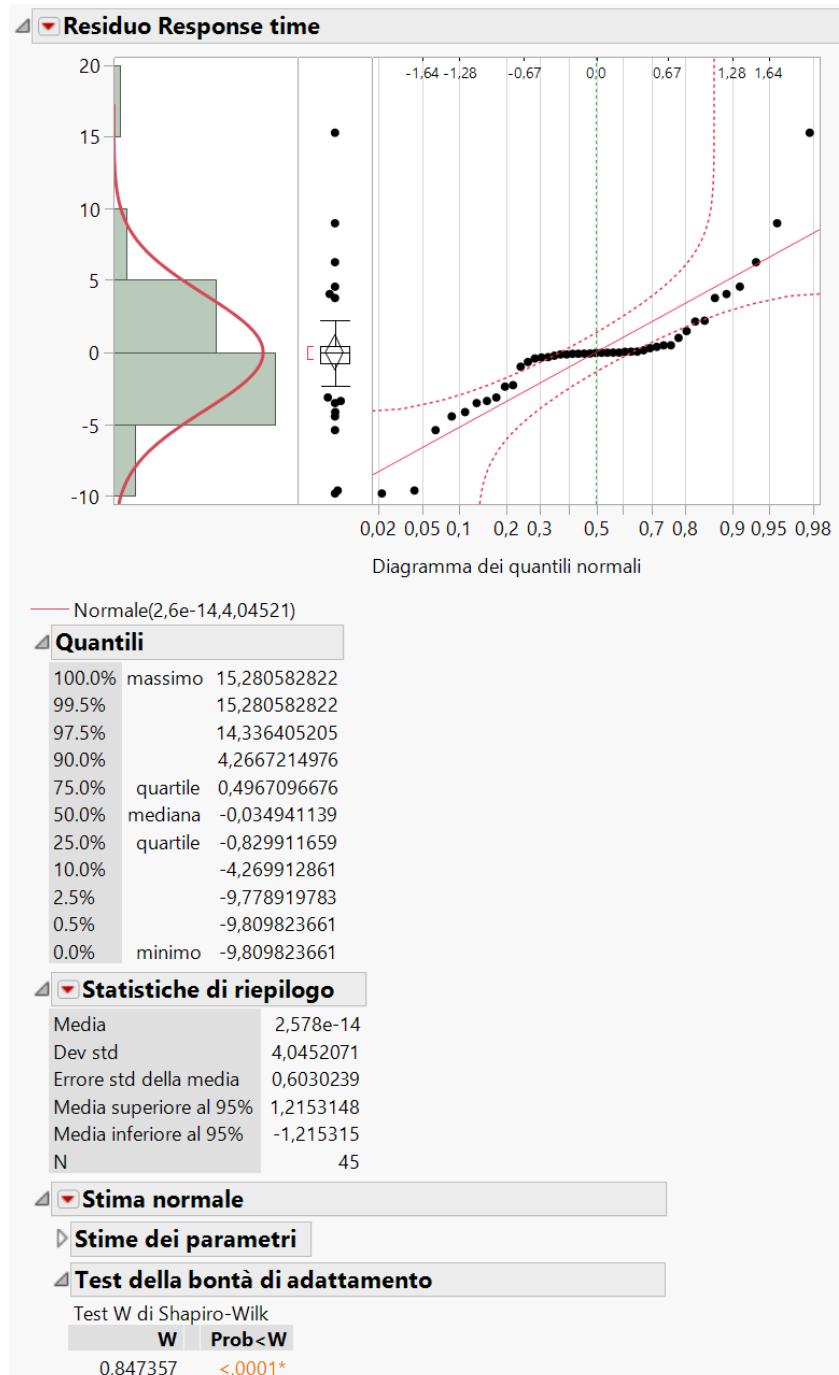


Figura 3.18: Test di normalità degli errori

Per verificare l'omoschedasticità degli errori è possibile effettuare un test visuale, tracciando uno **scatter plot risposta stimata-errore**. Come mostrato in Figura 3.19, la dispersione dei punti dello scatter plot non è costante, per cui è possibile rifiutare l'ipotesi di omoschedasticità degli errori.

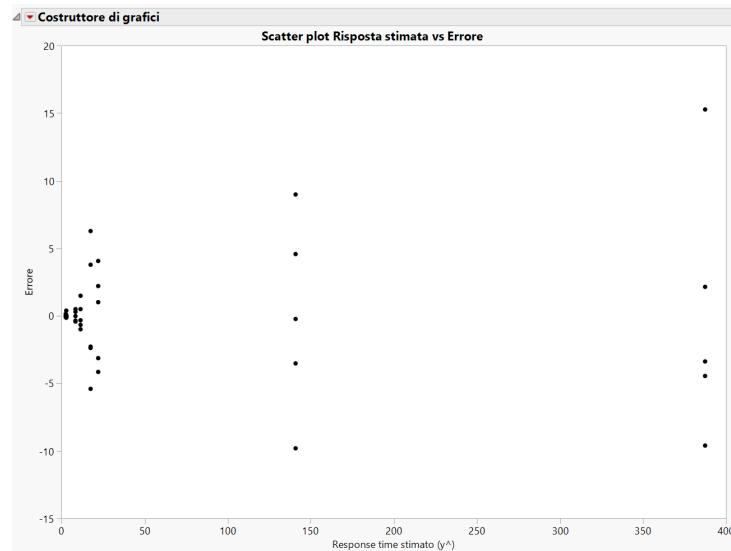


Figura 3.19: Scatter plot \hat{y} -Errore

In alternativa, è possibile effettuare un test analitico, ovvero un **test sulle varianze ineguali**, i cui risultati sono riportati in Figura 3.20. Il test rigetta l'ipotesi nulla che le varianze per ogni livello dei fattori siano uguali, per cui è possibile concludere che l'omoschedasticità degli errori non è verificata.

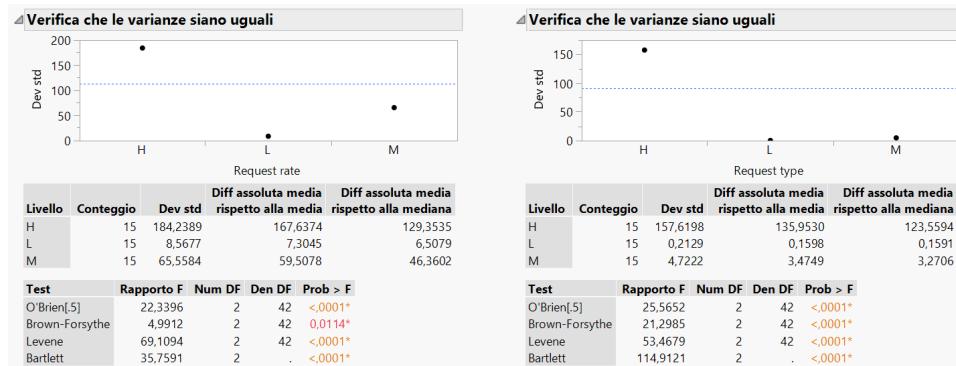


Figura 3.20: Test di omoschedasticità degli errori

Non essendo verificata né la normalità né l'omoschedasticità, si rientra nell'ultima riga della Tabella 3.10, per cui per determinare la significatività dei fattori si può effettuare un **test di Kruskal-Wallis**. Dai risultati del test riportati in Figura 3.21 emerge che solo il secondo dei due fattori, ovvero quello relativo al tipo di risorsa, è statisticamente significativo.

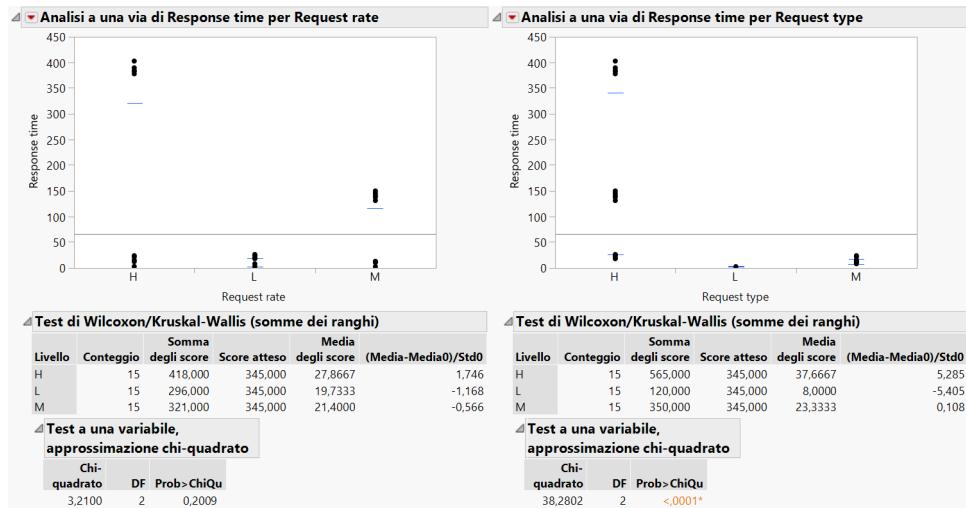


Figura 3.21: Test di significatività dei fattori

Capitolo 4

Reliability

4.1 Esercizio 1

4.1.1 Traccia

Trovare la *reliability* $R(t)$ e il *mean time to failure* ($MTTF$) del sistema avente il seguente *reliability block diagram* (RBD). Nel calcolare il valore di $MTTF$ si assuma che tutti i componenti siano identici e falliscano randomicamente con un *failure rate* λ .

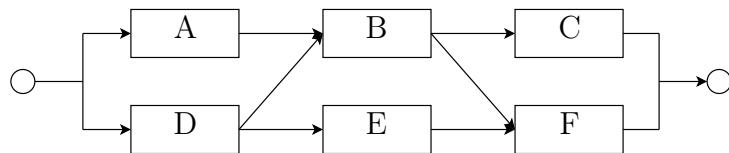


Figura 4.1: RBD

4.1.2 Soluzione

4.1.2.1 Upper bound

Innanzitutto, è possibile osservare che il sistema non presenta una struttura serie-parallelo. In questi casi si può calcolare rapidamente un limite superiore della reliability del sistema, applicando il **teorema dell'upper bound**. Tale teorema afferma che la reliability di un sistema è minore o uguale alla reliability del sistema costituito dal parallelo dei *success path*¹ del sistema originale, ovvero:

$$R_{sys} \leq 1 - \prod_i (1 - R_{path_i}) \quad (4.1)$$

dove R_{path_i} è la reliability dell'i-esimo success path del sistema.

In generale, il teorema fornisce solo un limite superiore alla reliability del sistema, tuttavia, se i success path sono indipendenti, con riferimento all'Equazione 4.1, vale la relazione di uguaglianza.

¹Un *success path* di un sistema è un qualsiasi percorso lungo l'RBD del sistema che conduce dall'input all'output.

Applicando il teorema dell'upper bound al sistema in Figura 4.1, è possibile considerare il seguente sistema:

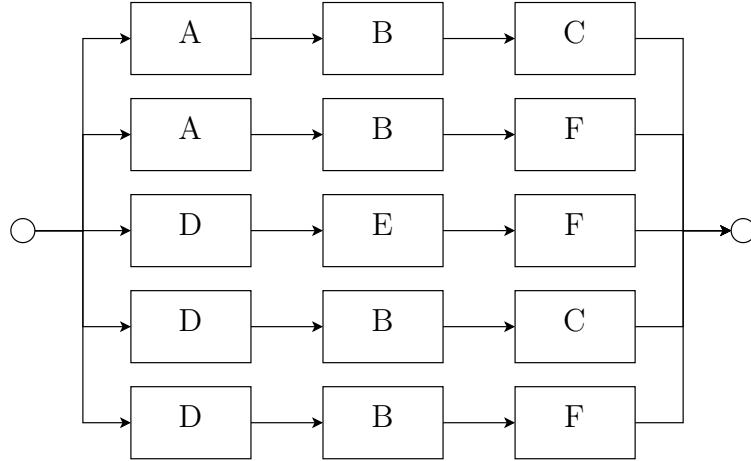


Figura 4.2: RBD parallelo dei success path

Dal momento che il nuovo sistema ha una struttura serie-parallelo, possiamo calcolarne facilmente la reliability applicando le formule per la risoluzione dei sistemi serie e parallelo.

In particolare, per i sistemi costituiti da soli componenti in **serie** è possibile calcolare la reliability del sistema a partire dalla reliability dei singoli componenti tramite la relazione:

$$R_s(t) = \prod_{i=1}^n R_i(t)$$

dove:

- n rappresenta il numero di componenti in serie
- $R_i(t)$ rappresenta la reliability dell'i-esimo componente

In maniera simile, per i sistemi costituiti da soli componenti in **parallelo** è possibile calcolare la reliability del sistema a partire dalla reliability dei singoli componenti tramite la relazione:

$$R_p(t) = 1 - \prod_{i=1}^n (1 - R_i(t))$$

dove:

- n rappresenta il numero di componenti in parallelo
- $R_i(t)$ rappresenta la reliability dell'i-esimo componente

La reliability del sistema in Figura 4.2 può essere ottenuta come:

$$R_{sys_{up}} = 1 - (1 - R_A R_B R_C)(1 - R_A R_B R_F)(1 - R_D R_E R_F)(1 - R_D R_B R_C)(1 - R_D R_B R_F)$$

Dal momento che i componenti sono identici e falliscono randomicamente con failure rate λ , la reliability dell'i-esimo componente sarà $R_i = e^{-\lambda t}$. Sostituendo l'espressione della reliability di ciascun componente otteniamo:

$$R_{sys_{up}}(t, \lambda) = 1 - (1 - e^{-3\lambda t})^5$$

Tenendo in conto che i success path del sistema originale non sono indipendenti, possiamo concludere:

$$R_{sys} \leq 1 - (1 - e^{-3\lambda t})^5$$

4.1.2.2 Conditioning

Per avere una stima più precisa della reliability del sistema è possibile applicare la tecnica del **conditioning**. Tale tecnica è basata sulla *legge della probabilità totale* la quale afferma che, dati A_1, \dots, A_n eventi mutuamente esclusivi ($A_i \cap A_j = \emptyset, \forall i \neq j$) tali che $B \subseteq \cup_{i=1}^n A_i$, vale la seguente relazione:

$$P(B) = \sum_{i=1}^n P(B|A_i)P(A_i) \quad (4.2)$$

Dal momento che la reliability $R(t)$ di un sistema è definita come la probabilità che il sistema non fallisca fino all'istante t , è possibile calcolare questa probabilità condizionandola al fallimento di un componente del sistema. Dalla 4.2 si ricava quindi:

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

dove:

- $P(sys)$ è la probabilità che il sistema funzioni;
- R_m è la reliability del componente m del sistema;
- $P(sys|m)$ è la probabilità che il sistema funzioni dato il fatto che il componente m funziona;
- $P(sys|\bar{m})$ è la probabilità che il sistema funzioni dato il fatto che il componente m fallisce;

Applichiamo la tecnica del conditioning al sistema in Figura 4.1, condizionando rispetto al componente E.

- Se E fallisce, consideriamo il seguente sistema:

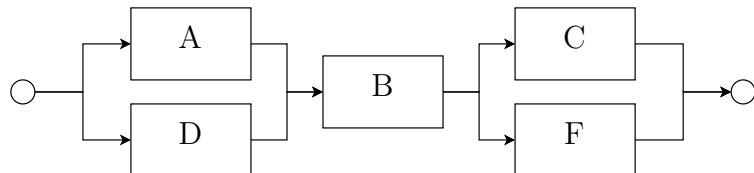


Figura 4.3: RBD condizionato al fallimento di E

Il nuovo sistema presenta una struttura serie-parallelo per cui possiamo calcolarne facilmente la reliability:

$$R_{sys_{\overline{E}}} = (1 - (1 - R_A)(1 - R_D))R_B(1 - (1 - R_C)(1 - R_F))$$

Sostituendo l'espressione della reliability di ciascun componente otteniamo:

$$R_{sys_{\overline{E}}} = (1 - (1 - e^{-\lambda t})^2)^2 e^{-\lambda t}$$

- *Se E funziona*, consideriamo il seguente sistema:

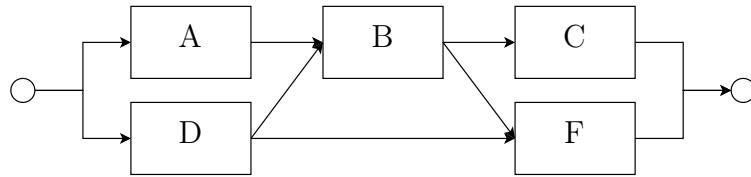


Figura 4.4: RBD condizionato al funzionamento di E

Sfortunatamente, il nuovo sistema non presenta una struttura serie-parallelo, tuttavia, per calcolarne la reliability possiamo applicare nuovamente il conditioning.

Considerando il sistema in Figura 4.4, condizioniamo rispetto al componente B:

- *Se B fallisce*, consideriamo il seguente sistema:

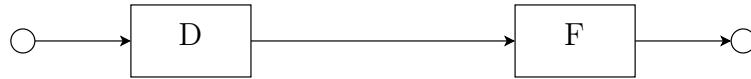


Figura 4.5: RBD condizionato al funzionamento di E e al fallimento di B

Il nuovo sistema presenta una struttura serie-parallelo per cui possiamo calcolarne facilmente la reliability:

$$R_{sys_{E\overline{B}}} = R_D R_F$$

Sostituendo l'espressione della reliability di ciascun componente otteniamo:

$$R_{sys_{E\overline{B}}} = e^{-2\lambda t}$$

- *Se B funziona*, consideriamo il seguente sistema:

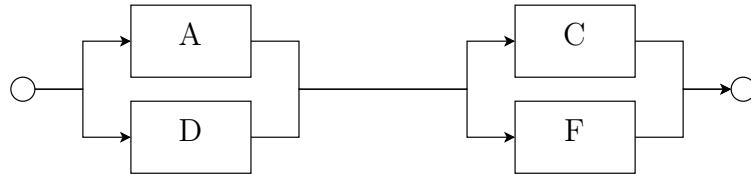


Figura 4.6: RBD condizionato al funzionamento di E e di B

Il nuovo sistema presenta una struttura serie-parallelo per cui possiamo calcolarne facilmente la reliability:

$$R_{sys_{EB}} = (1 - (1 - R_A)(1 - R_D))(1 - (1 - R_C)(1 - R_F))$$

Sostituendo l'espressione della reliability di ciascun componente otteniamo:

$$R_{sys_{EB}} = (1 - (1 - e^{-\lambda t})^2)^2$$

Possiamo quindi calcolare la reliability del sistema condizionato al funzionamento di E come:

$$\begin{aligned} R_{sys_E} &= R_B R_{sys_{EB}} + (1 - R_B) R_{sys_{E\bar{B}}} \\ &= R_B (1 - (1 - R_A)(1 - R_D))(1 - (1 - R_C)(1 - R_F)) + (1 - R_B) R_D R_F \end{aligned}$$

In conclusione, la reliability del sistema originale può essere ottenuta come:

$$\begin{aligned} R_{sys}(t, \lambda) &= R_E R_{sys_E} + (1 - R_E) R_{sys_{\bar{E}}} \\ &= \dots \\ &= e^{-5\lambda t} - 5e^{-4\lambda t} + 5e^{-3\lambda t} \end{aligned}$$

4.1.2.3 Analisi grafica

Fissando il valore di λ è possibile osservare graficamente come la funzione $R_{sys_{up}}(t)$ sia effettivamente un upper bound per la funzione di reliability del sistema $R_{sys}(t)$. Scegliendo $\lambda = 0.5$ si ottiene il grafico in Figura 4.7.

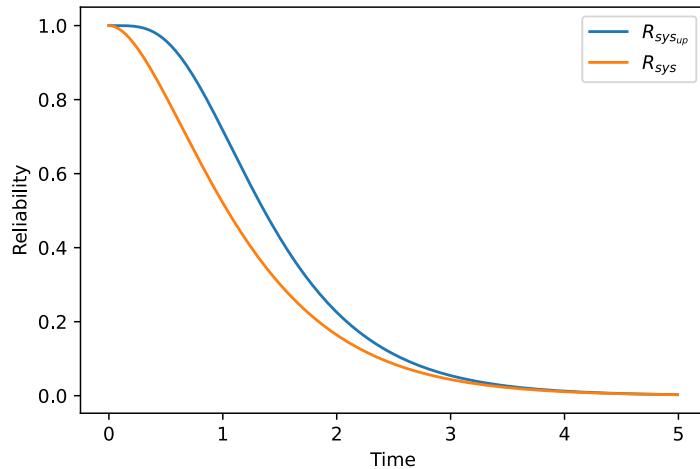


Figura 4.7: Confronto tra le funzioni $R_{sys_{up}}(t)$ ed $R_{sys}(t)$

4.1.2.4 MTTF

Una volta calcolata la reliability del sistema, per ottenere l'*MTTF* del sistema è sufficiente integrare la funzione $R_{sys}(t, \lambda)$, ovvero:

$$\begin{aligned} MTTF(\lambda) &= \int_0^{\infty} R_{sys}(t, \lambda) dt = \int_0^{\infty} (e^{-5\lambda t} - 5e^{-4\lambda t} + 5e^{-3\lambda t}) dt \\ &= \frac{1}{5\lambda} - \frac{5}{4\lambda} + \frac{5}{3\lambda} = \frac{37}{60\lambda} = 0.617\lambda \end{aligned}$$

4.2 Esercizio 2

4.2.1 Traccia

Si vogliono confrontare due diversi schemi di reliability per un sistema che utilizza la ridondanza. Si supponga che il sistema necessiti di s componenti identici in serie per operare propriamente e che siano dati $(m \times s)$ componenti.

1. Dato che la reliability di un componente è r , derivare l'espressione per le reliability delle due configurazioni.
2. Confrontare le due espressioni per $m = 2$ ed $s = 4$.
3. Quale dei due schemi mostrati in figura assicura una reliability maggiore?
4. Modificare lo schema con la reliability più bassa al fine di ottenere la stessa reliability per entrambi.

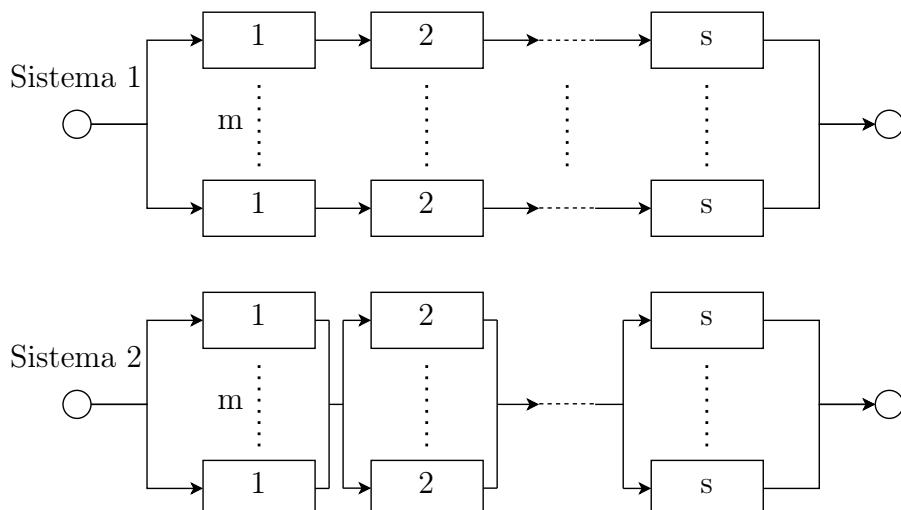


Figura 4.8: RBD

4.2.2 Soluzione

4.2.2.1 Reliability dei sistemi

Come possiamo osservare in Figura 4.8, i due sistemi hanno una struttura serie-parallelo, per cui possiamo calcolarne facilmente la reliability:

- Il *sistema 1* è costituito dal parallelo di m serie, ognuna delle quali è composta da s elementi, per cui si ha:

$$R_{sys_1} = 1 - \prod_{i=1}^m (1 - r^s) = 1 - (1 - r^s)^m \quad (4.3)$$

- Il *sistema 2* è costituito dalla serie di s paralleli, ognuno dei quali è composto da m elementi, per cui si ha:

$$R_{sys_2} = \left(1 - \prod_{i=1}^m (1 - r) \right)^s = (1 - (1 - r)^m)^s \quad (4.4)$$

4.2.2.2 Confronto tra i sistemi

Un semplice modo per determinare quale tra due sistemi è più reliable consiste nel verificare quale dei due ha il maggior numero di success path. Osservando i RBD in Figura 4.8 si evince che:

- il primo sistema presenta un numero di success path pari ad m ;
- il secondo sistema presenta un numero di success path pari ad m^s .

Per capire come si è giunti al secondo risultato, si consideri il caso di $m = 2$ ed $s = n$. In questo caso, ogni singolo success path del secondo sistema può essere visto come una stringa di n bit in cui il bit i -esimo è pari ad 1 se in posizione i -esima si attraversa il nodo superiore e pari a 0 se si attraversa il nodo inferiore. Di conseguenza, il numero di success path è pari al numero di possibili configurazioni che può assumere una stringa di n bit, ovvero 2^n . Generalizzando il discorso per un m qualsiasi si ottiene il risultato precedentemente illustrato.

In conclusione, presentando un maggior numero di success path, il secondo sistema risulta essere più reliable del primo.

4.2.2.3 Confronto tra i sistemi con $m = 2$ e $s = 4$

Per confrontare la reliability dei due sistemi con $m = 2$ e $s = 4$, per prima cosa fissiamo i valori di m ed s nelle espressioni 4.3 e 4.4, da cui si ricava:

$$\begin{aligned} R_{sys_1} &= 1 - (1 - r^4)^2 \\ R_{sys_2} &= (1 - (1 - r)^2)^4 \end{aligned}$$

In Figura 4.9 è evidenziato come, al variare del parametro r , il sistema 2 abbia sempre un valore di reliability maggiore rispetto al sistema 1.

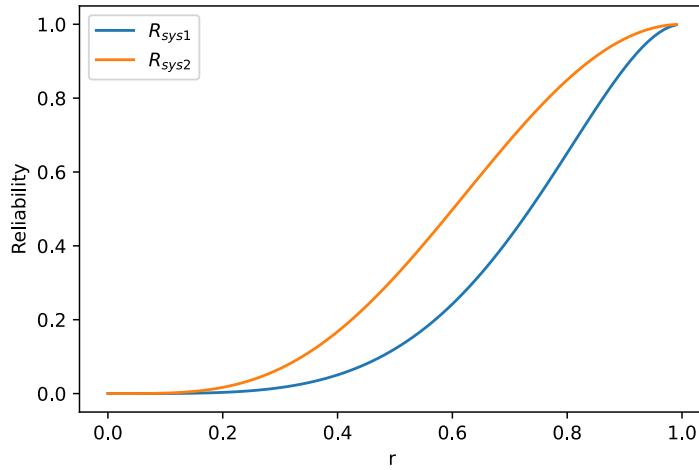


Figura 4.9: Confronto tra le funzioni $R_{sys_1}(r)$ ed $R_{sys_2}(r)$

4.2.2.4 Modifica di un sistema

Per rendere il primo sistema reliable quanto il secondo è sufficiente modificarne la struttura in modo che i due sistemi abbiano lo stesso numero di success path. Più precisamente, in questo modo si rendono uguali gli upper bound dei due sistemi. Si tenga presente che per il primo sistema l'upper bound coincide con la reliability effettiva, in quanto presenta success path indipendenti.

In base a quanto detto nel Paragrafo 4.2.2.2, fissati i valori di s ed m del secondo sistema, il primo sistema avrà lo stesso numero di success path se presenta $s' = s$ e $m' = m^s$. La reliability del primo sistema modificato sarà quindi:

$$R_{sys'_1} = 1 - (1 - r^s)^{m^s}$$

In Figura 4.10 si riporta il confronto tra le reliability dei sistemi effettuato per $m = 2$ e $s = 4$.

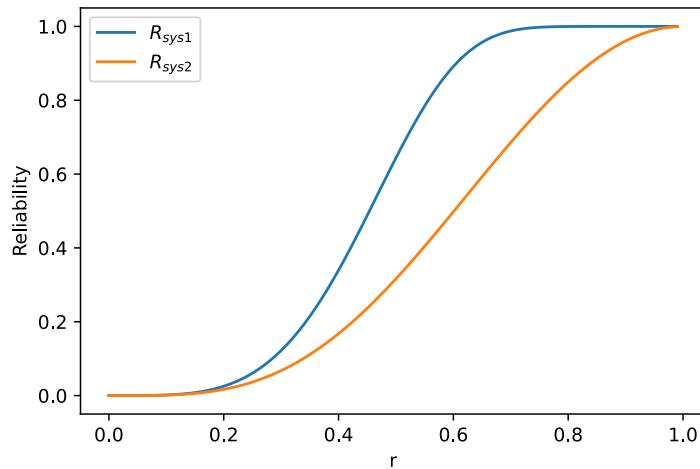


Figura 4.10: Confronto tra le funzioni $R_{sys'_1}(r)$ ed $R_{sys_2}(r)$

4.3 Esercizio 3

4.3.1 Traccia

Si riporta di seguito l'architettura di una rete di calcolatori di un sistema bancario. L'architettura è chiamata *skip-ring network* ed è progettata per permettere ai processori di comunicare anche se uno dei nodi fallisce. Ad esempio, se il nodo 1 fallisce, il nodo 8 può bypassare il nodo fallito, dirigendo i dati lungo il link alternativo che collega i nodi 8 e 2.

- Assumendo che i collegamenti siano perfetti e che ogni nodo abbia una reliability R_m , derivare l'espressione per la reliability della rete.
- Se R_m segue una legge esponenziale di fallimento ed il failure rate di ogni nodo è di 0.005 fallimenti per ora, determinare la reliability del sistema alla fine di un periodo di 48 ore.

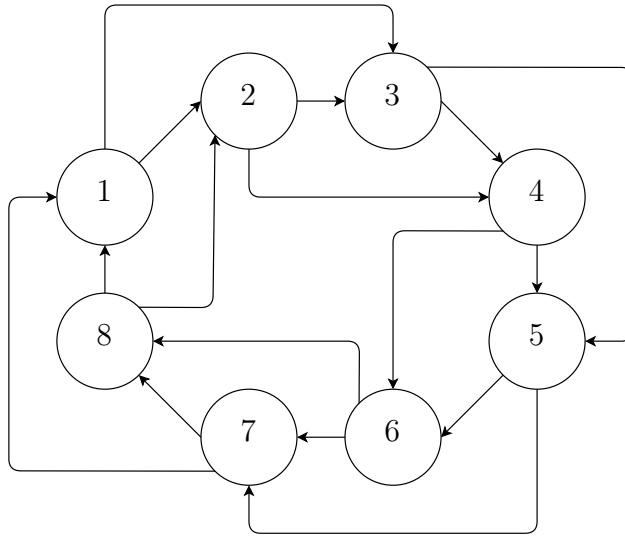


Figura 4.11: Architettura skip-ring network

4.3.2 Soluzione

4.3.2.1 Reliability del sistema

In una skip-ring network, il sistema complessivo fallisce nel caso in cui falliscano almeno due nodi consecutivi.

Per prima cosa si può osservare che se falliscono cinque nodi sicuramente il sistema fallisce.

Per calcolare la reliability del sistema è possibile considerare quindi tutte le possibili configurazioni in cui la rete funziona correttamente. In particolare, tali configurazioni sono le seguenti:

- *Nessun nodo nella rete fallisce.* In questo caso è possibile calcolare la probabilità che il sistema funzioni dato che nessun nodo della rete è fallito, ovvero $P(\text{sys}|0\text{fails})$.
- *Un solo nodo nella rete fallisce.* In questo caso è possibile calcolare la probabilità che il sistema funzioni dato che un nodo della rete è fallito, ovvero $P(\text{sys}|1\text{fails})$. Tale valore di probabilità va moltiplicato per tutte le possibili configurazioni che i nodi falliti possono assumere. In particolare, si hanno 8 possibili configurazioni in cui due nodi della rete falliscano rispettando la condizione di non adiacenza.
- *Due nodi nella rete falliscono.* In questo caso è possibile calcolare la probabilità che il sistema funzioni dato che due nodi della rete sono falliti, ovvero $P(\text{sys}|2\text{fails})$. Tale valore di probabilità va moltiplicato per tutte le possibili configurazioni che i nodi falliti possono assumere. In particolare, si hanno 20 possibili configurazioni in cui due nodi della rete falliscano rispettando la condizione di non adiacenza.
- *Tre nodi nella rete falliscono.* In questo caso è possibile calcolare la probabilità che il sistema funzioni dato che tre nodi della rete sono falliti, ovvero $P(\text{sys}|3\text{fails})$. Tale valore di probabilità va moltiplicato per tutte le possibili configurazioni che i

nodi falliti possono assumere. In particolare, si hanno 16 possibili configurazioni in cui due nodi della rete falliscano rispettando la condizione di non adiacenza.

- *Quattro nodi nella rete falliscono.* In questo caso è possibile calcolare la probabilità che il sistema funzioni dato che due nodi della rete sono falliti, ovvero $P(sys|3fails)$. Tale valore di probabilità va moltiplicato per tutte le possibili configurazioni che i nodi falliti possono assumere. In particolare, si hanno 2 possibili configurazioni in cui due nodi della rete falliscano rispettando la condizione di non adiacenza.

Sommando le probabilità delle configurazioni discusse è possibile ottenere la probabilità che il sistema funzioni correttamente:

$$\begin{aligned} R_{sys} &= P(sys) = P(sys|0fails) + 8P(sys|1fails) + 20P(sys|2fails) + \\ &\quad + 16P(sys|3fails) + 2P(sys|4fails) \\ &= R_m^8 + 8R_m^7(1 - R_m) + 20R_m^6(1 - R_m)^2 + 16R_m^5(1 - R_m)^3 + 2R_m^4(1 - R_m)^4 \end{aligned}$$

4.3.2.2 Reliability del sistema dopo 48 ore

Sostituendo $R_m(t) = e^{-\lambda t}$ con $\lambda = 0.005 \frac{\text{fallimenti}}{\text{ora}}$ all'interno dell'espressione della reliability del sistema si ottiene:

$$R_{sys}(48) \simeq 0.7289$$

In Figura 4.12 è riportato il grafico della reliability del sistema ed è evidenziato il valore $R_{sys}(48)$.

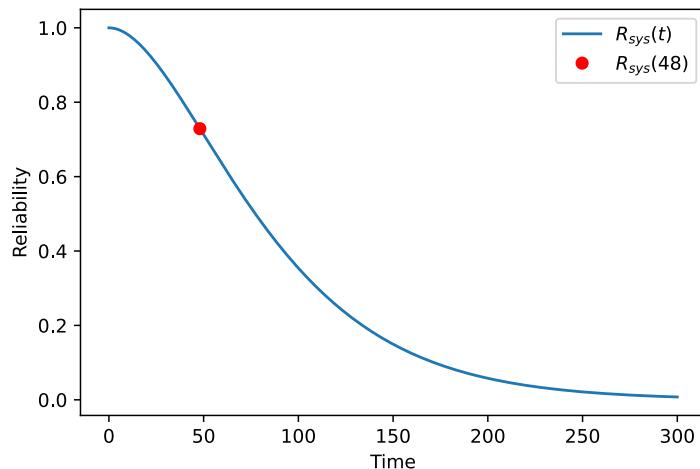


Figura 4.12: Reliability skip-ring network

4.3.2.3 Fault tree (FT)

Di seguito si riporta il fault tree della skip-ring network, nel quale si evidenzia come il fallimento di una qualsiasi coppia di nodi consecutivi porti al fallimento del sistema. Dal momento che il sistema non è di tipo serie-parallello, il FT presenta eventi ripetuti

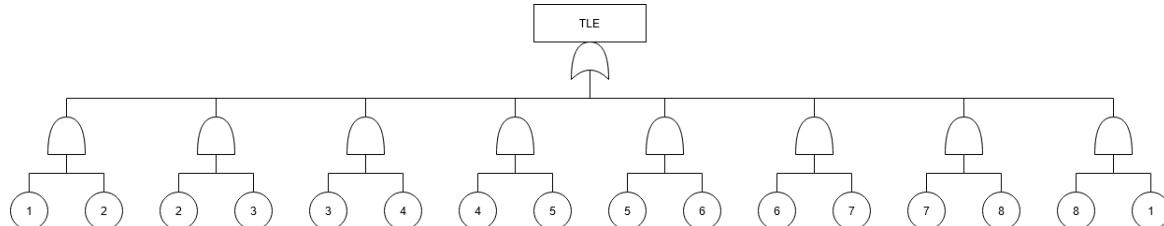


Figura 4.13: Fault tree skip-ring network

4.3.2.4 Reliability Block Diagram (RBD)

Il reliability block diagram del sistema è rappresentato dall'architettura stessa della rete, riportata in Figura 4.11.

Nel Paragrafo 4.3.2.1 si è visto come ottenere un'espressione precisa della reliability del sistema a partire dallo schema architettonico. Tuttavia, può essere più intuitivo ragionare sull'RBD ottenuto a partire dal FT in Figura 4.13. Tale diagramma rappresenta una forma serie-parallelo della skip ring network, in cui sono presenti nodi ripetuti.

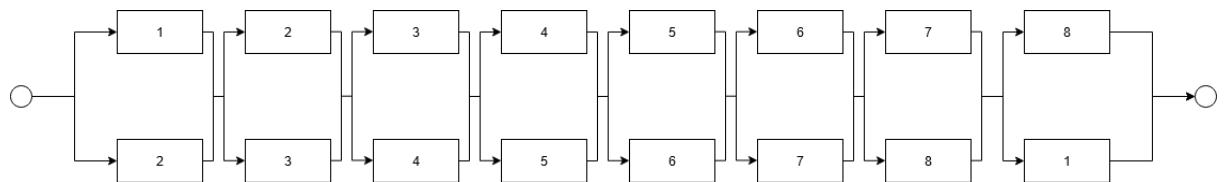


Figura 4.14: RBD serie-parallelo skip-ring network

4.4 Esercizio 4

4.4.1 Traccia

Comparare gli schemi in Figura 4.15, assumendo un'occorrenza esponenziale dei fallimenti con i seguenti valori:

- $MTTF_A = 1000 \text{ h}$
- $MTTF_B = 5000 \text{ h}$
- $MTTF_C = 2000 \text{ h}$

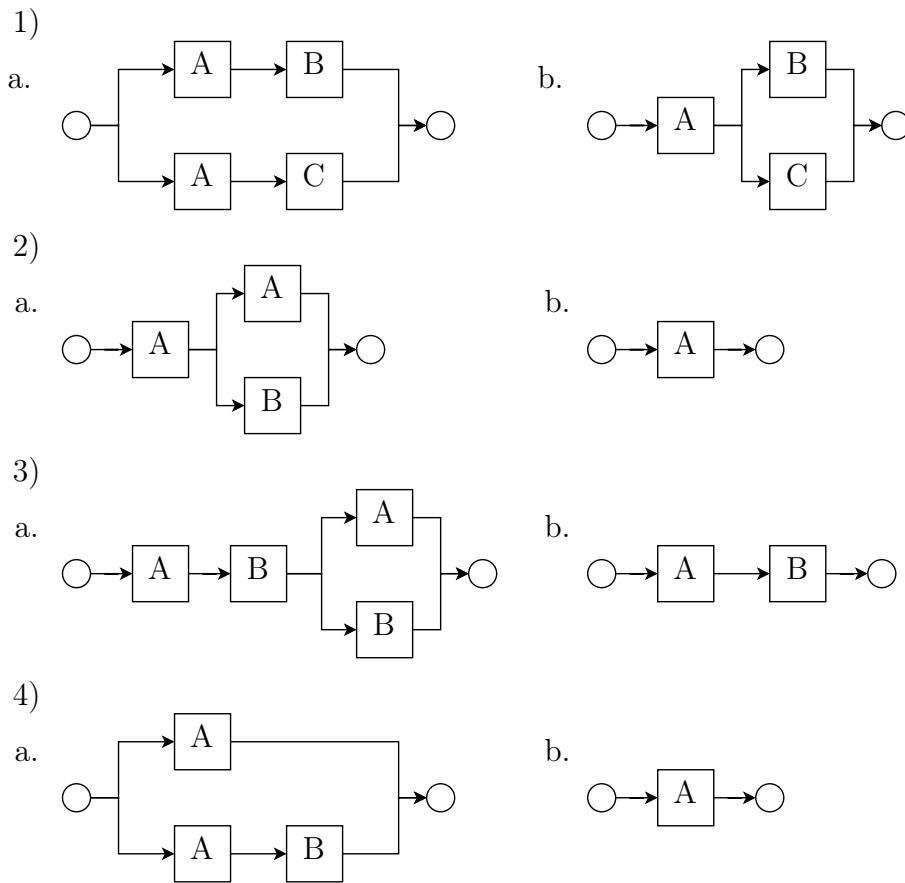


Figura 4.15: RBD

4.4.2 Soluzione

Per calcolare le reliability dei sistemi proposti è sufficiente applicare le formule per la risoluzione dei sistemi serie e parallelo.

4.4.2.1 Schema 1

La reliability del primo sistema può essere calcolata facilmente osservando che è costituito dal parallelo tra la serie dei componenti A e B e la serie dei componenti A e C.

$$\begin{aligned}
 R_{sys_1} &= 1 - (1 - R_A R_B)(1 - R_A R_C) \\
 &= 1 - (1 - e^{-\frac{1}{1000}t} e^{-\frac{1}{5000}t})(1 - e^{-\frac{1}{1000}t} e^{-\frac{1}{2000}t}) \\
 &= 1 - (1 - e^{-\frac{3}{25000}t})(1 - e^{-\frac{3}{2000}t}) = e^{-\frac{3}{2000}t} + e^{-\frac{3}{2500}t} - e^{-\frac{27}{10000}t}
 \end{aligned}$$

La reliability del secondo sistema può essere calcolata facilmente osservando che è costituito dalla serie tra il componente A e il parallelo tra i componenti B e C.

$$\begin{aligned}
 R_{sys_2} &= R_A[1 - (1 - R_B)(1 - R_C)] \\
 &= e^{-\frac{1}{1000}t}[1 - (1 - e^{-\frac{1}{5000}t})(1 - e^{-\frac{1}{2000}t})] \\
 &= e^{-\frac{1}{1000}t}[1 - (1 - e^{-\frac{1}{2000}t} - e^{-\frac{1}{5000}t} + e^{-\frac{7}{10000}t})] \\
 &= e^{-\frac{1}{1000}t}[e^{-\frac{1}{2000}t} + e^{-\frac{1}{5000}t} - e^{-\frac{7}{10000}t}] = e^{-\frac{3}{2000}t} + e^{-\frac{3}{2500}t} - e^{-\frac{17}{10000}t}
 \end{aligned}$$

Per confrontare le reliability dei due schemi è possibile tracciare il grafico delle funzioni $R_{sys_1}(t)$ ed $R_{sys_2}(t)$. Come è possibile osservare in Figura 4.16 lo schema 1 risulta migliore dello schema 2 in quanto $R_{sys_1}(t) > R_{sys_2}(t) \forall t$.

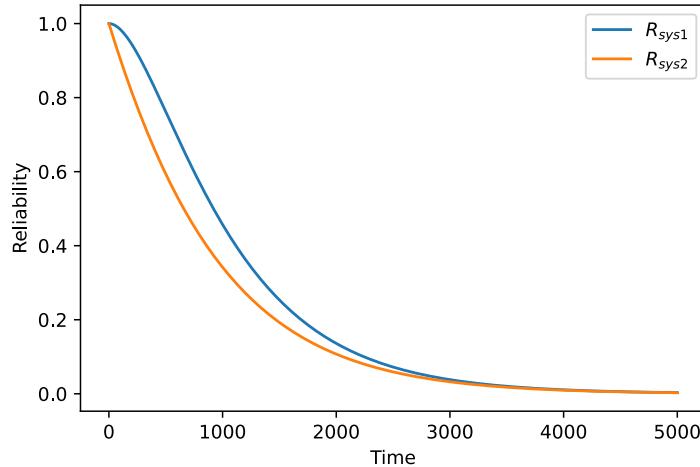


Figura 4.16: Confronto tra le funzioni $R_{sys_1}(t)$ ed $R_{sys_2}(t)$

4.4.2.2 Schema 2

La reliability del primo sistema può essere calcolata facilmente osservando che è costituito dalla serie tra il componente A e il parallelo tra i componenti A e B.

$$\begin{aligned}
 R_{sys_1} &= R_A(1 - (1 - R_A)(1 - R_B)) \\
 &= e^{-\frac{1}{1000}t}[1 - (1 - e^{-\frac{1}{1000}t})(1 - e^{-\frac{1}{500}t})] \\
 &= e^{-\frac{1}{1000}t}[1 - (1 + e^{-\frac{3}{2500}t} - e^{-\frac{1}{500}t} - e^{-\frac{1}{1000}t})] \\
 &= e^{-\frac{1}{1000}t}[-e^{-\frac{3}{2500}t} + e^{-\frac{1}{500}t} + e^{-\frac{1}{1000}t}] \\
 &= e^{-\frac{3}{2500}t} + e^{-\frac{1}{500}t} - e^{-\frac{11}{5000}t}
 \end{aligned}$$

La reliability del secondo sistema è semplicemente la reliability del componente A.

$$R_{sys_2} = R_A = e^{-\frac{1}{1000}t}$$

Per confrontare le reliability dei due schemi è possibile tracciare il grafico delle funzioni $R_{sys_1}(t)$ ed $R_{sys_2}(t)$. Come è possibile osservare in Figura 4.17 lo schema 2 risulta migliore dello schema 1 in quanto $R_{sys_2}(t) > R_{sys_1}(t) \forall t$.

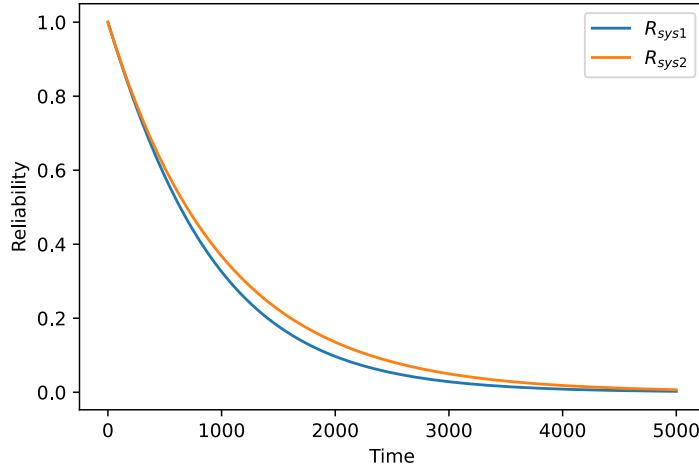


Figura 4.17: Confronto tra le funzioni $R_{sys_1}(t)$ ed $R_{sys_2}(t)$

4.4.2.3 Schema 3

La reliability del primo sistema può essere calcolata facilmente osservando che è costituito dalla serie tra i componenti A e B e il parallelo tra i componenti A e B.

$$\begin{aligned}
 R_{sys_1} &= R_A R_B (1 - (1 - R_A)(1 - R_B)) \\
 &= e^{-\frac{1}{1000}t} e^{-\frac{1}{5000}t} [1 - (1 - e^{-\frac{1}{1000}t})(1 - e^{-\frac{1}{5000}t})] \\
 &= e^{-\frac{3}{2500}t} [1 - (1 - e^{-\frac{1}{5000}t} - e^{-\frac{1}{1000}t} + e^{-\frac{3}{2500}t})] \\
 &= e^{-\frac{3}{2500}t} [e^{-\frac{1}{5000}t} + e^{-\frac{1}{1000}t} - e^{-\frac{3}{2500}t}] \\
 &= e^{-\frac{7}{5000}t} + e^{-\frac{11}{5000}t} - e^{-\frac{6}{2500}t}
 \end{aligned}$$

La reliability del secondo sistema può essere calcolata facilmente osservando che è costituito dalla serie tra i componenti A e B.

$$R_{sys_2} = R_A R_B = e^{-\frac{1}{1000}t} e^{-\frac{1}{5000}t} = e^{-\frac{3}{2500}t}$$

Per confrontare le reliability dei due schemi è possibile tracciare il grafico delle funzioni $R_{sys_1}(t)$ ed $R_{sys_2}(t)$. Come è possibile osservare in Figura 4.18 lo schema 2 risulta migliore dello schema 1 in quanto $R_{sys_2}(t) > R_{sys_1}(t) \forall t$.

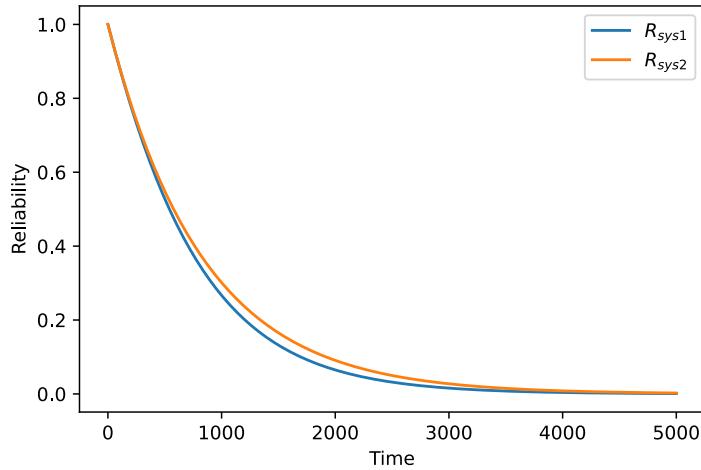


Figura 4.18: Confronto tra le funzioni $R_{sys_1}(t)$ ed $R_{sys_2}(t)$

4.4.2.4 Schema 4

La reliability del primo sistema può essere calcolata facilmente osservando che è costituito dal parallelo tra il componente A e la serie dei componenti A e B.

$$\begin{aligned}
 R_{sys_1} &= 1 - (1 - R_A)(1 - R_A R_B) \\
 &= 1 - (1 - e^{-\frac{1}{1000}t})(1 - e^{-\frac{1}{1000}t} e^{-\frac{1}{5000}t}) \\
 &= 1 - (1 - e^{-\frac{3}{2500}t} - e^{-\frac{1}{1000}t} + e^{-\frac{11}{5000}t}) \\
 &= e^{-\frac{3}{2500}t} + e^{-\frac{1}{1000}t} - e^{-\frac{11}{5000}t}
 \end{aligned}$$

La reliability del secondo sistema è semplicemente la reliability del componente A.

$$R_{sys_2} = R_A = e^{-\frac{1}{1000}t}$$

Per confrontare le reliability dei due schemi è possibile tracciare il grafico delle funzioni $R_{sys_1}(t)$ ed $R_{sys_2}(t)$. Come è possibile osservare in Figura 4.19 lo schema 1 risulta migliore dello schema 2 in quanto $R_{sys_1}(t) > R_{sys_2}(t) \forall t$.

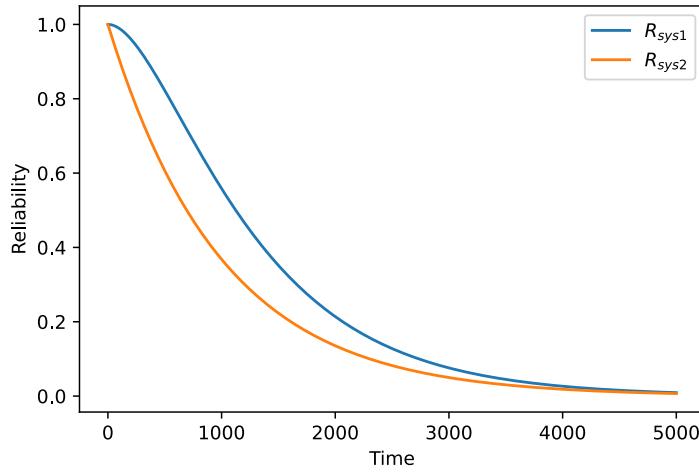


Figura 4.19: Confronto tra le funzioni $R_{sys1}(t)$ ed $R_{sys2}(t)$

4.5 Esercizio 5

4.5.1 Traccia

Il sistema in Figura 4.20 rappresenta un sistema di calcolo di un elicottero. Il sistema ha un processore dual-redundant e delle unità di interfaccia anch'esse dual-redundant. Due bus sono usati nel sistema e ciascuno di essi è dual-redundant. La parte interessante del sistema è l'equipaggiamento di navigazione. La navigazione del velivolo può essere completamente gestita utilizzando l'Inertial Navigation System (INS). Se l'INS fallisce, il velivolo può essere pilotato utilizzando la combinazione di Doppler e dell'Altitude Heading and Reference System (AHRS). Il sistema prevede tre unità AHRS, delle quali una sola è necessaria. Questo è un esempio di ridondanza funzionale dove i dati dell'AHRS e del Doppler possono essere utilizzati per rimpiazzare l'INS, se l'INS fallisce. A causa degli altri sensori e della strumentazione, entrambi i bus sono necessari per il corretto funzionamento del sistema, a prescindere da quale modalità di navigazione è impiegata.

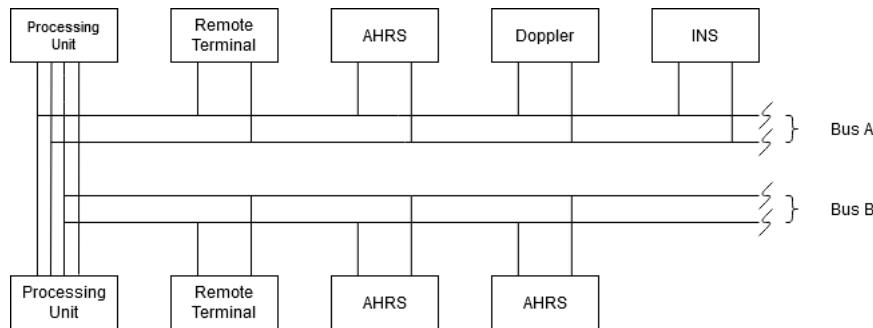


Figura 4.20: Architettura del sistema

1. Disegnare il Reliability Block Diagram del sistema
2. Disegnare il *Fault Tree* (FT) del sistema ed analizzarne i minimal cutsets
3. Calcolare la reliability per un'ora di volo utilizzando i valori di MTTF dati in Tabella 4.1. Si assuma che sia valida una legge di fallimento esponenziale e che la fault coverage sia perfetta.

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

Tabella 4.1: MTTF dei componenti del sistema

4. Ripetere l'analisi del punto precedente, considerando un fattore di coverage per la fault detection e la riconfigurazione delle processing units. Utilizzando gli stessi valori di MTTF, determinare approssimativamente il valore di fault coverage necessario per ottenere una reliability (alla fine di un'ora) di 0.99999.

4.5.2 Soluzione

4.5.2.1 Reliability Block Diagram (RBD)

Dalle indicazioni fornite nella traccia, è facile comprendere come il reliability block diagram del sistema sia dato dalla serie di diversi macroblocchi.

In particolare, osservando la Figura 4.21 si evidenziano i seguenti blocchi:

- *Blocco Bus A*: composto dal parallelo di due componenti Bus A, essendo quest'ultimo utilizzato in una configurazione dual redundant.
- *Blocco Bus B*: composto dal parallelo di due componenti Bus B, essendo quest'ultimo utilizzato in una configurazione dual redundant. I blocchi relativi al Bus A ed al Bus B sono posti in serie dal momento che per funzionare correttamente, il sistema necessita di entrambi.
- *Blocco Processing Unit*: composto dal parallelo di due componenti Processing Unit, essendo quest'ultimo utilizzato in una configurazione dual redundant.
- *Blocco Remote Terminal*: composto dal parallelo di due componenti Remote Terminal, essendo quest'ultimo utilizzato in una configurazione dual redundant.
- *Blocco sistema di navigazione*: composto a sua volta dai blocchi:
 - *Blocco sistema di navigazione inerziale (INS)*: composto da un singolo componente INS.

- *Blocco sistema di navigazione ausiliario:* composto dalla serie tra un componente Doppler ed il parallelo di tre componenti AHRS. Tale serie è giustificata dal fatto che il sistema, per funzionare correttamente, necessita sia del componente Doppler che di almeno un componente AHRS.

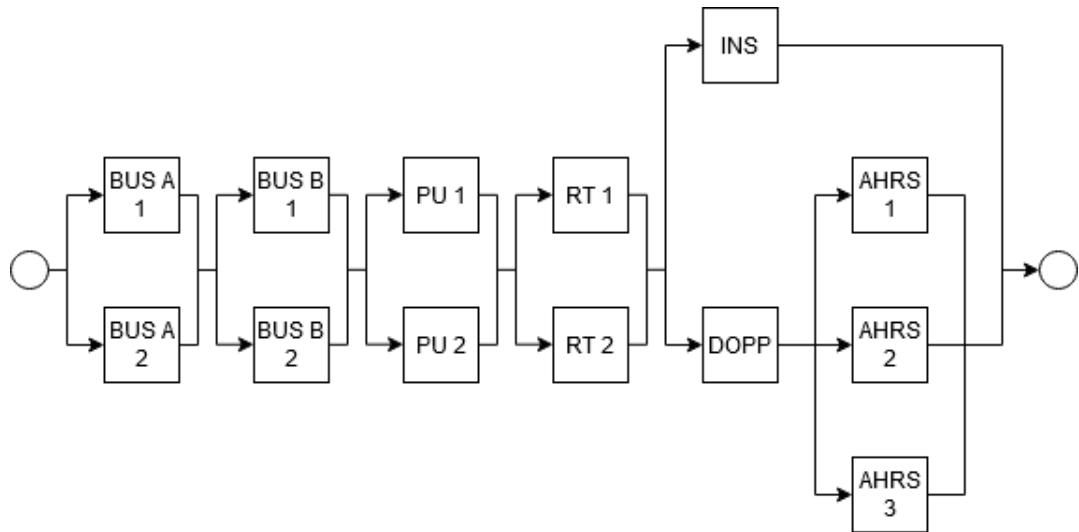


Figura 4.21: RBD

4.5.2.2 Fault Tree (FT)

Partendo dall'RBD del sistema, risulta particolarmente semplice ottenere il Fault Tree. In particolare:

- I componenti o i sottosistemi connessi in serie nell'RBD vengono connessi tra loro tramite porte OR nel Fault Tree.
- I componenti o i sottosistemi in parallelo nell'RBD vengono connessi tra loro tramite porte AND nel Fault Tree.

Seguendo queste semplici regole è possibile ricavare il Fault Tree in Figura 4.22.

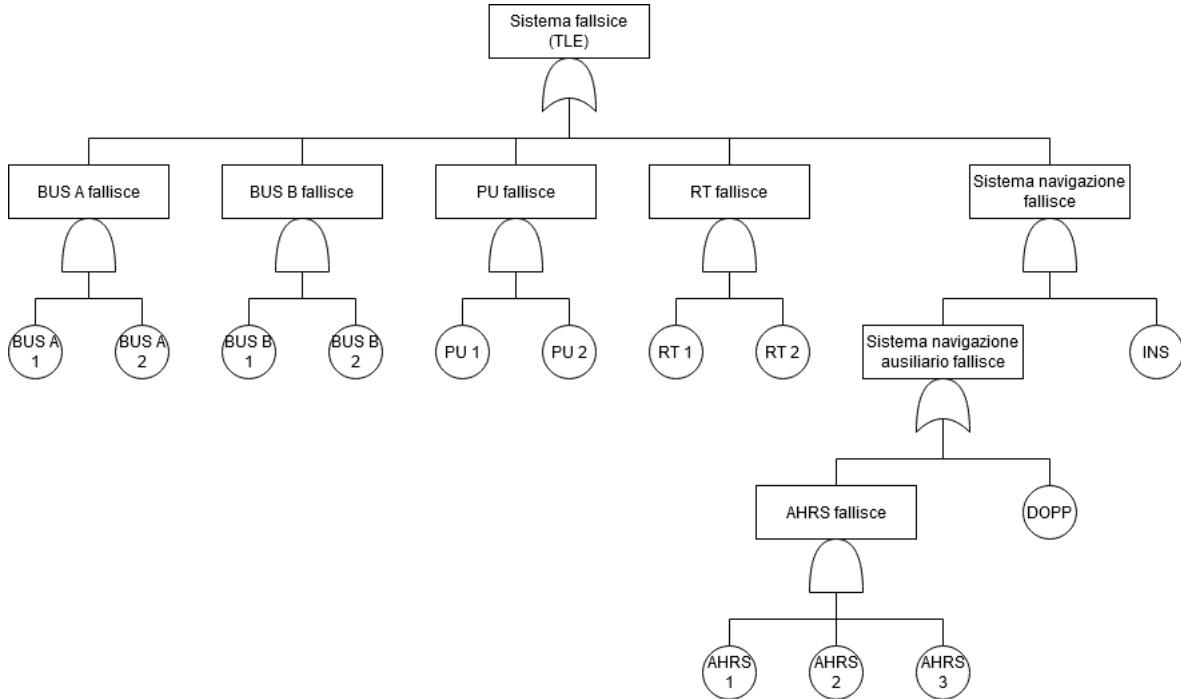


Figura 4.22: Fault Tree

Un **cut-set** in un FT è un insieme di **basic event** i quali, se occorrono simultaneamente, fanno sì che occorra anche il **top level event**, ovvero il fallimento del sistema. Un cut-set è detto **minimal cut-set** se non può essere ridotto senza perdere il suo status di cut-set.

Al fine di evidenziare i minimal cut-set del sistema, è utile esprimere il Fault Tree in forma analitica tramite la sua **funzione di struttura** ed esprimerla in forma normale disgiuntiva:

$$\begin{aligned}
 \phi = & (BUS_{A_1} \cdot BUS_{A_2}) + (BUS_{B_1} \cdot BUS_{B_2}) + (PU_1 \cdot PU_2) + (RT_1 \cdot RT_2) + \\
 & (AHRS_1 \cdot AHRS_2 \cdot AHRS_3 + DOPP) \cdot INS \\
 = & (BUS_{A_1} \cdot BUS_{A_2}) + (BUS_{B_1} \cdot BUS_{B_2}) + (PU_1 \cdot PU_2) + (RT_1 \cdot RT_2) + \\
 & (AHRS_1 \cdot AHRS_2 \cdot AHRS_3 \cdot INS) + (DOPP \cdot INS)
 \end{aligned}$$

A questo punto ogni mintermine rappresenta un minimal cut-set per il Fault Tree:

$$MCS_1 = \{BUS_{A_1}, BUS_{A_2}\}$$

$$MCS_2 = \{BUS_{B_1}, BUS_{B_2}\}$$

$$MCS_3 = \{PU_1, PU_2\}$$

$$MCS_4 = \{RT_1, RT_2\}$$

$$MCS_5 = \{AHRS_1, AHRS_2, AHRS_3, INS\}$$

$$MCS_6 = \{DOPP, INS\}$$

4.5.2.3 Reliability del sistema

La funzione di reliability del sistema si può ricavare a partire dalla struttura dell'RBD oppure, in maniera equivalente, a partire dalla funzione di struttura. In particolare, osservando la funzione di struttura si ottiene:

$$\begin{aligned} R_{sys}(t) &= [1 - (1 - R_{BUS})^2][1 - (1 - R_{BUS})^2][1 - (1 - R_{PU})^2][1 - (1 - R_{RT})^2] \cdot \\ &\quad \cdot [1 - (1 - R_{INS})(1 - R_{AHRS})^3][1 - (1 - R_{INS})(1 - R_{DOPP})] = \\ &= [1 - (1 - e^{-\frac{1}{60000}t})^2][1 - (1 - e^{-\frac{1}{60000}t})^2][1 - (1 - e^{-\frac{1}{10000}t})^2][1 - (1 - e^{-\frac{1}{4500}t})^2] \cdot \\ &\quad \cdot [1 - (1 - e^{-\frac{1}{2000}t})(1 - e^{-\frac{1}{2000}t})^3][1 - (1 - e^{-\frac{1}{2000}t})(1 - e^{-\frac{1}{500}t})] \end{aligned}$$

Valutando la funzione $R_{sys}(t)$ per $t = 1$ si ottiene:

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

Infine, in Figura 4.23 è riportato l'andamento della funzione di reliability del sistema al variare del tempo.

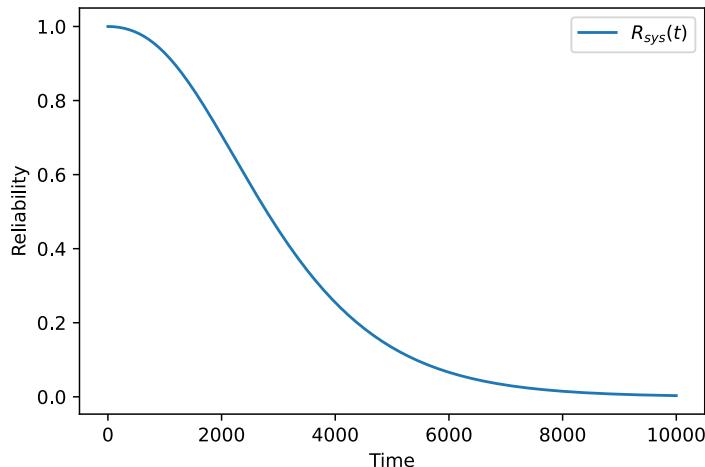


Figura 4.23: Reliability del sistema

4.5.2.4 Reliability con fattore di coverage

Nella sezione precedente si è supposto che un componente fallito sia in grado di rilevare il proprio fallimento e di disabilitarsi di conseguenza. Adesso si considera il caso in cui la fault detection non è perfetta e la riconfigurazione del componente fallito potrebbe non avere successo. È quindi necessario considerare un **fattore di coverage** per il componente processing unit. Tale fattore rappresenta proprio la probabilità che il componente fallito riconosca il proprio fallimento e si riconfiguri in maniera opportuna, in modo da non portare al fallimento del sistema.

Introducendo un fattore di coverage c per le processing unit, l'unico termine da modificare nell'espressione della reliability del sistema è proprio quello relativo al parallelo

delle due processing unit. La reliability del parallelo è data quindi dalla probabilità che una replica funzioni più la probabilità che tale replica fallisca per la probabilità di rilevare il fallimento, moltiplicato la probabilità che l'altra replica funzioni.

$$[R_{PU} + c(1 - R_{PU})R_{PU}]$$

Riscrivendo l'espressione della reliability del sistema si ottiene quindi:

$$R_{sys}(t) = [1 - (1 - R_{BUS})^2]^2 [R_{PU} + c(1 - R_{PU})R_{PU}] [1 - (1 - R_{RT})^2] \cdot [1 - (1 - R_{INS})(1 - R_{AHRS})^3] [1 - (1 - R_{INS})(1 - R_{DOPP})] \quad (4.5)$$

Al fine di ottenere il valore di c tale per cui $R_{sys}(1) \geq 0.99999$ è sufficiente risolvere l'Equazione 4.5 valutata in $t = 1$ rispetto all'unica variabile c . Svolgendo i calcoli si ottiene la seguente condizione:

$$c \geq 0.910574$$

Capitolo 5

Field Failure Data Analysis

5.1 Traccia

Condurre una *log based Field Failure Data Analysis* (FFDA) per due sistemi *large-scaled*:

- *Mercury*, sistema del National Center for Supercomputing Application (NCSA) alla University of Illinois
- *Blue Gene/L*, sistema del Lawrence Livermore National Labs (LLNL).

5.2 Field Failure Data Analysis (FFDA)

La **Field Failure Data Analysis** (FFDA) è una tecnica di misurazione diretta volta a misurare attributi di dependability di un sistema in esecuzione in un ambiente di lavoro reale. In altre parole, consiste nell'analisi dei dati relativi ai fallimenti di un sistema, dove i dati sono raccolti sul campo. Tipicamente tale tecnica prevede il susseguirsi delle seguenti fasi:

- **Data loggin o data collection**: vengono raccolti dati grezzi relativi al funzionamento del sistema.
- **Data filtering**: vengono filtrati i dati raccolti nella fase precedente, al fine di ridurre la quantità di dati da analizzare, concentrando l'attenzione solo sui dati significativi.
- **Data manipulation**: vengono manipolati i dati filtrati nella fase precedente, al fine di tracciare una corrispondenza tra eventi raccolti ed eventuali fallimenti del sistema.
- **Data analysis**: nell'ultima fase vengono analizzati i dati processati nelle fasi precedenti. Gli obiettivi dell'analisi possono essere diversi, dalla *failure classification* all'estrazione della distribuzione del *time to failure* e della *reliability* del sistema.

Nel contesto dell'analisi dei sistemi Mercury e Blue Gene/L si interverrà a valle della fase di data filering, svolgendo di fatto solo le ultime due fasi.

5.3 Mercury

Mercury è un supercomputer costituito da nodi di elaborazione IBM. Come riportato in Figura 5.1, l'architettura del sistema è divisa in 3 livelli, *login*, *computation* e *storage*, ognuno dei quali prevede un certo numero di nodi. In aggiunta a questi tre tipi di nodi, è presente un nodo di gestione detto *master*.

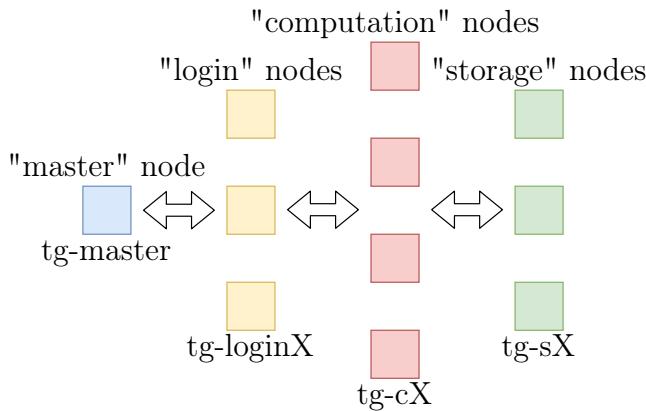


Figura 5.1: Architettura Mercury

I dati da analizzare relativi a Mercury, contenuti in un file di log chiamato *MercuryErrorLog.txt*, sono stati raccolti da un processo demone *syslog*, tipico dei sistemi Linux. Il file contiene 80854 entry, ognuna delle quali prevede i seguenti campi:

- *timestamp*: intero in formato UNIX;
- *nodo di origine*: stringa del tipo *tg-x* indicante il nodo in cui si è verificato l'errore;
- *sottosistema di origine*: stringa indicante il sottosistema da cui ha origine l'errore. Esistono 5 sottosistemi differenti, DEV, MEM, I-O, NET e PRO, e ogni nodo può contenere più sottosistemi.
- *messaggio testuale*: stringa indicante il messaggio relativo all'errore.

All'interno del file di log sono riportate tutte e sole le entry relative ai *fatal error*. In altre parole, è stata effettuata una fase di data filtering preliminare.

5.3.1 Sistema

In questa sezione si riportano le ultime due fasi della FFDA, eseguite considerando i dati relativi all'intero sistema Mercury.

5.3.1.1 Data Manipulation

Uno degli obiettivi principali della fase di data manipulation è l'identificazione dei fallimenti del sistema. A partire dal log filtrato, contenente una serie di errori, si vuole

estrarre una relazione tra errori e fallimenti del sistema. Bisogna infatti tenere presente che non esiste un rapporto uno ad uno tra entry del file di log e fallimenti del sistema.

Per identificare i fallimenti del sistema è possibile utilizzare un serie di tecniche note come tecniche di **coalescenza**, in grado appunto di individuare la correlazione esistente tra le entry del log.

5.3.1.1.1 Coalescenza spaziale La *coalescenza spaziale* si pone l'obiettivo di capire se esiste una correlazione temporale tra le entry del dataset. Si parla di coalescenza spaziale, in quanto si considerano entry provenienti da diversi nodi. La tecnica prevede di raggruppare le entry che distano tra loro meno di una determinata soglia. I gruppi di entry sufficientemente vicine sono detti **tuple**.

Di seguito si riporta lo pseudo-codice relativo all'algoritmo di tupling:

Algoritmo di tupling

```
if  $t(x_i + 1) - t(x_i) < W$  then
    add  $x_i$  to the current tuple
end if
```

dove x_i è l'i-esima entry del log, $t(x_i)$ è il timestamp di x_i e W è la soglia temporale fissata, detta **finestra di coalescenza**.

La scelta della finestra influenza notevolmente i risultati dell'algoritmo:

- se la finestra è troppo grande si verificano **collisioni**: eventi relativi a guasti diversi vengono inseriti nella stessa tupla;
- se la finestra è troppo piccola si verificano **troncamenti**: eventi relativi allo stesso guasto vengono inseriti in tuple diverse.

Dal punto di vista della reliability empirica del sistema, il verificarsi di collisioni è più critico rispetto al verificarsi di troncamenti. Se si hanno collisioni la stima effettuata sarà infatti ottimistica, mentre se si hanno troncamenti la stima sarà pessimistica.

5.3.1.1.1.1 Sensitivity Analysis Per scegliere opportunamente la dimensione della finestra di coalescenza si esegue la *sensitivity analysis*. Per prima cosa si calcola il numero di tuple ottenute applicando l'algoritmo di tupling al variare di W .

Sensitivity Analysis

```
def sensitivity_analysis(dataframe, c_wins, filepath):
    if(not os.path.isfile(filepath)):
        print('Creating file',filepath, 'this may take a while...\n')
        tuples = []
    for win in c_wins:
        tup = 1
        prev = int(dataframe.loc[0,'Timestamp'])
        for index in range(dataframe.shape[0]-1):
            succ = int(dataframe.loc[index+1,'Timestamp'])
            if succ - prev >= win:
                tup = tup + 1
                prev = succ
        tuples.append(tup)
    return tuples
```

```

df_sens = pd.DataFrame(data = {'Window' : c_wins, 'Tuples' : tuples})
df_sens.to_csv(filepath)

else:
    print('File already exists!\n')
    df_sens = pd.read_csv(filepath, index_col = 0)

return df_sens

```

Una volta eseguito il tupling al variare della finestra di coalescenza, si riporta su un grafico l'andamento del numero di tuple in funzione di W.

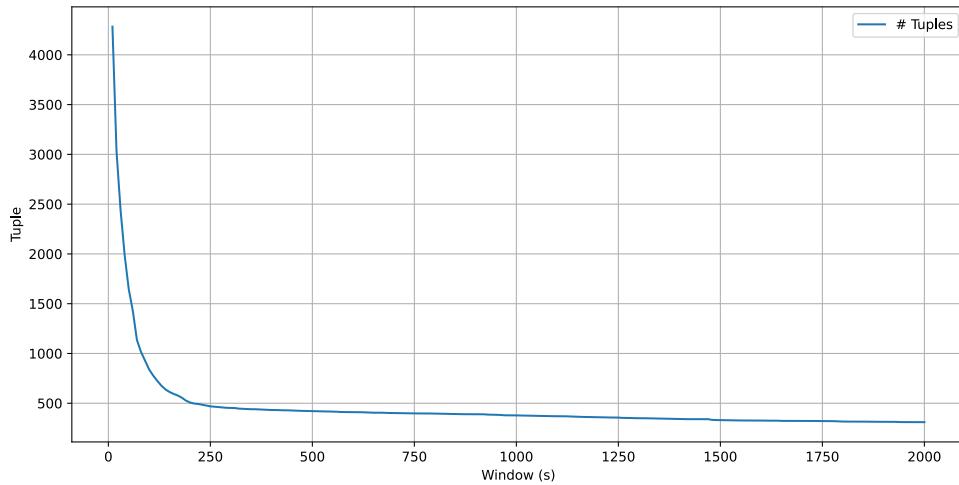


Figura 5.2: Sensitivity Analysis Mercury

Il valore ottimo della dimensione di W è quello immediatamente seguente al gomito della curva. In questo caso, si sceglie come dimensione della finestra di coalescenza un valore pari a 250 secondi (circa 4 minuti).

5.3.1.1.1.2 Tupling Individuato il valore ottimale di W, è possibile eseguire il tupling, andando stavolta a memorizzare la tupla di appartenenza associata ad ogni entry del log. Eseguendo l'algoritmo di tupling riportato di seguito si ottengono 469 tuple totali.

Sensitivity Analysis

```

def tupling(dataframe, c_win, filepath):
    if(not os.path.isfile(filepath)):
        print('Creating file', filepath, 'this may take a while...\n')

    tup = 0
    prev = int(dataframe.loc[0,'Timestamp'])
    dataframe.loc[0,'Tuple'] = 0

    for index in range(dataframe.shape[0]-1):
        succ = int(dataframe.loc[index+1,'Timestamp'])
        if succ - prev >= c_win:
            tup = tup + 1
            dataframe.loc[index+1,'Tuple'] = tup
        prev = succ

```

```

    dataframe.to_csv(filepath)

else:
    print('File already exists!\n')
    dataframe = pd.read_csv(filepath, index_col = 0)

return dataframe

```

Dopo aver determinato le tuple è possibile studiarne alcune statistiche. In particolare, per ogni tupla vengono determinate le seguenti informazioni:

- *# entries*: numero di entry del log appartenenti alla tupla;
- *start time*: timestamp della prima entry appartenente alla tupla;
- *end time*: timestamp dell'ultima entry appartenente alla tupla;
- *length*: differenza tra il timestamp dell'ultima entry e quello della prima entry della tupla;
- *density*: rapporto tra numero di entry e lunghezza della tupla;
- *interarrival*: tempo (in secondi) che intercorre tra la prima entry della tupla corrente e l'ultima entry della tupla precedente;

Calcolo Statistiche

```

def statistics(dataframe):
    df_stat = pd.DataFrame(columns=['# Entries', 'Start time', 'End time',
                                    'Length', 'Density', 'Interarrival'])

    df_stat['# Entries'] = dataframe.groupby(by='Tuple').count()['Timestamp']
    df_stat['Start time'] = dataframe.groupby(by='Tuple')
        .min(numeric_only = True)['Timestamp']
    df_stat['End time'] = dataframe.groupby(by='Tuple')
        .max(numeric_only = True)['Timestamp']
    df_stat['Length'] = df_stat['End time'] - df_stat['Start time']
    df_stat['Density'] = df_stat['# Entries'] / df_stat['Length']
    df_temp = pd.DataFrame(data = {'Start time' : [np.nan]})
    df_temp = df_temp.append(df_stat, ignore_index = True)
    df_stat['Interarrival'] = (df_stat['Start time'] - df_temp['End time'])
        .dropna().astype(int)

    return df_stat

```

5.3.1.1.3 Bottleneck In Figura 5.3 si riporta il numero di entry del log contenute in ogni tupla. Dal grafico è possibile osservare che un significativo numero di entry del log è contenuto in un numero limitato di tuple consecutive. Questo significa che molti eventi di log si sono verificati in un determinato arco temporale. Nello specifico, nell'arco temporale che va dallo start time della tupla 281 all'end time della tupla 306 si sono verificati 62836 eventi di log, che rappresentano circa il 78% di tutti gli eventi di log.

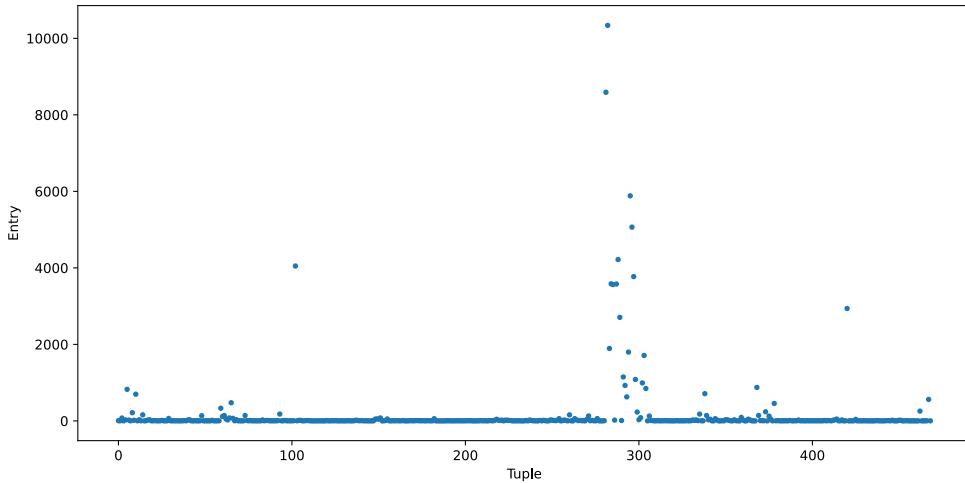


Figura 5.3: Tupling Mercury

Convertendo i timestamp UNIX relativi a start time ed end time rispettivamente delle tuple 281 e 306, si ottiene che l'arco temporale di interesse è quello che va dal lunedì 12 febbraio 2007 ore 22:58:33 al mercoledì 14 febbraio 2007 ore 20:44:37. Questo significa che il 78% degli errori si sono manifestati in soli tre giorni, il che è rilevante se si considera che l'arco temporale complessivo che va dal timestamp della prima entry del log (lunedì 01/01/2007 ore 07:47:40) a quello dell'ultima entry del log (lunedì 26/03/2007 ore 21:19:03) è della durata di 84 giorni.

Andando a studiare ulteriormente gli eventi caduti in questo arco temporale è possibile osservare che 62263 eventi, ovvero il 99%, si manifestano in un unico nodo: *tg-401*.

In conclusione, il 77% di tutti gli errori presenti nel file di log sono avvenuti tra il 12/02/2007 e il 14/02/2007 sul nodo *tg-401*, il quale rappresenta probabilmente il **bottleneck** del sistema.

5.3.1.1.2 Analisi dei troncamenti

Come detto in precedenza la scelta della finestra di coalescenza influisce sul numero di troncamenti e collisioni commessi a valle del tupling.

Si vuole quindi effettuare una stima dei troncamenti che commessi. Per prima cosa si calcola il valore q corrispondente al 10° percentile (0.1 quantile) dei tempi di interarrivo. Tale valore è pari a 491 secondi (circa 8 minuti), circa il doppio della finestra di coalescenza utilizzata. Si individuano poi tutte le coppie di tuple adiacenti che distano meno di questo valore. Per ognuna di queste coppie è possibile stimare un troncamento. Da questa prima stima si ottengono 47 troncamenti, il che significa che per circa il 10% delle tuple totali potrebbe essersi verificato un troncamento.

Una stima potenzialmente più precisa può essere effettuata considerando tra le 47 tuple precedentemente individuate solo quelle per cui il nodo di origine più frequente è lo stesso di quello della tupla precedente. Da questa ulteriore stima sono ottenuti 38 troncamenti, pari circa all'8% delle tuple totali.

5.3.1.1.3 Analisi delle collisioni Analogamente si vuole effettuare una stima delle collisioni commesse a causa del tupling. Il numero di potenziali collisioni è ottenuto individuando il numero di tuple per le quali si hanno più nodi di origine. Sono quindi stimate 51 collisioni, il che significa che per circa l'11% delle tuple totali potrebbero essersi verificate collisioni.

5.3.1.2 Data Analysis

Terminata la fase di data manipulation è possibile proseguire con l'analisi dei dati vera e propria. L'obiettivo di questa fase è quello di estrarre dai dati alcune informazioni relative alla reliability del sistema.

5.3.1.2.1 Empirical Time To Failure (TTF) Per prima cosa è possibile estrarre dai dati la distribuzione di probabilità empirica del time to failure, ovvero del tempo oltre il quale si verifica un fallimento del sistema. A tal fine, si applicano ai dati relativi ai tempi interarrivo una serie di test di **Goodness of Fit** (GoF), test statistici che verificano la presenza di una distribuzione nota nei dati. Per effettuare tali test sono stati utilizzati i metodi offerti dalla libreria *Reliability* [3].

In Figura 5.4 si riportano i risultati del fitting effettuato per tutte le distribuzioni previste dalla libreria. La bontà del fitting delle diverse distribuzioni è dato dall'ordine con cui esse appaiono nella legenda.

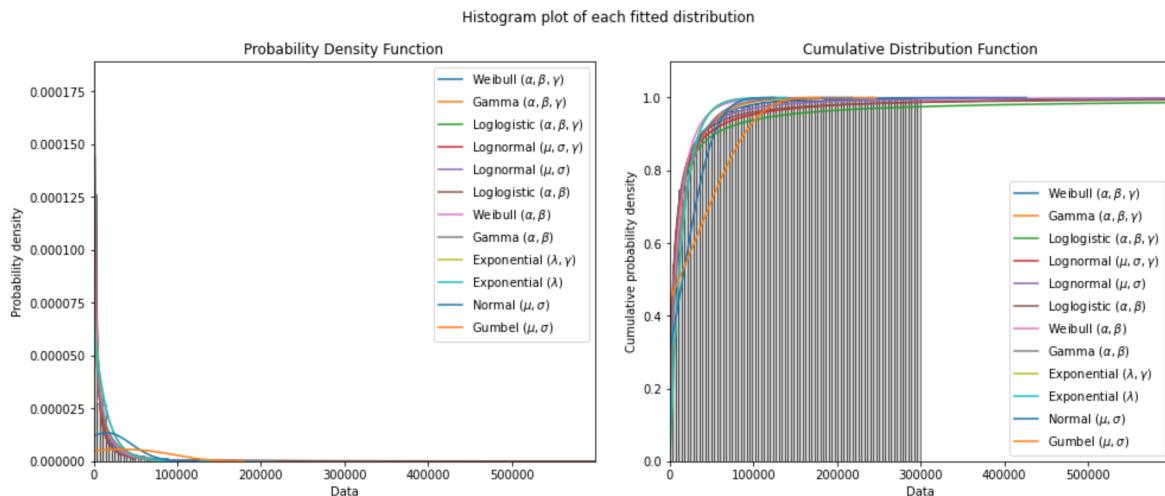


Figura 5.4: Test di fitting

Come si evince dai grafici precedenti, la distribuzione che meglio approssima i dati del sistema è una *Weibull* caratterizzata da 3 parametri.

Si riportano in Figura 5.5 i risultati ottenuti effettuando il fitting mirato della distribuzione Weibull a 3 parametri, i quali assumono i valori in Tabella 5.1.

α	β	γ
9306.84	0.580581	253

Tabella 5.1: Parametri Weibull

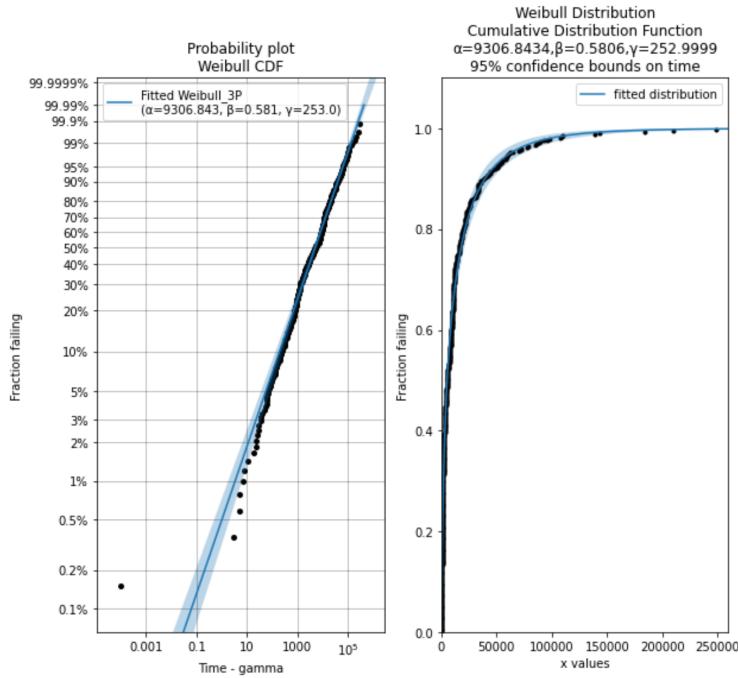


Figura 5.5: Test di fitting con distribuzione Weibull

Per avere un'ulteriore conferma dei risultati ottenuti dai test precedenti, è possibile effettuare un ulteriore test di GoF non parametrico, il test di **Kolmogorov-Smirnov**.

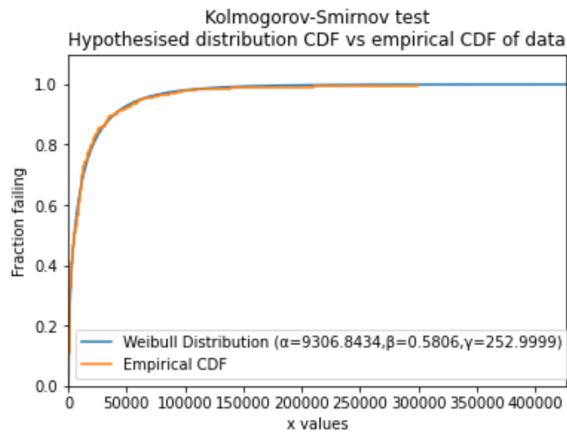


Figura 5.6: Test di Kolmogorov-Smirnov

Effettuato il test è possibile affermare con un livello di significatività di 0.05 che è possibile accettare l'ipotesi che i dati provengono da una distribuzione Weibull avente i parametri riportati in precedenza.

È bene osservare che in generale le distribuzioni con tre parametri, in particolare Gamma, Weibull e Loglogistica, rappresentano una famiglia di distribuzioni e pertanto riescono ad adattarsi bene a molte distribuzioni empiriche. Tale tipologia di distribuzioni infatti riesce ad approssimare molto bene la distribuzione empirica in esame e consente di realizzare analisi predittive sui dati in maniera accurata.

D'altro canto, distribuzioni più semplici come ad esempio weibull e lognormale con due parametri oppure la distribuzione esponenziale con un parametro, nel contesto della FFDA riescono a fornire un'indicazione sulla natura dei fallimenti.

A supporto dei test precedenti, è stato inoltre eseguito il test di Kolmogorov-Smirnov per le distribuzioni lognormale, weibull con due parametri ed esponenziale.

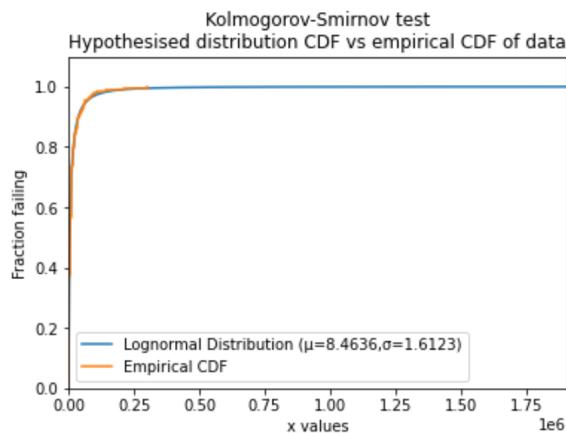


Figura 5.7: Test di Kolmogorov-Smirnov distribuzione Lognormale

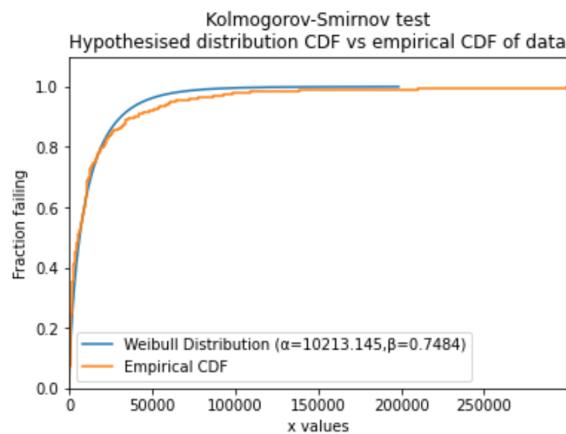


Figura 5.8: Test di Kolmogorov-Smirnov distribuzione Weibull

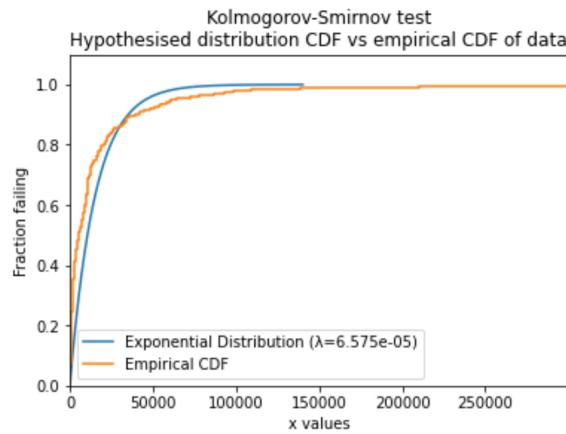


Figura 5.9: Test di Kolmogorov-Smirnov distribuzione Esponenziale

I risultati del test riportati nelle Figure 5.7, 5.8 e 5.9 rigettano la provenienza della distribuzione empirica dalle distribuzioni note con un livello di significatività di 0.05.

Da una prima analisi quindi non è possibile determinare la natura dei fallimenti che coinvolgono il sistema Mercury.

5.3.1.2.2 Empirical Reliability Un'informazione fondamentale che è possibile ricavare dai dati è la reliability empirica del sistema. A tal fine è sufficiente calcolare la CDF empirica dei tempi interarrivo, la quale rappresenta la *unreliability* del sistema, ovvero la probabilità che il sistema fallisca entro un certo istante. Ottenuta la CDF si può quindi ricavare la reliability empirica del sistema semplicemente come 1-CDF. Per calcolare la CDF empirica sono stati utilizzati i metodi forniti dalla libreria *statsmodels* [4].

Di seguito si riporta in Figura 5.10 l'andamento della CDF e della reliability empirica del sistema.

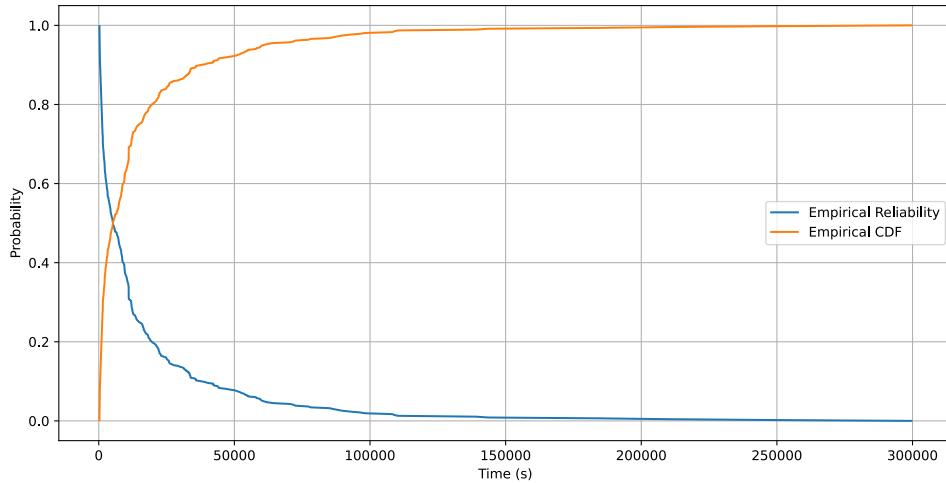


Figura 5.10: Reliability e CDF empirica

5.3.2 Sottosistemi

Nella sezione corrente si ripetono le stesse operazioni eseguite in precedenza sull'intero sistema, andando però a considerare i dati relativi ad ogni singolo sottosistema. Si vanno quindi a suddividere le diverse entry del log a seconda del sottosistema di origine.

In Figura 5.11 si riporta il numero entry presenti nel file di log per ciascun sottosistema. Come si evince dal grafico, il sottosistema più critico è DEV.

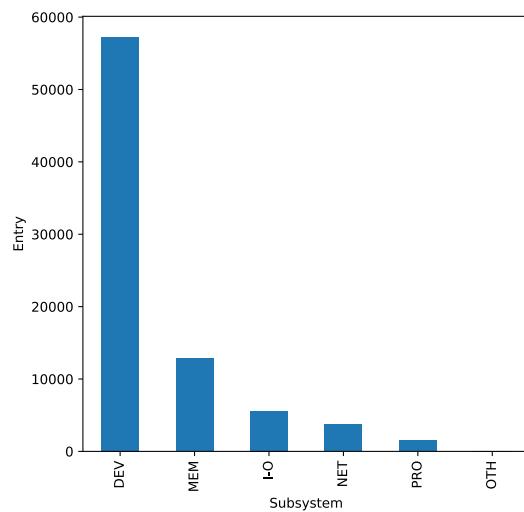


Figura 5.11: Numero di entry per sottosistema

5.3.2.1 Data Manipulation

5.3.2.1.1 Coalescenza spaziale

5.3.2.1.1.1 Sensitivity Analysis Di seguito si propone la sensitivity analysis effettuata per ciascun sottosistema.

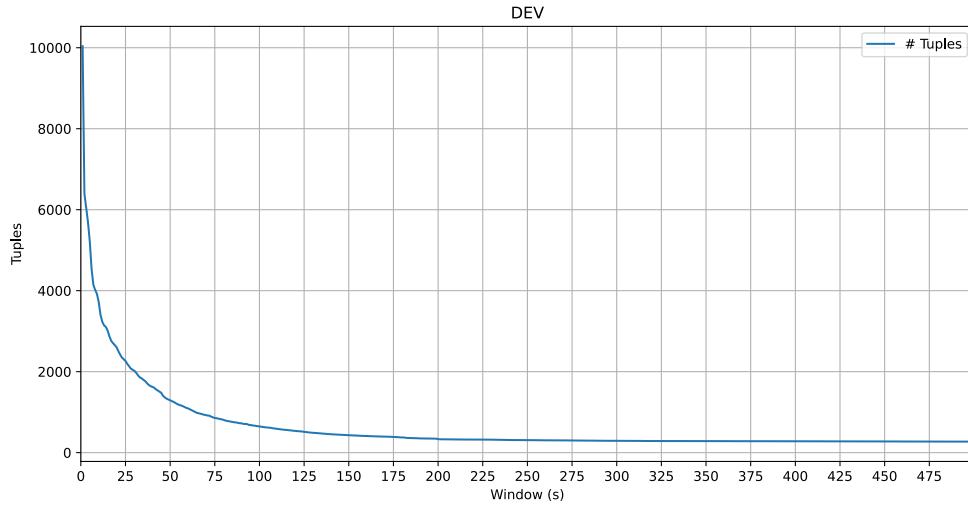


Figura 5.12: Sensitivity Analysis sottosistema DEV

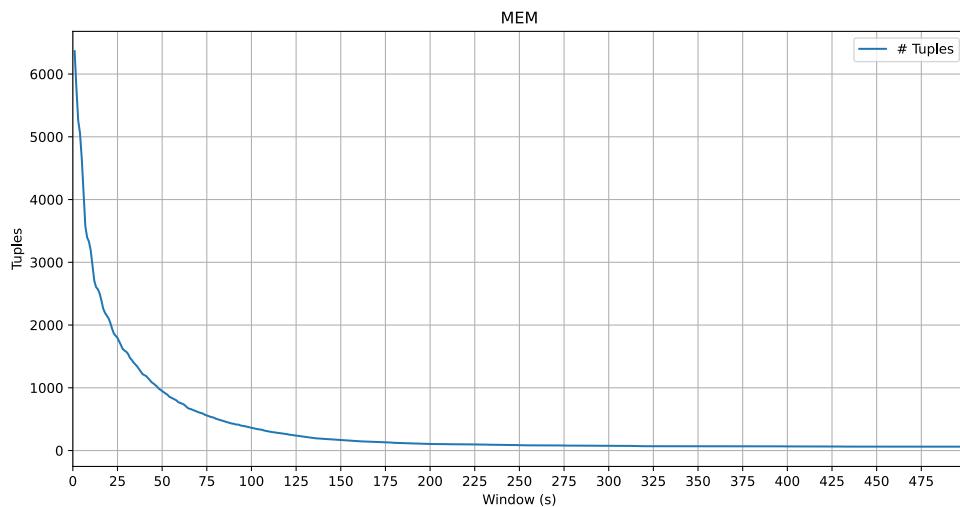


Figura 5.13: Sensitivity Analysis sottosistema MEM

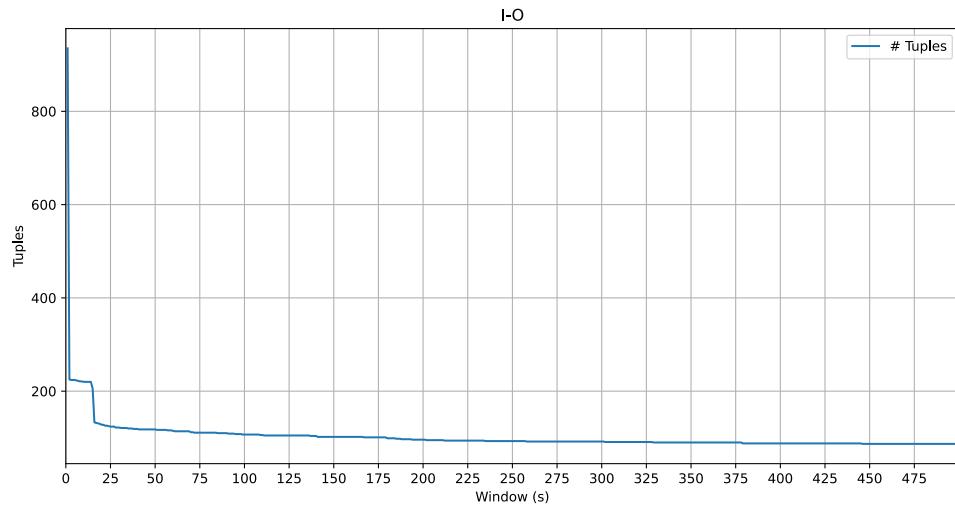


Figura 5.14: Sensitivity Analysis sottosistema I/O

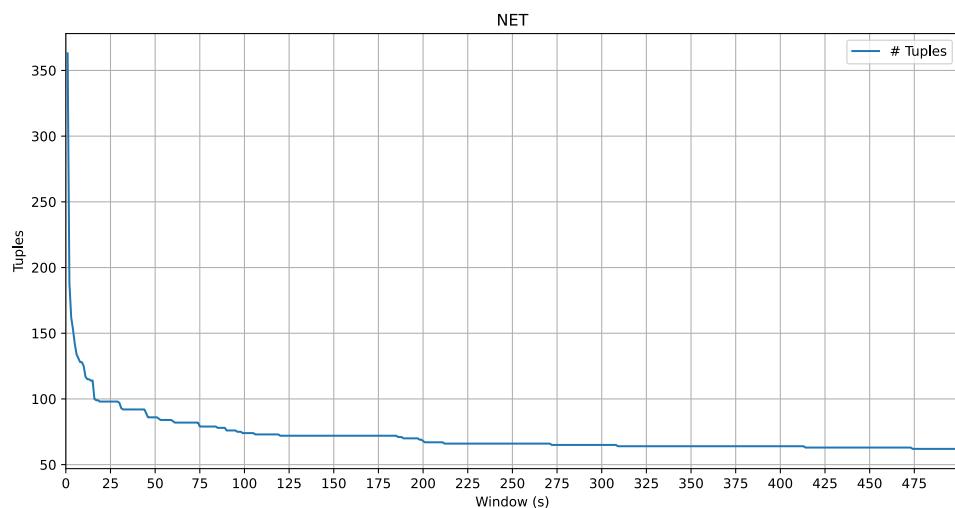


Figura 5.15: Sensitivity Analysis sottosistema NET

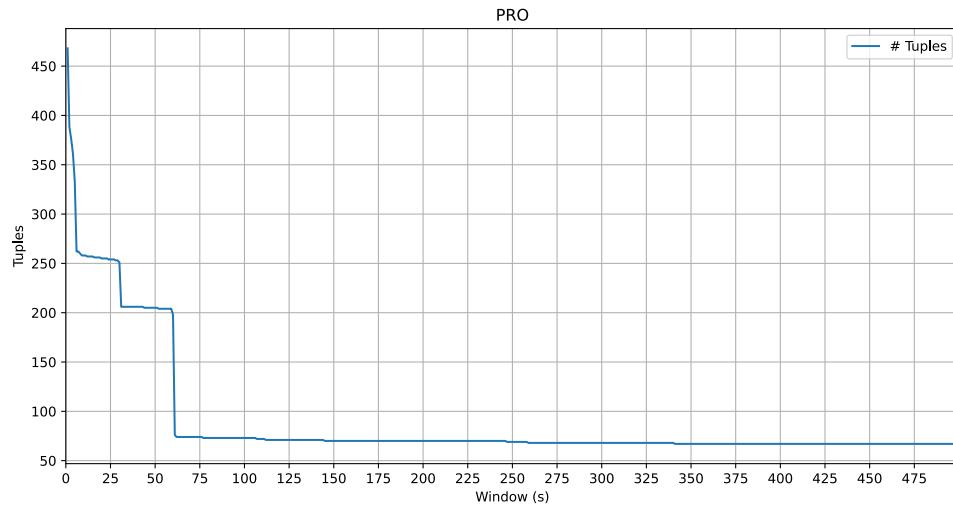


Figura 5.16: Sensitivity Analysis sottosistema PRO

A valle della sensitivity analysis, per ognuno dei sottosistemi si scelgono le finestre di coalescenza riportate in Tabella 5.2.

Sottosistema	Finestra di Coalescenza
DEV	200
MEM	200
I/O	100
NET	100

Tabella 5.2: Finestre di coalescenza sottosistemi

5.3.2.1.1.2 Tupling Si riporta in Tabella 5.3 il numero di tuple ottenuto per ciascun sottosistema eseguendo l'algoritmo di tupling con le finestre di coalescenza precedentemente fissate.

Sottosistema	# Tuple
DEV	334
MEM	105
I/O	107
NET	74

Tabella 5.3: Tuple sottosistemi

5.3.2.2 Data Analysis

5.3.2.2.1 Empirical Reliability Di seguito si riportano le reliability empiriche determinate per ciascun sottosistema. Come si evince dal grafico, il sottosistema meno reliable è DEV.

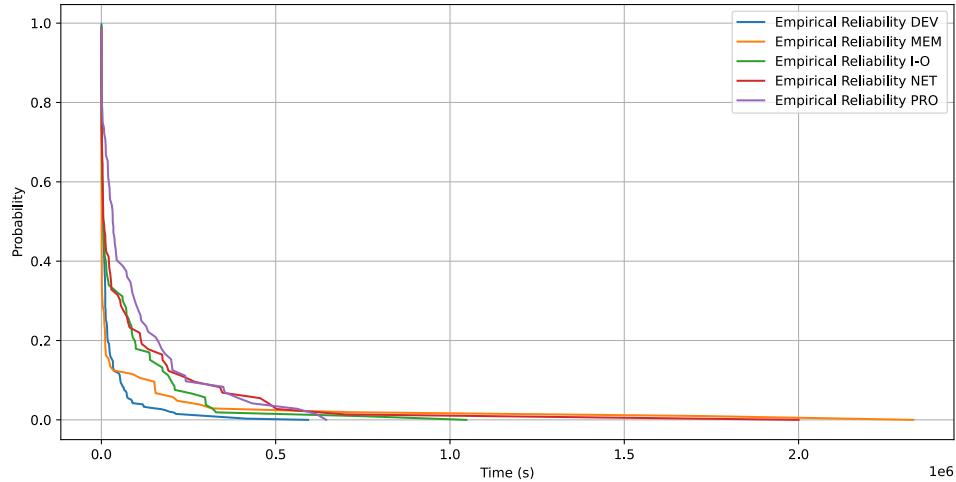


Figura 5.17: Reliability empirica sottosistemi

5.3.3 Tipologie di nodi

Nella sezione corrente si ripetono le stesse operazioni eseguite in precedenza sull'intero sistema, andando però a considerare i dati relativi ad ogni singola tipologia di nodi. Si vanno quindi a suddividere le diverse entry del log a seconda della tipologia del nodo di origine.

In Figura 5.18 si riporta il numero entry presenti nel file di log per ciascuna tipologia di nodi. Come si evince dal grafico, la tipologia più critica è *computation*.

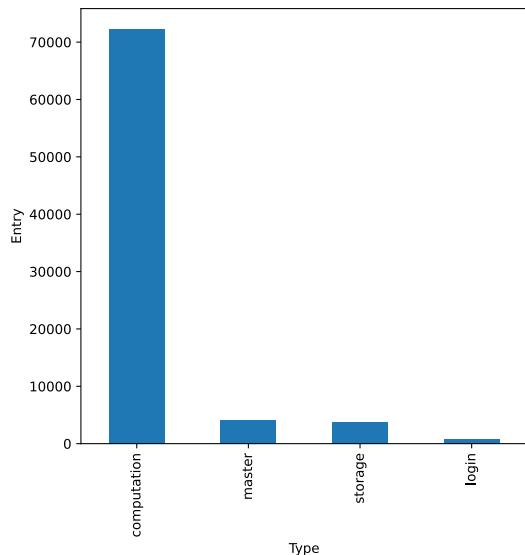


Figura 5.18: Numero di entry per tipologia di nodi

5.3.3.1 Data Manipulation

5.3.3.1.1 Coalescenza spaziale

5.3.3.1.1.1 Sensitivity Analysis Di seguito si propone la sensitivity analysis effettuata per ciascuna tipologia di nodi.

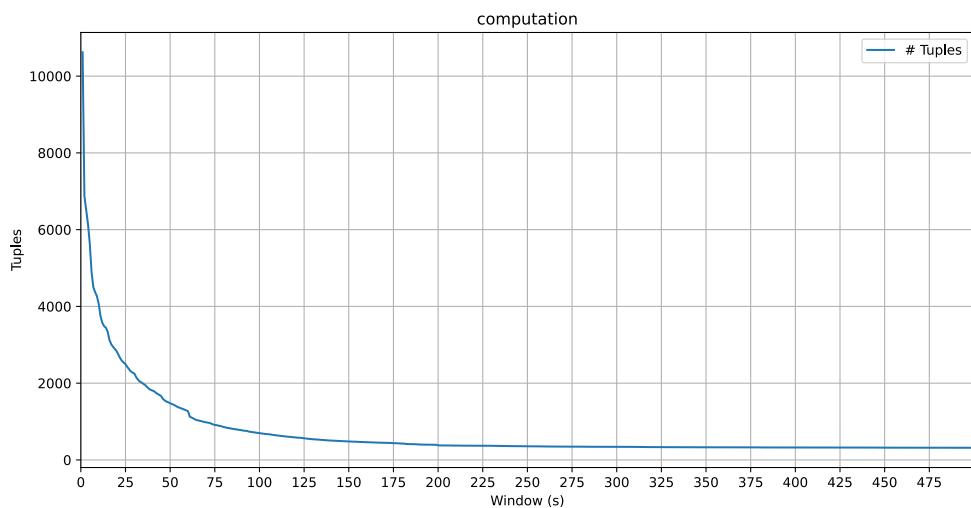


Figura 5.19: Sensitivity Analysis tipologia *computation*

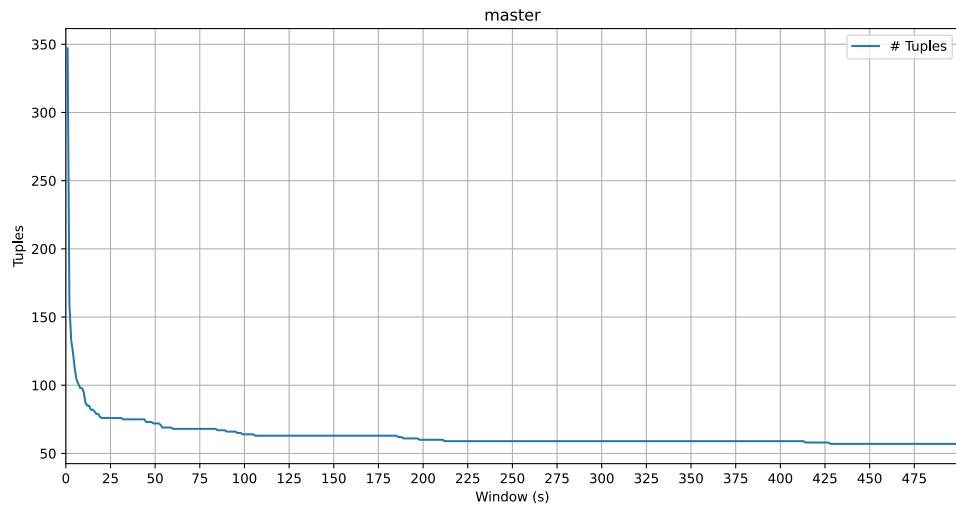


Figura 5.20: Sensitivity Analysis tipologia *master*

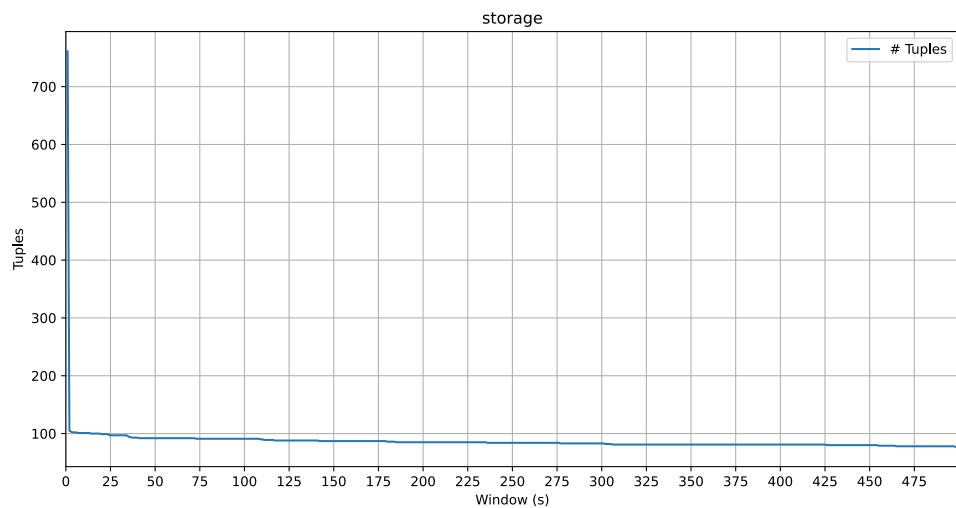


Figura 5.21: Sensitivity Analysis sottosistema I/O

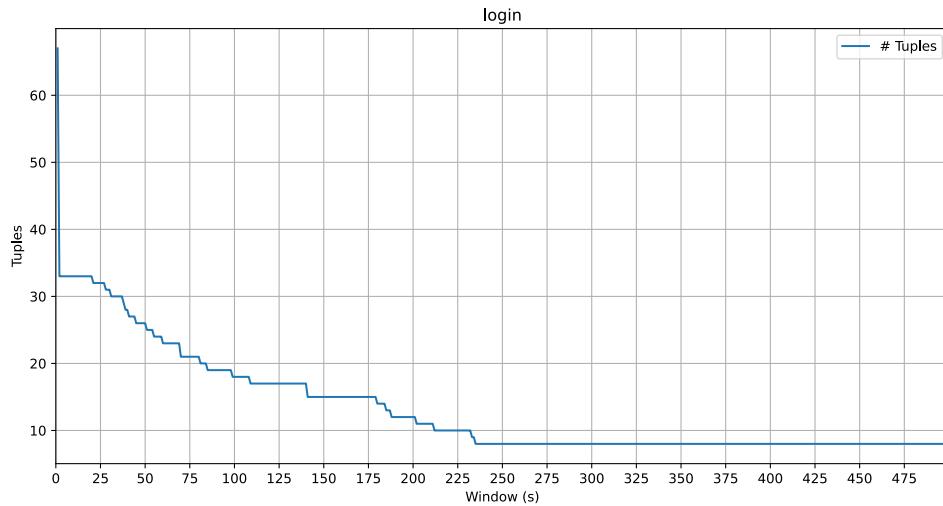


Figura 5.22: Sensitivity Analysis tipologia *login*

A valle della sensitivity analysis, per ogni tipologia si scelgono le finestre di coalescenza riportate in Tabella 5.4.

Tipologia	Finestra di Coalescenza
computation	200
master	200
storage	100
login	300

Tabella 5.4: Finestre di coalescenza tipologie

5.3.3.1.1.2 Tupling Si riporta in Tabella 5.5 il numero di tuple ottenuto per ciascuna tipologia eseguendo l'algoritmo di tupling con le finestre di coalescenza precedentemente fissate:

Tipologia	# Tuple
computation	382
master	60
storage	91
login	8

Tabella 5.5: Tuple tipologie

5.3.3.2 Data Analysis

5.3.3.2.1 Empirical Reliability Di seguito si riportano le reliability empiriche determinate per ciascuna tipologia. Come si evince dal grafico, la tipologia meno reliable è *computation*.

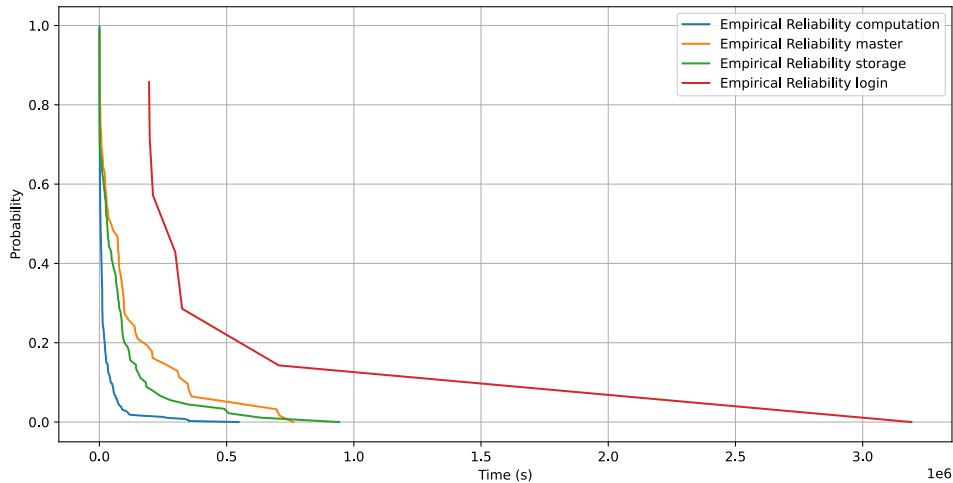


Figura 5.23: Reliability empirica tipologie

5.3.4 Nodi critici

Nella sezione corrente si ripetono le stesse operazioni eseguite in precedenza sull'intero sistema, andando però a considerare i dati relativi ai 5 nodi più critici del sistema.

In Figura 5.24 si riporta il numero di entry presenti nel file di log per ciascun nodo di origine. Come si evince dal grafico, i 5 nodi più critici del sistema, ovvero quelli a cui sono associate più entry del log, sono: *tg-c401*, *tg-master*, *tg-c572*, *tg-s044*, *tg-c238*. Inoltre, è evidente che il nodo *tg-c401* presenta un numero di errori notevolmente superiore rispetto agli altri, il che conferma che tale nodo rappresenta il bottleneck del sistema.

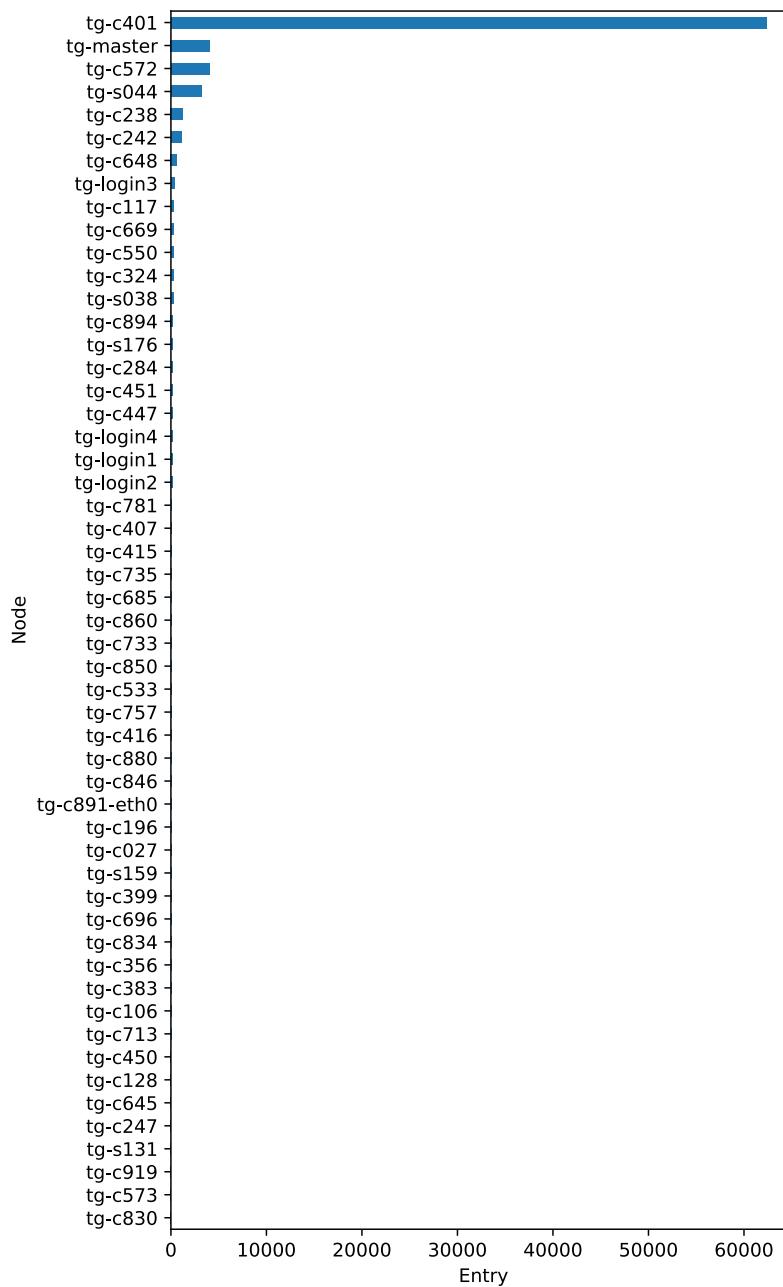


Figura 5.24: Numero di entry per tipologia di nodi

5.3.4.1 Data Manipulation

5.3.4.1.1 Coalescenza temporale In questo caso, dal momento che si considerano entry appartenenti ai singoli nodi si parla di *coalescenza temporale*.

5.3.4.1.1.1 Sensitivity Analysis Di seguito si propone la sensitivity analysis effettuata per ognuno dei 5 nodi più critici.

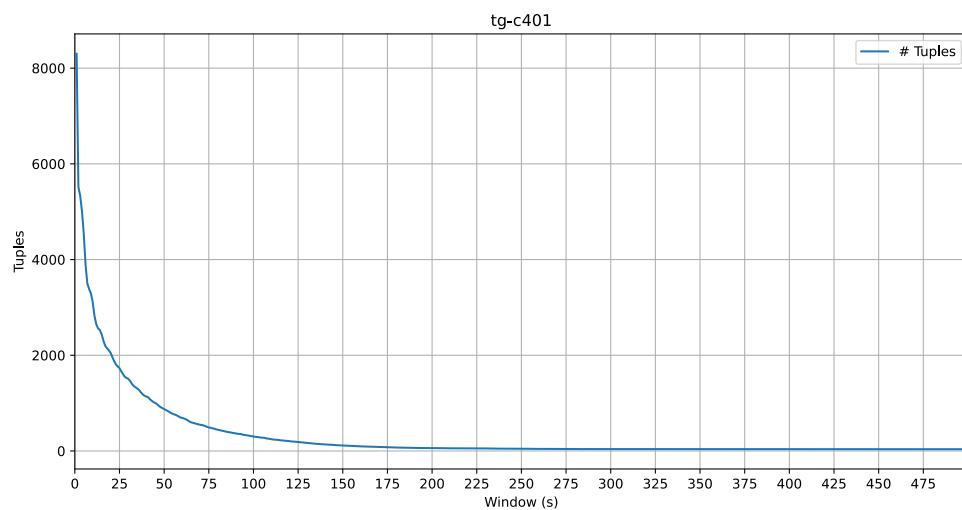


Figura 5.25: Sensitivity Analysis nodo *tg-c401*

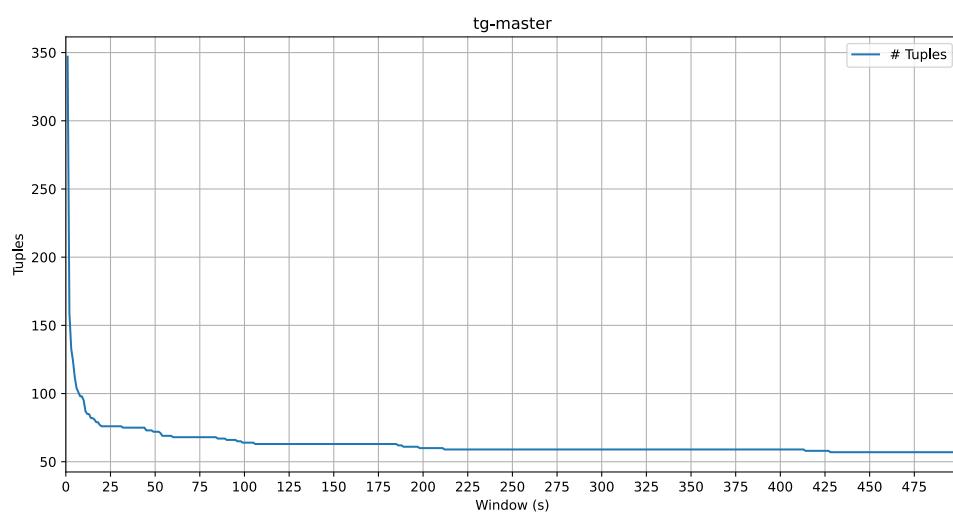


Figura 5.26: Sensitivity Analysis nodo *tg-master*

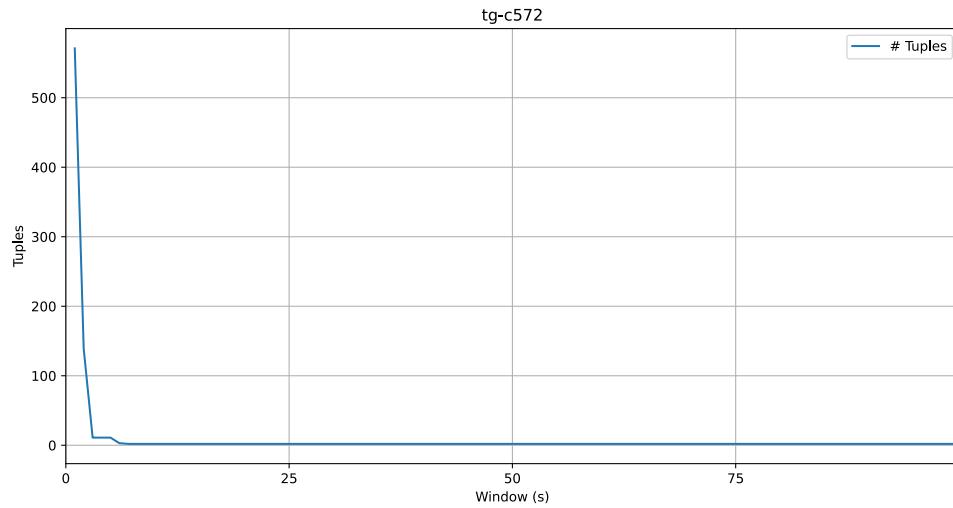


Figura 5.27: Sensitivity Analysis nodo *tg-c572*

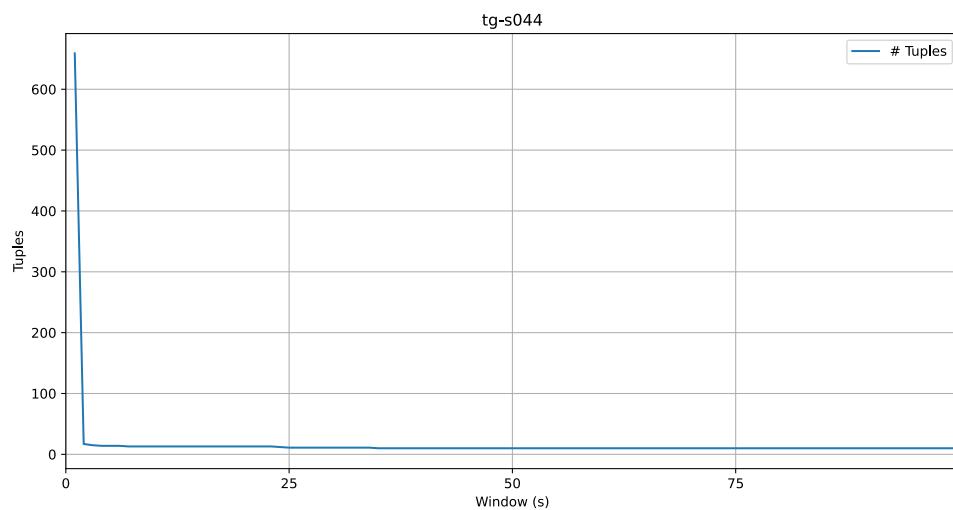


Figura 5.28: Sensitivity Analysis nodo *tg-s044*

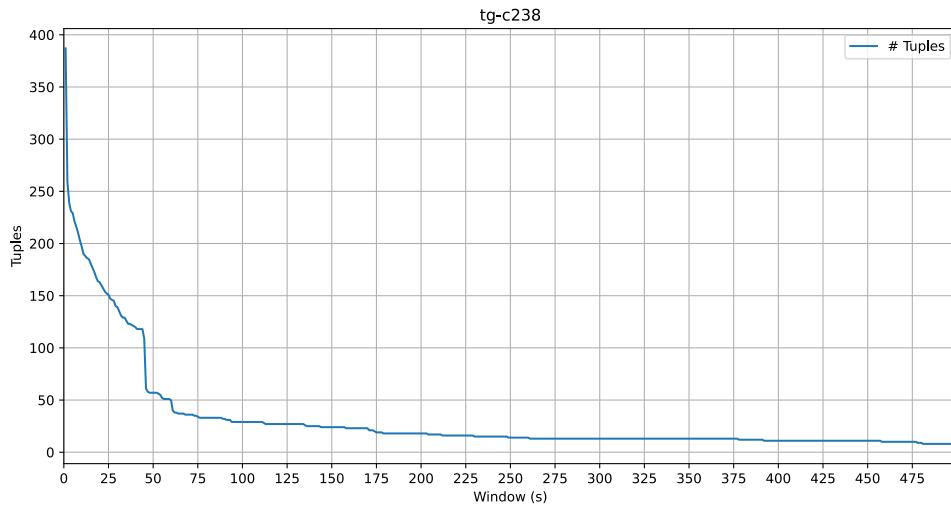


Figura 5.29: Sensitivity Analysis nodo *tg-c238*

A valle della sensitivity analysis, per ognuno dei nodi più critici si scelgono le finestre di coalescenza riportate in Tabella 5.6.

Nodo	Finestra di Coalescenza
tg-c401	200
tg-master	200
tg-c572	20
tg-s044	30
tg-c238	200

Tabella 5.6: Finestre di coalescenza nodi critici

5.3.4.1.1.2 Tupling Si riporta in Tabella 5.7 il numero di tuple ottenuto per ciascun nodo eseguendo l'algoritmo di tupling con le finestre di coalescenza precedentemente fissate:

Nodo	# Tuple
tg-c401	59
tg-master	60
tg-c572	2
tg-s044	11
tg-c238	18

Tabella 5.7: Tuple nodi critici

5.3.4.2 Data Analysis

5.3.4.2.1 Empirical Reliability Di seguito si riportano le reliability empiriche determinate per i nodi più critici che presentano un numero di tuple significativo. Come si evince dal grafico, il nodo meno reliable è *tg-c401*.

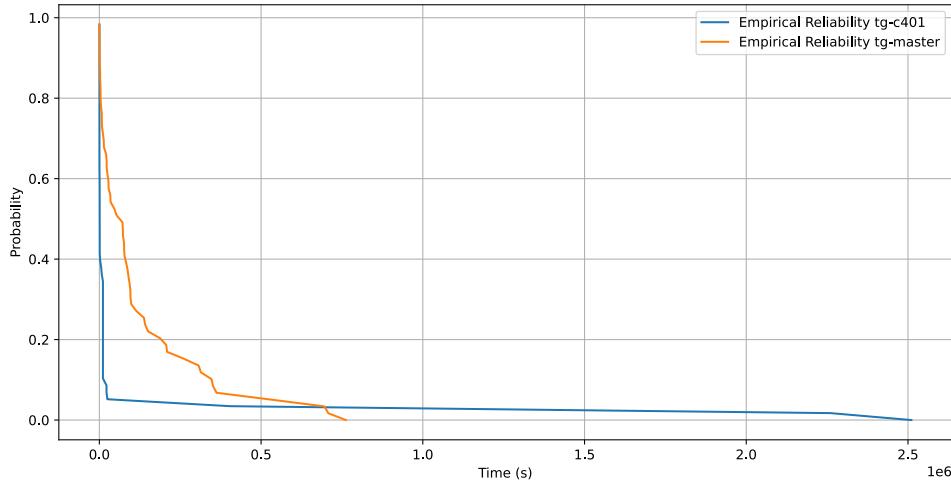


Figura 5.30: Reliability empirica nodi critici

5.3.5 Bottleneck

Come visto nel Paragrafo 5.3.1.1.1.3 e nella sezione precedente, il bottleneck del sistema è rappresentato dal nodo *tg-401*. Circa il 77% degli errori presenti nel file di log si sono verificati su tale nodo, in un periodo di tempo compreso tra il 12/02/2007 e il 14/02/2007. Inoltre, in questo arco di tempo si verifica circa il 99% di tutti gli errori generati dal nodo *tg-401*.

Nella sezione corrente si ripetono le stesse operazioni eseguite in precedenza sull'intero sistema, andando però a considerare solo i dati relativi al bottleneck del sistema, ovvero al nodo *tg-401* nell'arco temporale compreso tra il 12 e il 14 febbraio 2007.

5.3.5.1 Data Manipulation

5.3.5.1.1 Coalescenza temporale

5.3.5.1.1.1 Sensitivity Analysis Di seguito si propone la sensitivity analysis effettuata per il bottleneck del sistema.

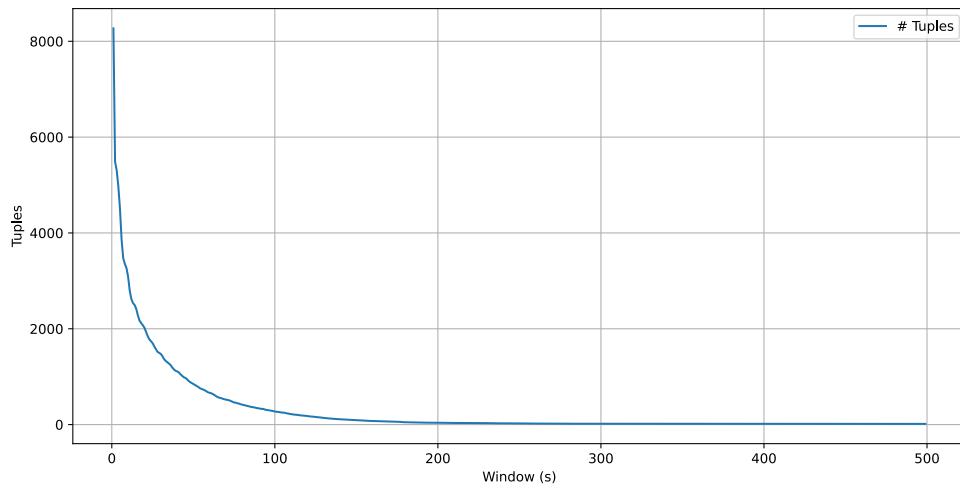


Figura 5.31: Sensitivity Analysis bottleneck

Si sceglie come dimensione della finestra di coalescenza 200 secondi.

5.3.5.1.1.2 Tupling Eseguendo l'algoritmo di tupling con la finestra di coalescenza fissata si ottengono 37 tuple totali.

5.3.5.2 Data Analysis

5.3.5.2.1 Empirical Reliability Di seguito si riporta l'andamento della CDF e della reliability empirica del bottleneck del sistema.

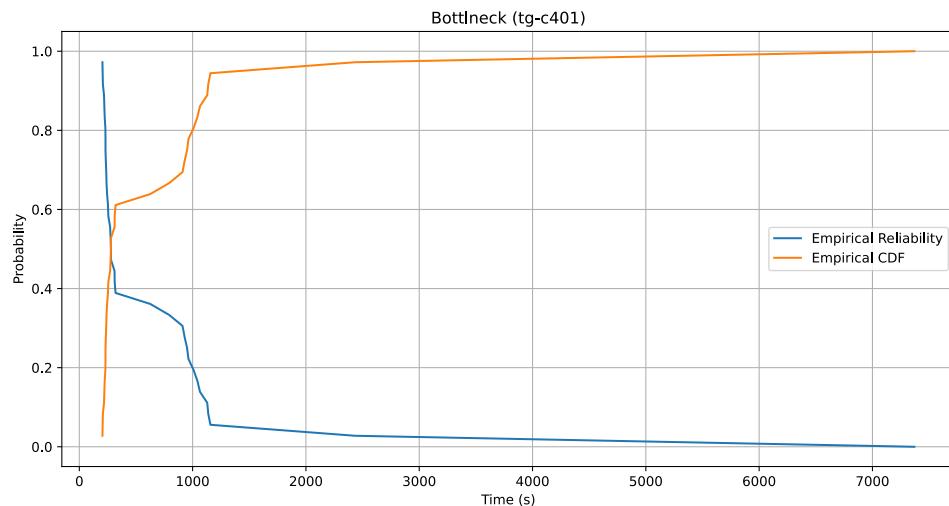


Figura 5.32: Reliability empirica bottleneck

5.3.5.2.2 Ulteriori analisi Nella Figura 5.33 si può osservare come gli errori relativi al bottleneck si ripartiscono nei soli due sottosistemi DEV e MEM.

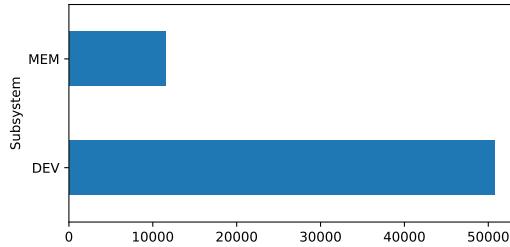


Figura 5.33: Numero di errori per sottosistema del nodo *tg-401*

Infine, dalla Tabella 5.8 si nota che gli errori generati dal nodo sono prevalentemente di tipo hardware.

Subsystem	Message	# Errors
DEV	+Platform Specific Error Detail:	5474
	+BEGIN HARDWARE ERROR STATE AT CPE	6030
	+END HARDWARE ERROR STATE AT CPE	3830
	+Platform PCI Component Error Info Section	6805
	+Platform Specific Error Info Section	5416
	Component Info: Vendor Id =x x	23152
MEM	+Mem Error Detail:	4
	Physical Address x	11552

Tabella 5.8: Messaggi bottleneck

5.3.6 Ulteriori analisi

Per concludere la trattazione sul sistema Mercury si riportano delle ulteriori analisi che evidenziano alcuni risultati interessanti.

5.3.6.1 Sottosistemi e Tipologie di nodi

In Figura 5.34 viene illustrato come i diversi sottosistemi sono distribuiti tra le tipologie di nodi all'interno del file di log e viceversa.

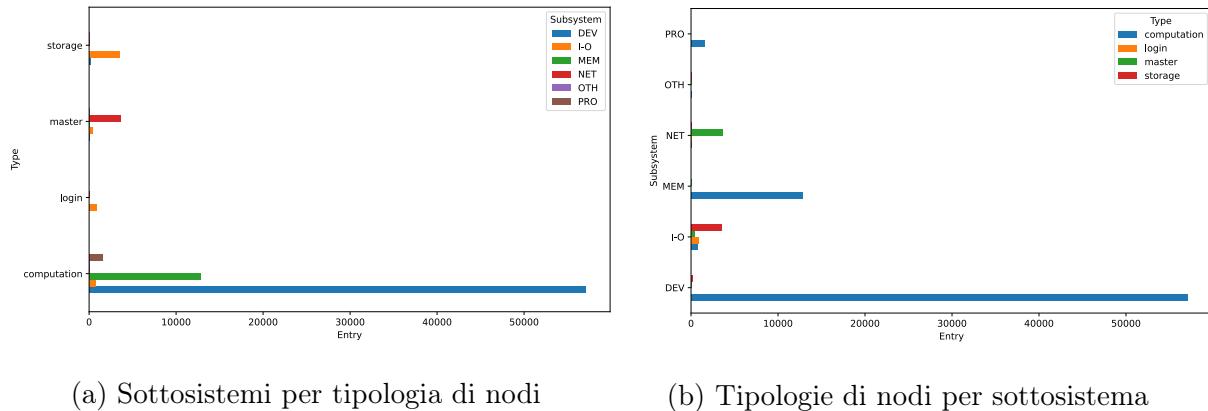


Figura 5.34: Sottosistemi e Tipologie di nodi

Dai grafici si possono trarre alcune conclusioni interessanti:

- gli errori originati dai nodi di tipo *computation* riguardano principalmente il sottosistema DEV e in parte MEM;
- gli errori originati dai nodi di tipo *storage* e *login* riguardano principalmente il sottosistema I-O;
- gli errori originati dal nodo di tipo *master* riguardano principalmente il sottosistema NET.

5.4 Blue Gene/L

Blue Gene/L (BGL) è un supercomputer sviluppato da IBM. Come riportato in Figura 5.35, il sistema è costituito da 64 *rack* (o *cabinet*), ognuno dei quali è diviso in 2 *midplane*; ciascun midplane contiene 16 *node board* (32 per rack), ognuna delle quali contiene a sua volta 16 *compute card* e 2 *I/O card* opzionali. I nodi dotati di una card di I/O sono *N0*, *N4*, *N8* ed *NC*.

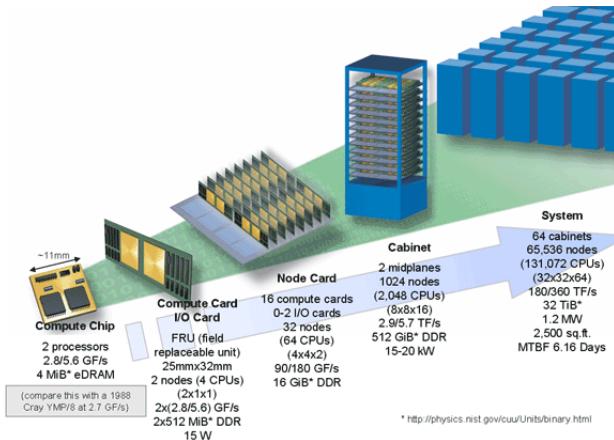


Figura 5.35: Architettura Blue Gene/L

I dati da analizzare relativi a BGL sono contenuti in un file di log chiamato *BGLErrorLog.txt*. Il file contiene 125624 entry, ognuna delle quali prevede i seguenti campi:

- *timestamp*: intero in formato UNIX;
- *nodo di origine*: stringa del tipo *Rxx-Mx-Nx* indicante il nodo in cui si è verificato l'errore. In particolare, sono indicati rack, midplane e nodo del sistema che ha generato l'evento.
- *card di origine*: stringa del tipo *Jxx-Uxx* indicante la card del nodo da cui ha origine l'errore. È possibile divere le card in due grandi categorie, card di I/O (*J18-Uxx*) e card di computation.
- *messaggio testuale*: stringa indicante il messaggio relativo all'errore.

All'interno del file di log sono riportate tutte e sole le entry relative ai fatal error. In altre parole, è stata effettuata una fase di data filtering preliminare.

5.4.1 Sistema

In questa sezione si riportano le ultime due fasi della FFDA, eseguite considerando i dati relativi all'intero sistema Blue Gene/L.

5.4.1.1 Data Manipulation

5.4.1.1.1 Coalescenza spaziale

5.4.1.1.1.1 Sensitivity Analysis Di seguito si propone la sensitivity analysis effettuata per i dati del sistema.

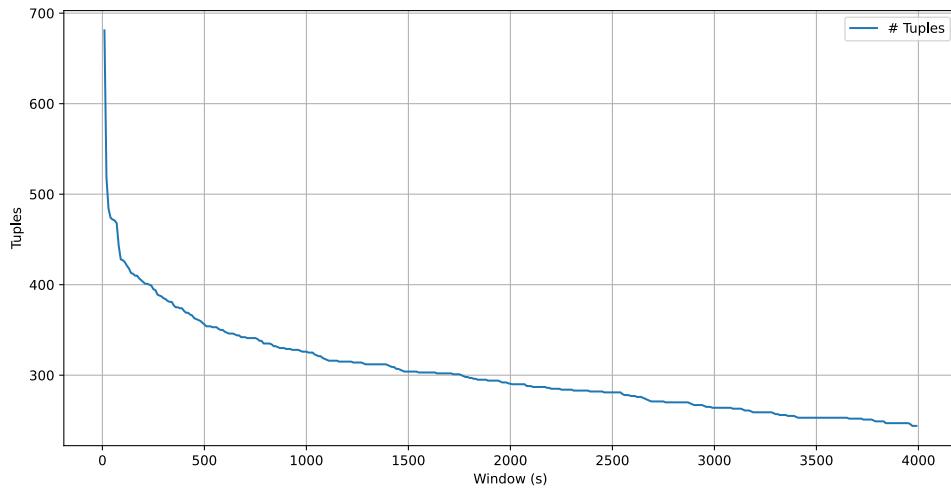


Figura 5.36: Sensitivity Analysis sistema

Si è scelta come dimensione della finestra di coalescenza il valore immediatamente successivo al gomito della curva, ovvero 500 secondi (circa 8 minuti).

5.4.1.1.2 Tupling Eseguendo l'algoritmo di tupling con la finestra di coalescenza fissata si ottengono 356 tuple totali.

Dopo aver determinato le tuple è possibile studiarne alcune statistiche, come fatto per Mercury nel paragrafo 5.3.1.1.2.

In Figura 5.37 si riporta il numero di entry del log contenute in ogni tupla. A differenza di Mercury, dal grafico è possibile osservare che le tuple ottenute sono abbastanza bilanciate, in termini di numero di entry contenute. Questo suggerisce che gli errori registrati nel file di log sono dispersi più uniformemente nel tempo, rispetto al sistema precedente.

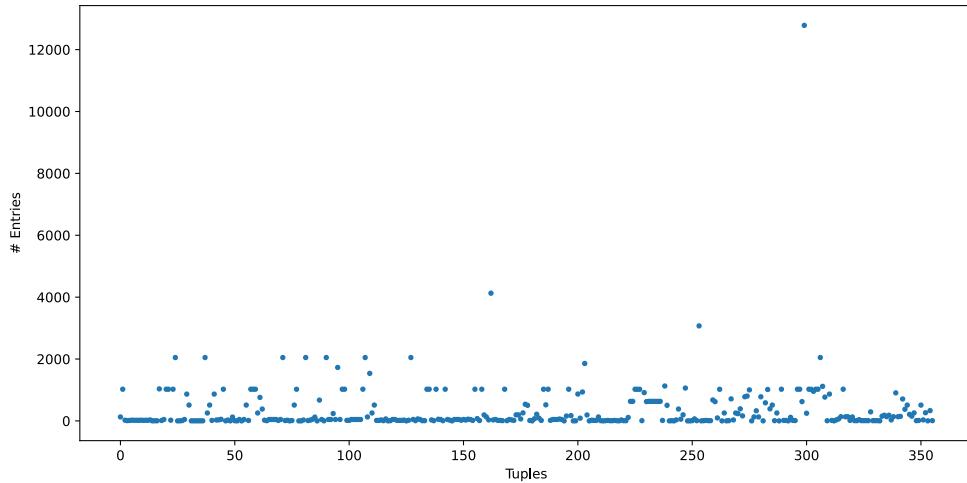


Figura 5.37: Tupling Blue Gene/L

5.4.1.1.2 Analisi dei troncamenti Come detto in precedenza, la scelta della finestra di coalescenza influisce sul numero di troncamenti e collisioni commessi a causa del tupling.

Si vuole quindi effettuare una stima dei troncamenti che commessi. Per prima cosa si calcola il valore q corrispondente al 10° percentile (0.1 quantile) dei tempi di interarrivo. Tale valore è pari a 1074 secondi (circa 18 minuti), circa il doppio della finestra di coalescenza utilizzata. Si individuano poi tutte le coppie di tuple adiacenti che distano meno di questo valore. Per ognuna di queste coppie è possibile stimare un troncamento. Da tale stima si ottengono 36 troncamenti, il che significa che per circa il 10% delle tuple totali potrebbe essersi verificato un troncamento.

5.4.1.1.3 Analisi delle collisioni Analogamente si vuole effettuare una stima delle collisioni commesse a causa del tupling.

Una prima soluzione prevede di calcolare numero di potenziali collisioni individuando il numero di tuple per le quali si hanno più rack di origine. Secondo questa soluzione vengono stimate 177 collisioni, il che significa che per circa il 50% delle tuple totali potrebbero essersi verificate collisioni. Tuttavia, data la scelta opportuna della finestra di coalescenza, tale risultato sembra eccessivo.

Si è pensato quindi di seguire un secondo approccio, secondo cui il numero di collisioni è dato dal numero di tuple per le quali si hanno più *tipologie di card* differenti, dove per tipologia di card si intende I-O o *computation*. Seguendo tale approccio si sono stimate 43 collisioni, il che significa che per circa il 12% delle tuple totali potrebbero essersi verificate collisioni.

5.4.1.2 Data Analysis

5.4.1.2.1 Empirical Time To Failure (TTF) In Figura 5.38 si riportano i risultati del fitting effettuato per tutte le distribuzioni previste dalla libreria *Reliability* [3].

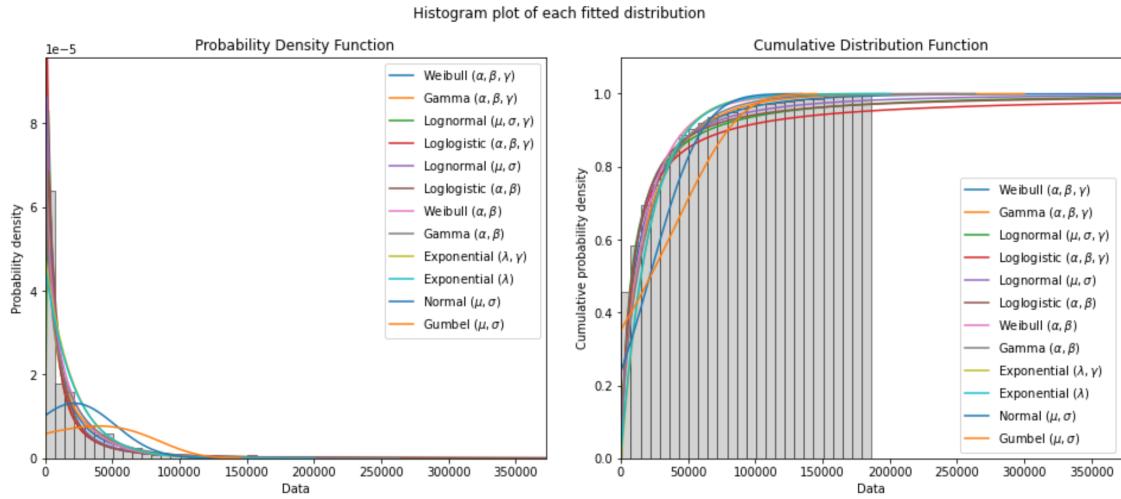


Figura 5.38: Test di fitting

Come si evince dai grafici precedenti, la distribuzione che meglio approssima i dati del sistema è una *Weibull* caratterizzata da 3 parametri.

Si riportano in Figura 5.39 i risultati ottenuti effettuando il fitting mirato della distribuzione Weibull a 3 parametri, i quali assumono i valori in Tabella 5.9.

α	β	γ
16201.8	0.671662	503

Tabella 5.9: Parametri Weibull

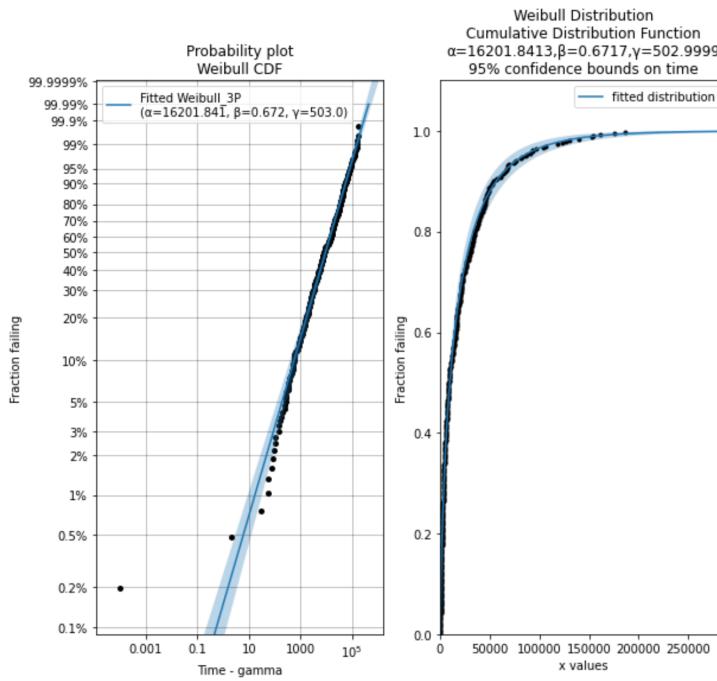


Figura 5.39: Test di fitting con distribuzione Weibull

Per avere un'ulteriore conferma dei risultati ottenuti dai test precedenti, è possibile effettuare il test di Kolmogorov-Smirnov.

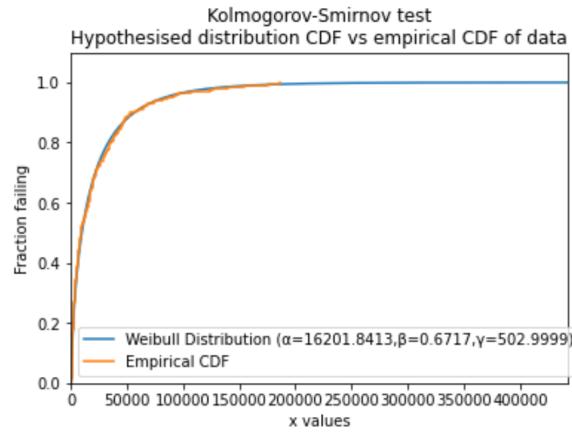


Figura 5.40: Test di Kolmogorov-Smirnov

Effettuato il test è possibile affermare che, con un livello di significatività di 0.05, si può accettare l'ipotesi che i dati provengono da una distribuzione Weibull avente i parametri riportati in precedenza.

Per quanto discusso nel Paragrafo 5.3.1.2.1, a supporto dei test precedenti, è stato inoltre eseguito il test di Kolmogorov-Smirnov per le distribuzioni lognormale e weibull con due parametri ed esponenziale al fine di individuare la natura dei fallimenti che coinvolgono il sistema.

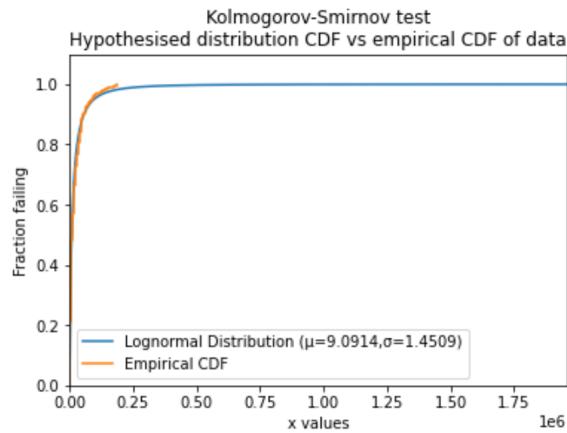


Figura 5.41: Test di Kolmogorov-Smirnov distribuzione Lognormale

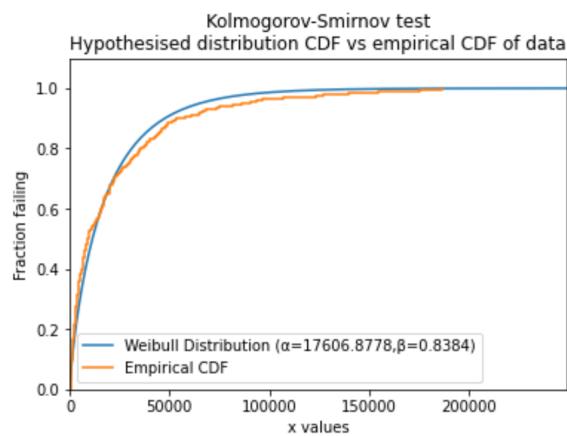


Figura 5.42: Test di Kolmogorov-Smirnov distribuzione Weibull

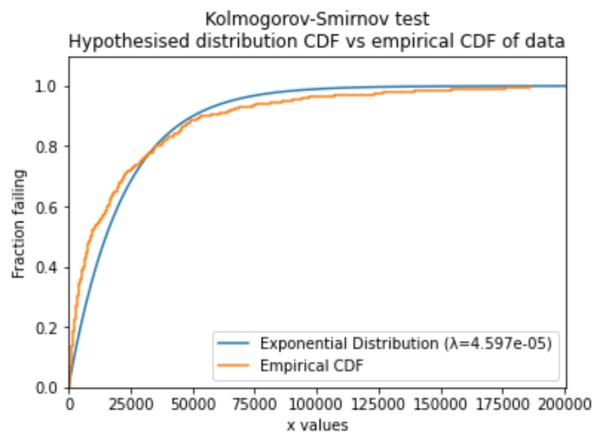


Figura 5.43: Test di Kolmogorov-Smirnov distribuzione Esponenziale

I risultati del test riportati nelle Figure 5.41, 5.42 e 5.43 rigettano la provenienza della distribuzione empirica dalle distribuzioni weibull ed esponenziale, mentre accettano l'ipotesi che i dati provengano da una distribuzione lognormale con un livello di significatività di 0.05

Da una prima analisi quindi si deduce che la natura dei fallimenti che coinvolgono il sistema Blue Gene/L è di tipo software.

5.4.1.2.2 Empirical Reliability Di seguito si riporta in Figura 5.44 l'andamento della CDF e della reliability empirica del sistema.

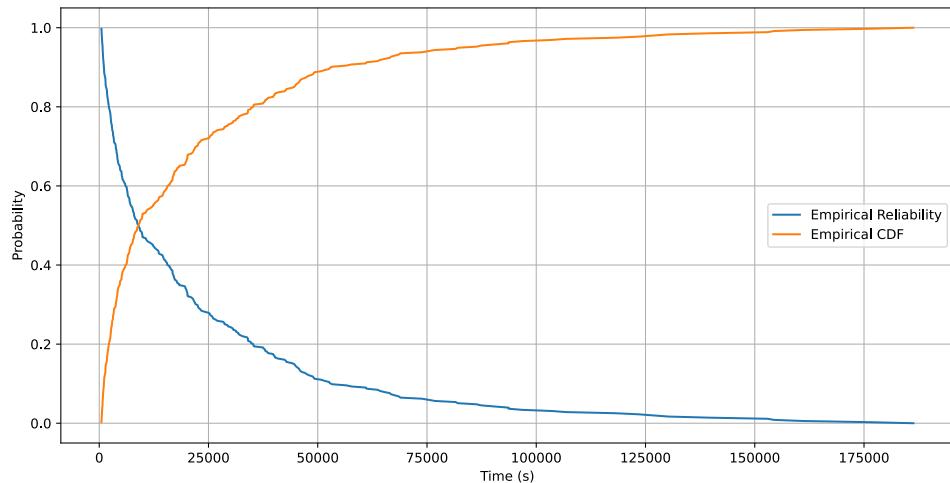


Figura 5.44: Reliability empirica

5.4.2 Rack critici

Nella sezione corrente si ripetono le stesse operazioni eseguite in precedenza sull'intero sistema, andando però a considerare i dati relativi agli 8 rack più critici del sistema. In Figura 5.45 si riporta il numero di entry presenti nel file di log per ciascun rack di origine. Come si evince dal grafico, gli 8 rack più critici del sistema, ovvero quelli a cui sono associate più entry del log, sono: *R63, R62, R57, R56, R46, R03*.

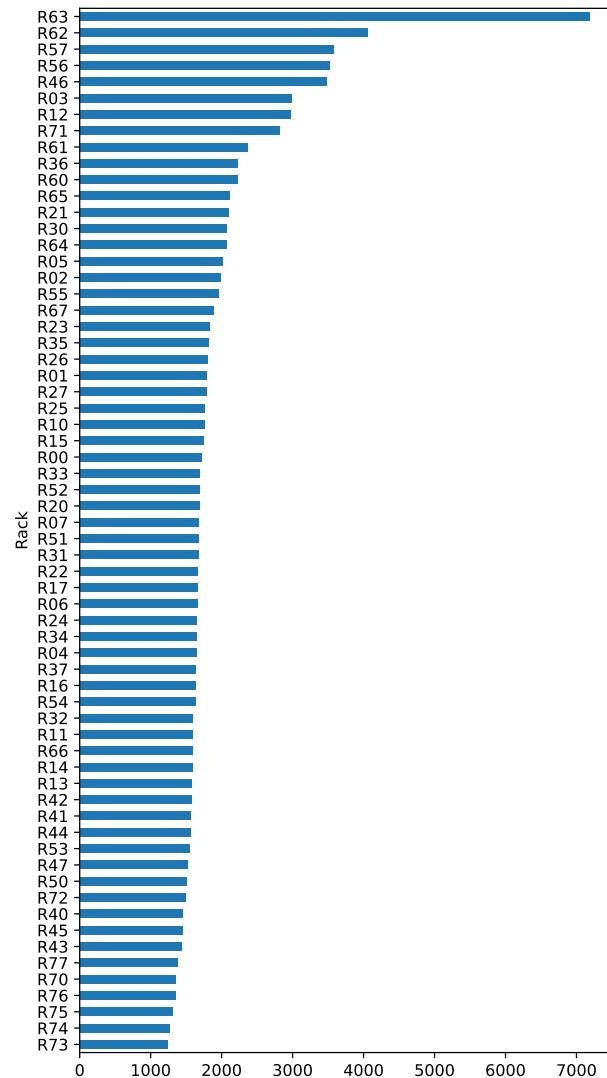


Figura 5.45: Numero di entry per rack

È possibile approfondire l'analisi dei rack più critici, studiando come le entry presenti nel log si dividono tra i diversi midplane di tali rack. Come si evince dal grafico in Figura 5.46, gli errori sono abbastanza bilanciati tra i due midplane, fatta eccezione che per il rack *R63*.

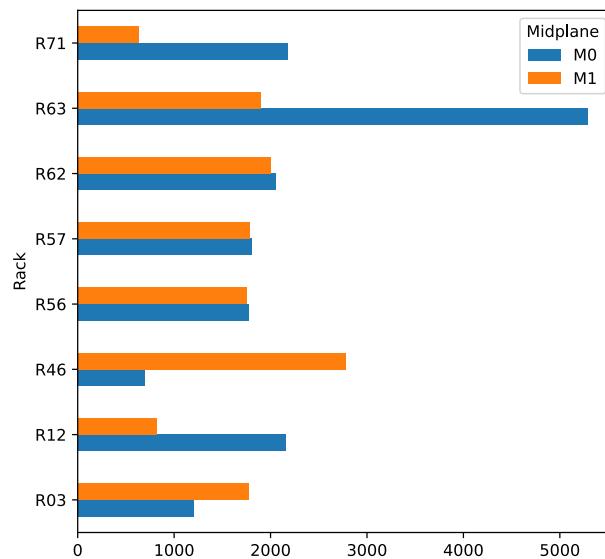


Figura 5.46: Numero di entry per midplane

Analogamente, in Figura 5.47 viene mostrato come gli errori relativi ai rack più critici sono distribuiti tra i diversi nodi dei midplane.

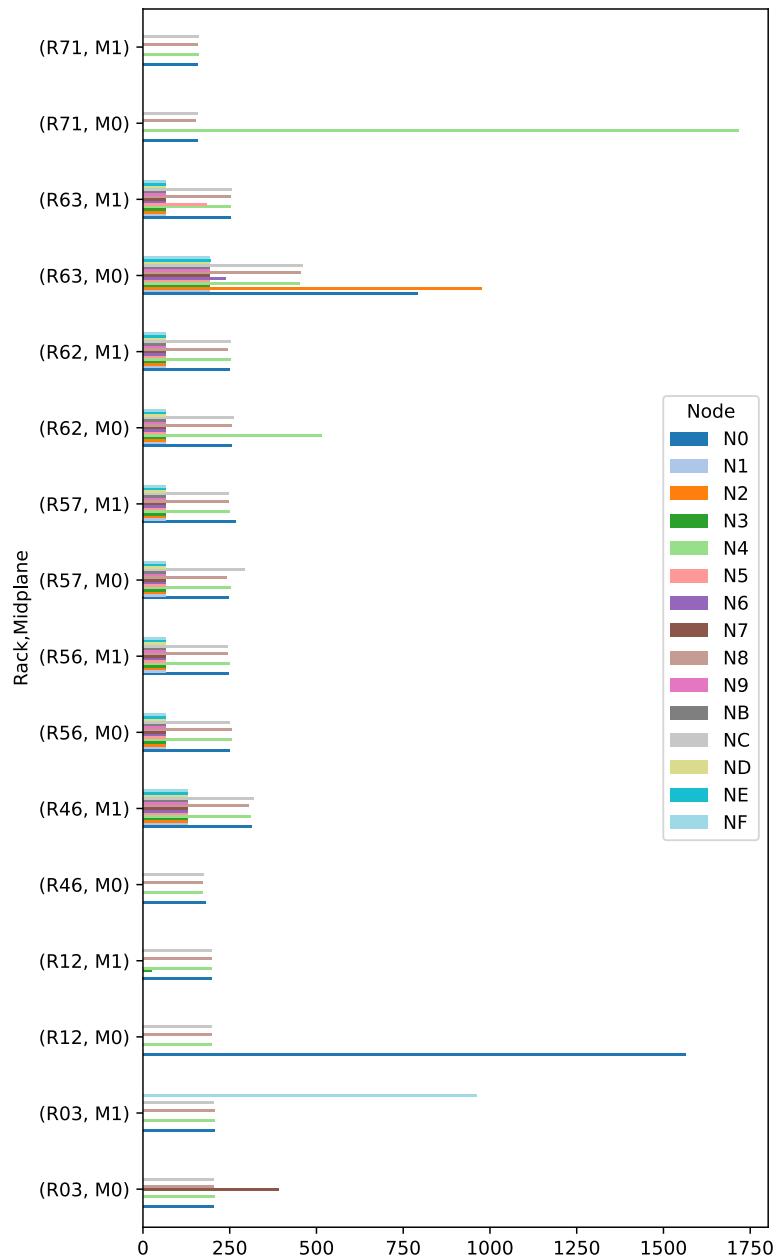


Figura 5.47: Numero di entry per nodo

Dal grafico è possibile estrarre le seguenti osservazioni:

- tutti gli 8 rack più critici presentano numerosi errori sui 4 nodi dotati della card di I/O ($N0, N4, N8, NC$);
- i primi 5 rack più critici presentano, oltre che ad errori sui nodi dotati anche di I/O, numerosi errori anche sui nodi puramente di calcolo;
- il rack più critico, $R63$, presenta numerosi errori relativi a tutti i nodi;
- il rack $R71$ possiede il nodo in cui si ha il maggior numero di fallimenti, ovvero $N4$, nodo dotato di I/O card.

5.4.2.1 Data Manipulation

5.4.2.1.1 Coalescenza spaziale

5.4.2.1.1.1 Sensitivity Analysis Di seguito si propone la sensitivity analysis effettuata per ognuno degli 8 rack più critici del sistema.

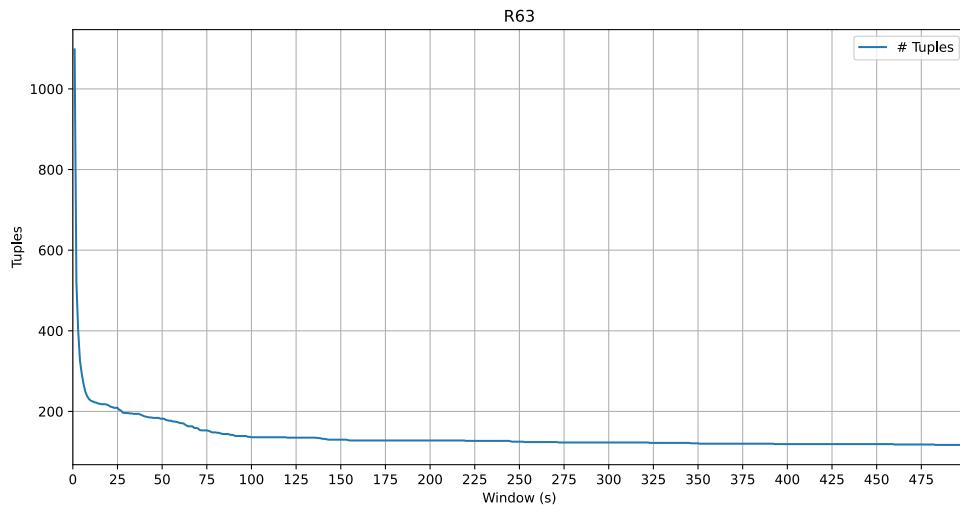


Figura 5.48: Sensitivity Analysis rack R63

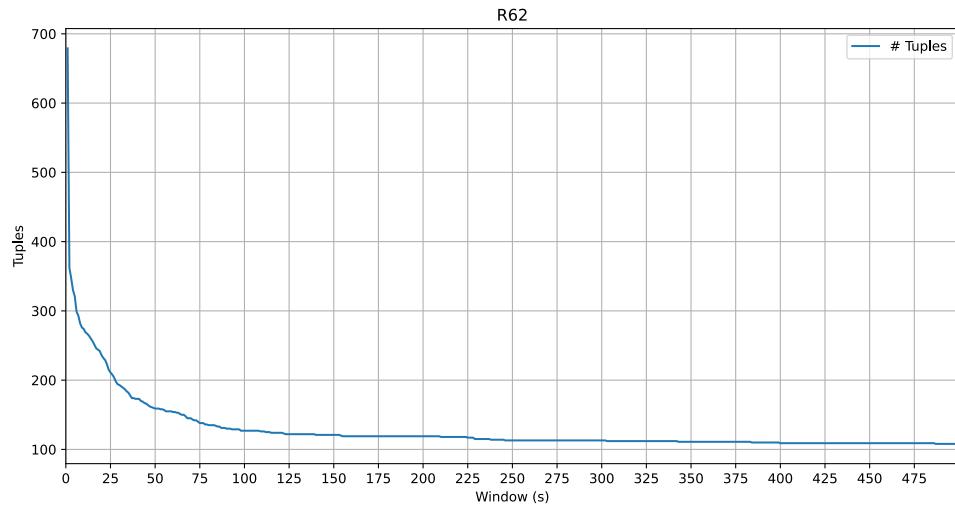


Figura 5.49: Sensitivity Analysis rack R62

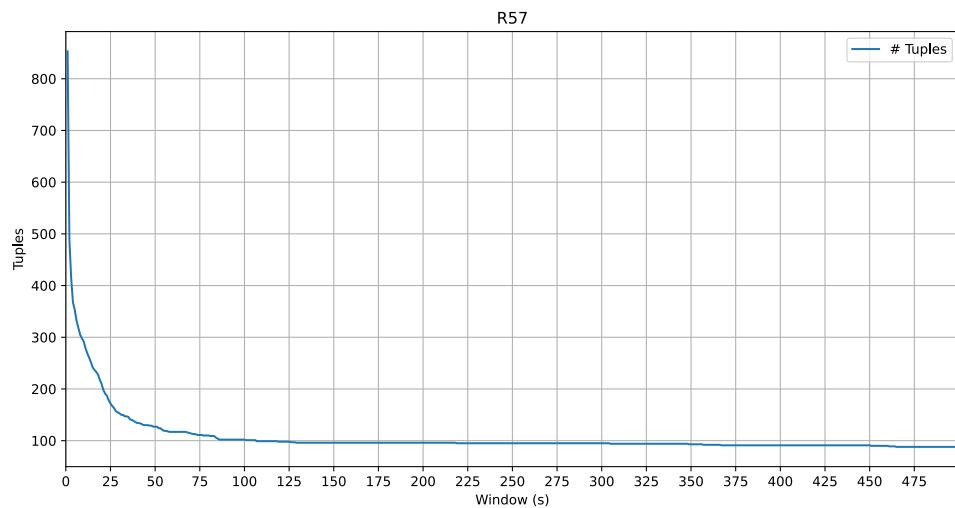


Figura 5.50: Sensitivity Analysis rack R57

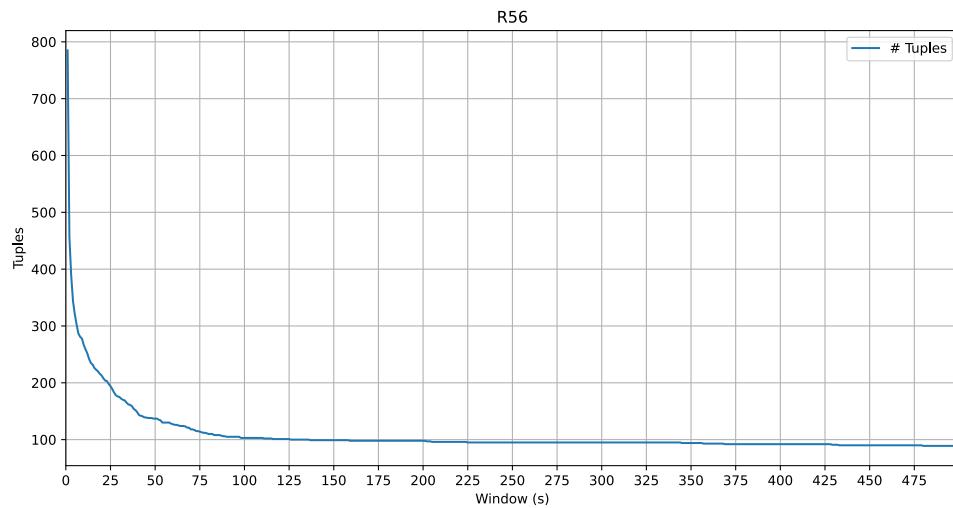


Figura 5.51: Sensitivity Analysis rack R56

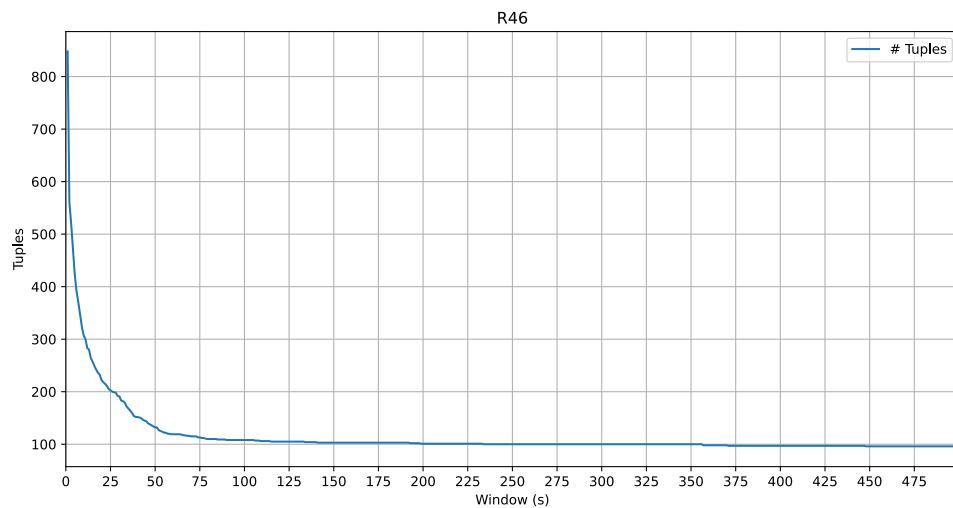


Figura 5.52: Sensitivity Analysis rack R46

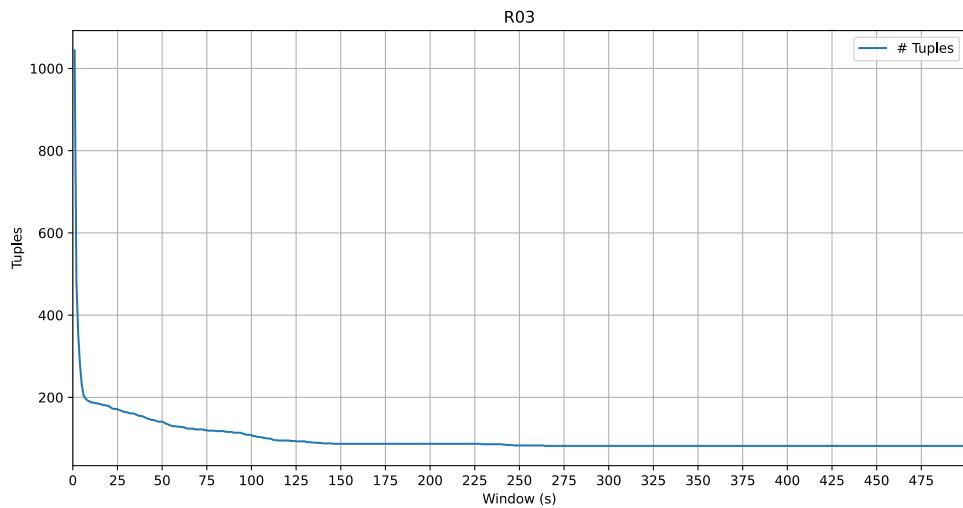


Figura 5.53: Sensitivity Analysis rack R03

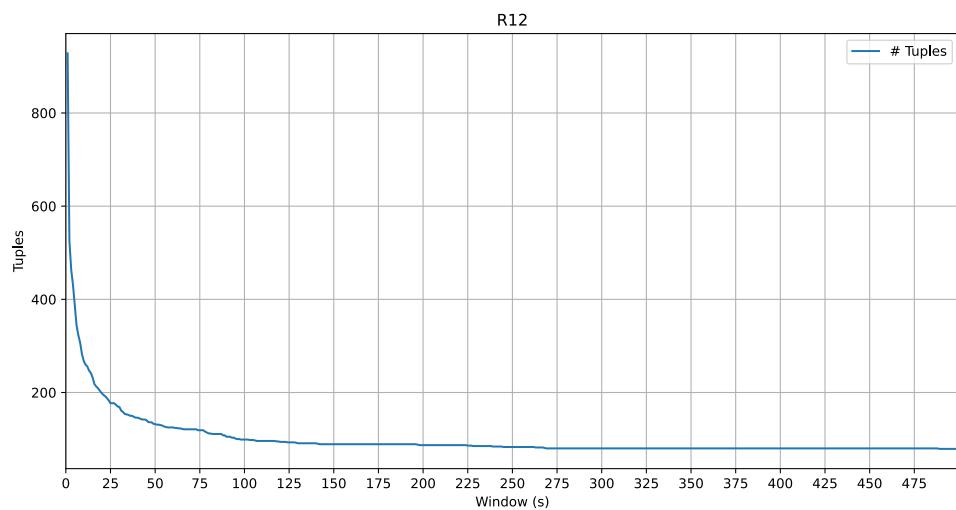


Figura 5.54: Sensitivity Analysis rack R12

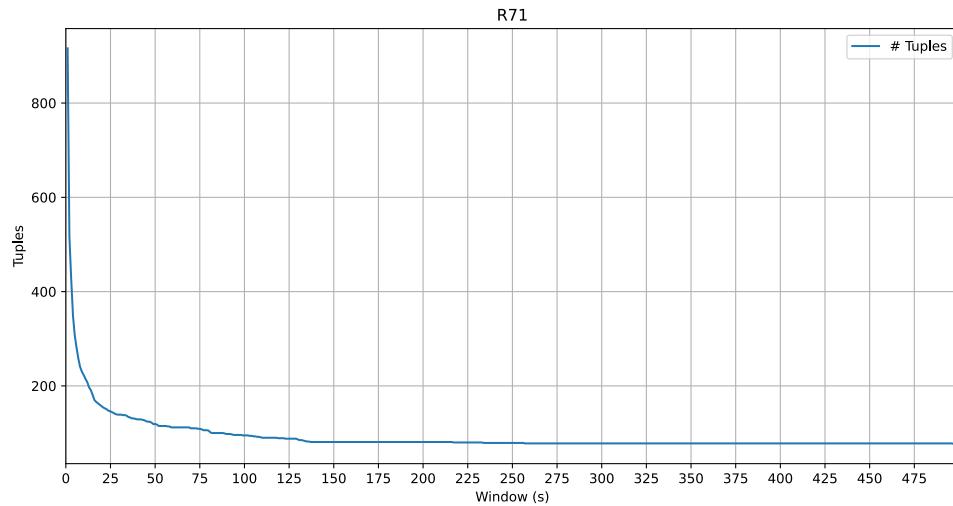


Figura 5.55: Sensitivity Analysis rack R71

A valle della sensitivity analysis, per ognuno dei rack più critici si scelgono le finestre di coalescenza riportate in Tabella 5.10.

Rack	Finestra di Coalescenza
R63	75
R62	75
R57	75
R56	75
R46	75
R03	75
R12	75
R71	75

Tabella 5.10: Finestre di coalescenza rack critici

5.4.2.1.1.2 Tupling Si riporta in Tabella 5.11 il numero di tuple ottenuto per ciascun rack eseguendo l'algoritmo di tupling con le finestre di coalescenza precedentemente fissate:

Rack	# Tuple
R63	153
R62	138
R57	111
R56	114
R46	113
R03	120
R12	119
R71	109

Tabella 5.11: Tuple rack critici

5.4.2.2 Data Analysis

5.4.2.2.1 Empirical Reliability Di seguito si riportano le reliability empiriche determinate per i rack più critici. Come si evince dal grafico, il rack meno reliable è *R63*, tuttavia la differenza rispetto agli altri non è così netta.

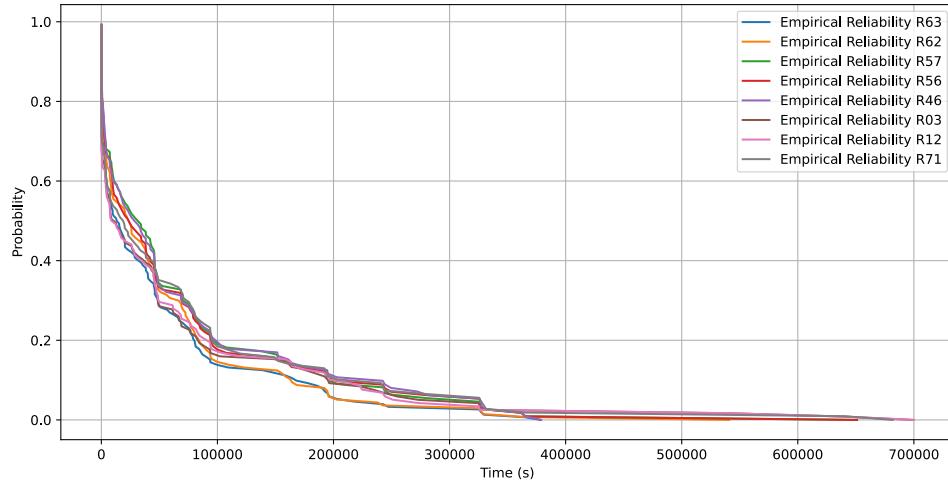


Figura 5.56: Reliability empirica rack critici

5.4.3 Tipologie di card

Nella sezione corrente si ripetono le stesse operazioni eseguite in precedenza sull'intero sistema, andando però a considerare i dati relativi ad ogni tipologia di card (computation e I/O card). Si vanno quindi a suddividere le diverse entry del log a seconda della tipologia di card di origine.

In Figura 5.57 si riporta il numero entry presenti nel file di log per ciascuna tipologia di card. Come si evince dal grafico, la tipologia più critica è *I-O*.

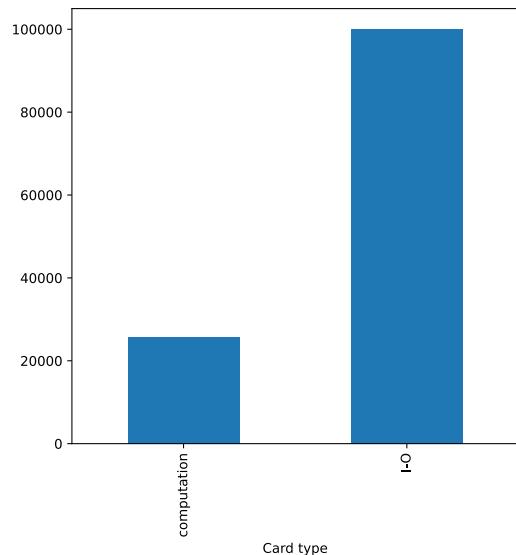


Figura 5.57: Numero di entry per tipologia di card

In Figura 5.58 è possibile osservare come le due tipologie di card si distribuiscono tra le diverse tuple, quindi nel tempo. Dal grafico si evince che le diverse tipologie sono distribuite abbastanza uniformemente nel tempo.

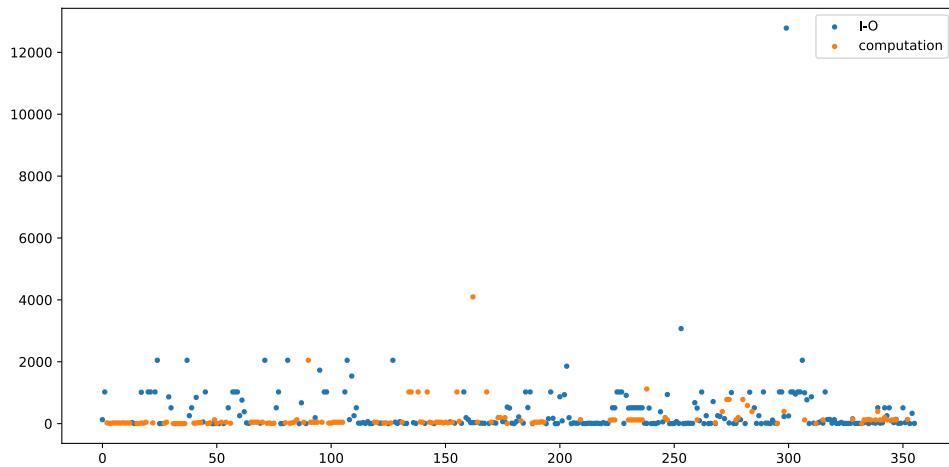


Figura 5.58: Distribuzione degli errori nelle tuple

In Figura 5.59 è possibile invece osservare come le due diverse tipologie di card si distribuiscono tra i diversi rack, quindi nello spazio. Dal grafico si evince che i problemi di I/O sono di fatto comuni a tutti i rack. Un ulteriore aspetto interessante è che i

rack più critici presentano non solo problemi legati alle card di I/O ma anche a quelle di computation.

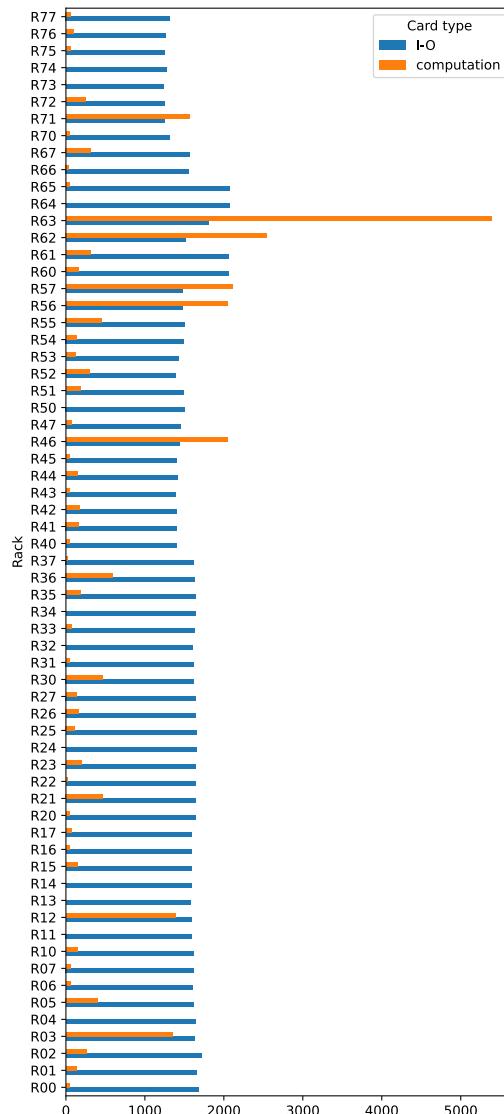


Figura 5.59: Numero di errori per rack

Infine, in Figura 5.60 si riporta il numero di errori per ciascuna card di origine. Come previsto, le card più critiche sono quelle di I/O (*J18-U11* e *J18-U01*). Dal grafico si

evince che il numero di errori è pressoché identico per entrambe le card *J18*.

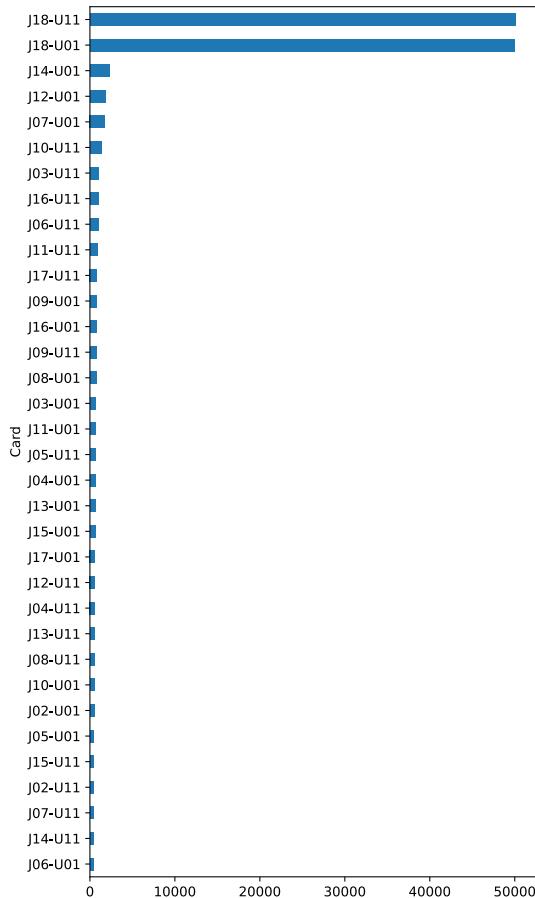


Figura 5.60: Numero di errori per nodo

A questo punto è chiaro che il **bottleneck** del sistema è rappresentato dall'intero sottosistema di I/O.

5.4.3.1 Data Manipulation

5.4.3.1.1 Coalescenza spaziale

5.4.3.1.1.1 Sensitivity Analysis Di seguito si propone la sensitivity analysis effettuata per ciascuna tipologia di card.

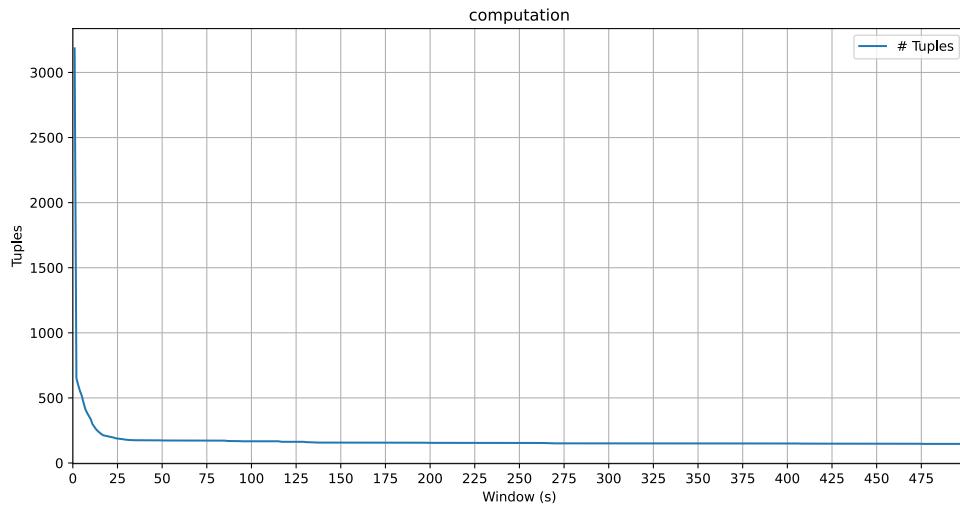


Figura 5.61: Sensitivity Analysis tipologia di card computation

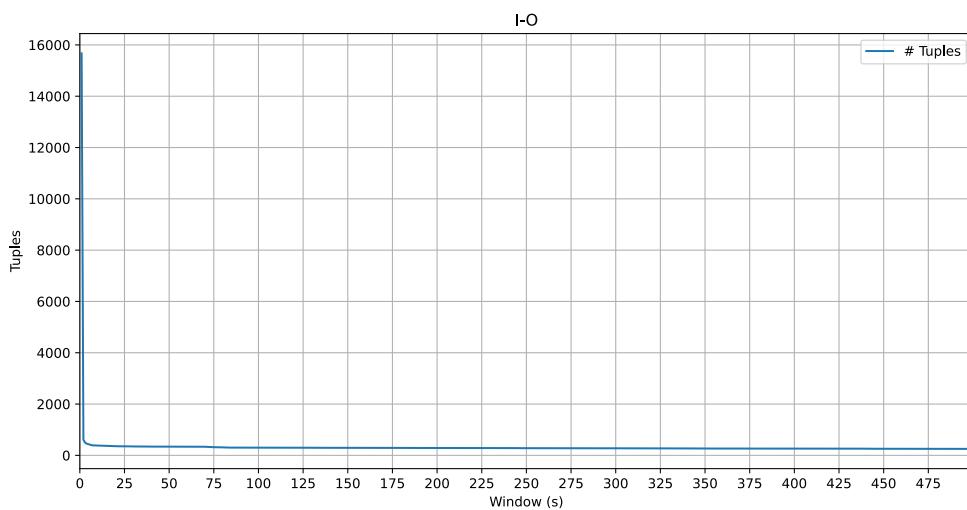


Figura 5.62: Sensitivity Analysis tipologia di card I/O

A valle della sensitivity analysis, per ogni tipologia si scelgono le finestre di coalescenza riportate in Tabella 5.12.

Tipologia	Finestra di Coalescenza
computation	25
I-O	10

Tabella 5.12: Finestre di coalescenza tipologie

5.4.3.1.1.2 Tupling Si riporta in Figura 5.13 il numero di tuple ottenuto per ciascuna tipologia eseguendo l'algoritmo di tupling con le finestre di coalescenza precedentemente fissate:

Tipologia	# Tuple
computation	188
I-O	380

Tabella 5.13: Finestre di coalescenza tipologie

5.4.3.2 Data Analysis

5.4.3.2.1 Empirical Reliability Di seguito si riportano le reliability empiriche determinate per ciascuna tipologia. Come si evince dal grafico, la tipologia meno reliable è *I-O*.

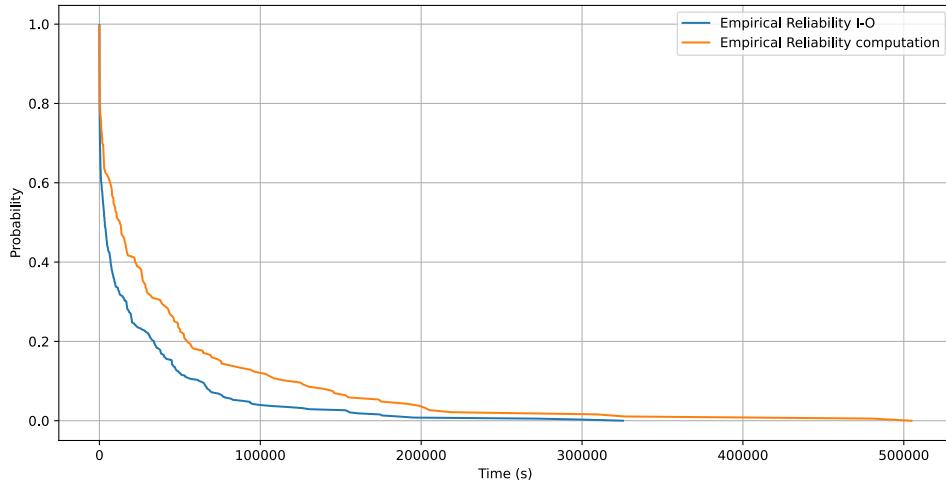


Figura 5.63: Reliability empirica

5.4.4 Nodi critici

Nella sezione corrente si ripetono le stesse operazioni eseguite in precedenza sull'intero sistema, andando però a considerare i dati relativi ai 5 nodi più critici del sistema, ovvero quelli a cui sono associate più entry del file di log. In Figura 5.64 si riporta il numero di

entry per i 5 nodi più critici, ovvero: $R71-M0-N4$, $R12-M0-N0$, $R63-M0-N2$, $R03-M1-NF$, $R63-M0-N0$. Come si evince dal grafico:

- 3 dei 5 nodi più critici sono dotati di una card di I/O;
- 2 dei nodi più critici appartengono al rack più critico, ovvero $R63$.
- a differenza di Mercury, non esiste un singolo nodo caratterizzato da un numero di entry notevolmente superiore rispetto agli altri.

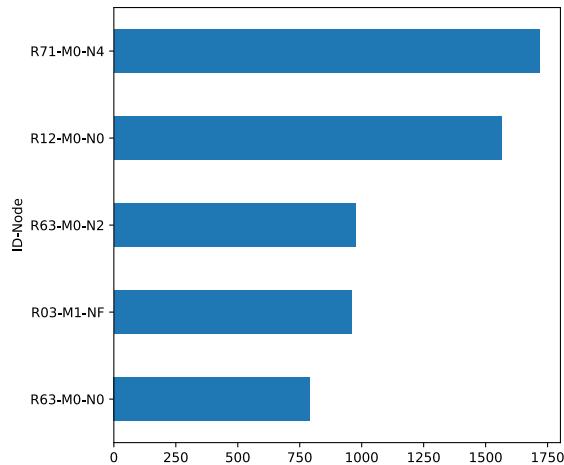


Figura 5.64: Numero di entry per nodo

5.4.4.1 Data Manipulation

5.4.4.1.1 Coalescenza temporale

5.4.4.1.1.1 Sensitivity Analysis Di seguito si propone la sensitivity analysis effettuata per ognuno dei 5 nodi più critici.

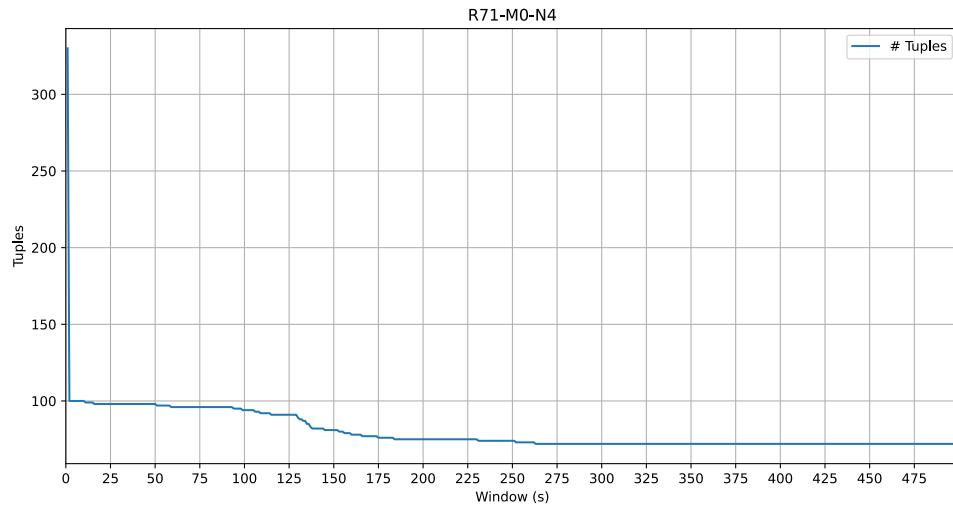


Figura 5.65: Sensitivity Analysis nodo R71-M0-N4

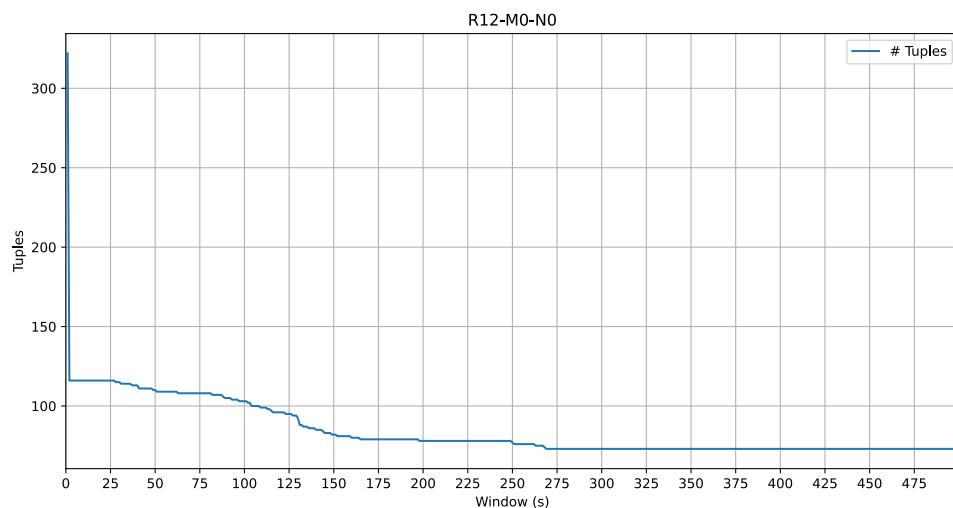


Figura 5.66: Sensitivity Analysis nodo R12-M0-N0

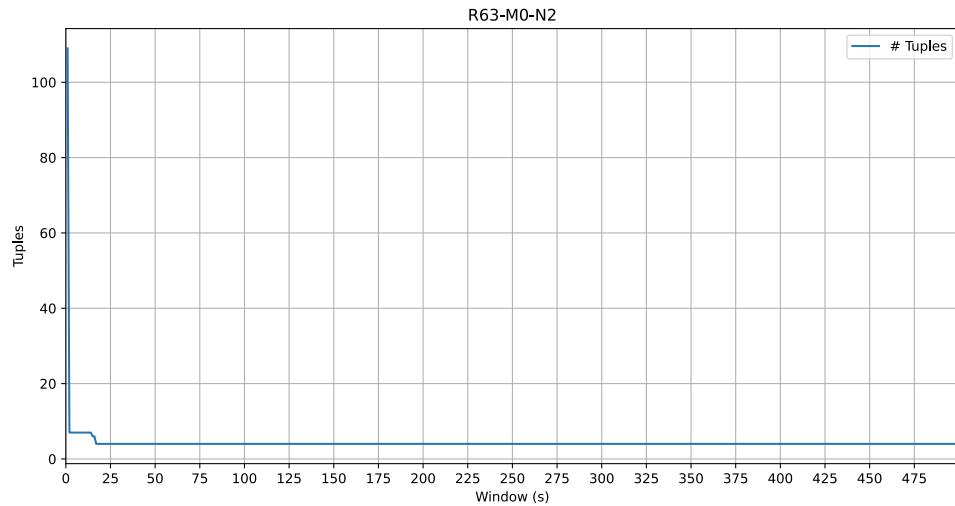


Figura 5.67: Sensitivity Analysis nodo R63-M0-N2

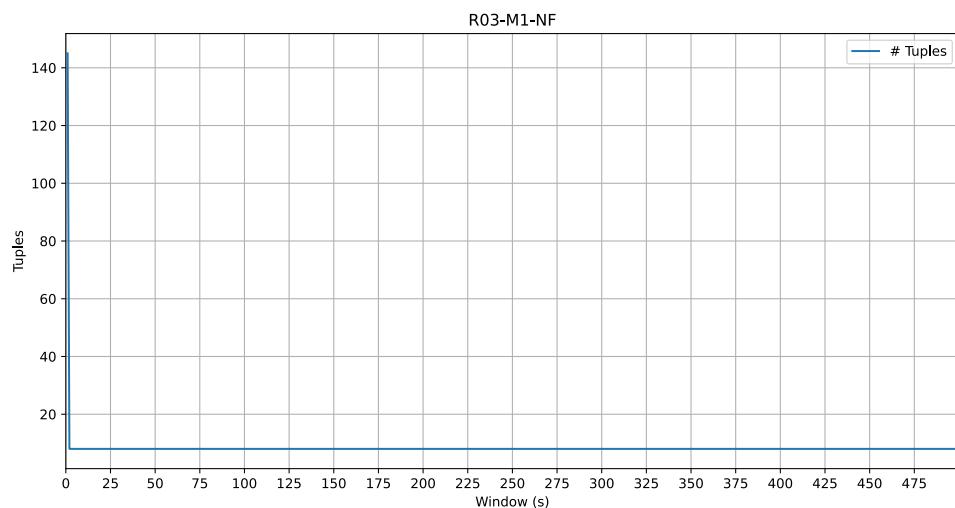


Figura 5.68: Sensitivity Analysis nodo R03-M1-NF

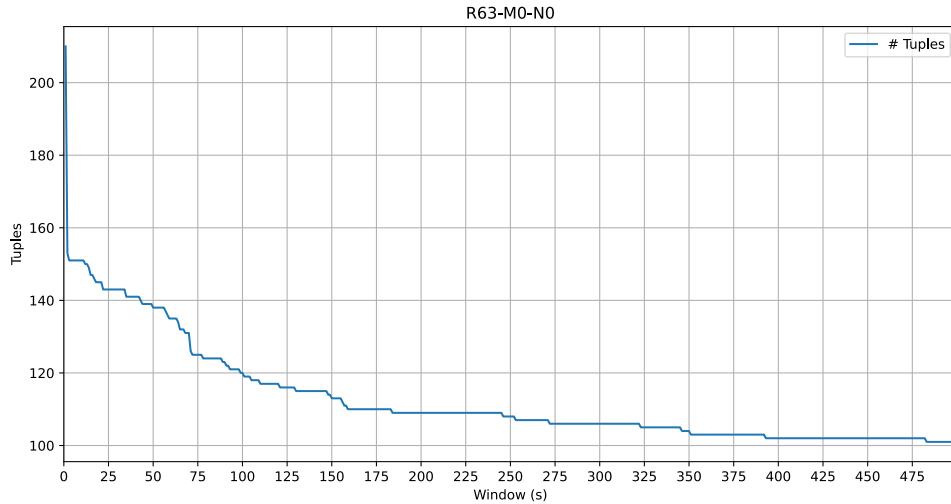


Figura 5.69: Sensitivity Analysis nodo R63-M0-N0

A valle della sensitivity analysis, per ognuno dei nodi più critici i scelgono le finestre di coalescenza riportate in Tabella 5.14.

Nodo	Finestra di Coalescenza
R71-M0-N4	175
R12-M0-N0	175
R63-M0-N2	20
R03-M1-NF	5
R63-M0-N0	175

Tabella 5.14: Finestre di coalescenza nodi critici

5.4.4.1.1.2 Tupling Si riporta in Tabella 5.15 il numero di tuple ottenuto per ciascun nodo eseguendo l'algoritmo di tupling con le finestre di coalescenza precedentemente fissate:

Nodo	# Tuple
R71-M0-N4	76
R12-M0-N0	79
R63-M0-N2	4
R03-M1-NF	8
R63-M0-N0	110

Tabella 5.15: Finestre di coalescenza nodi critici

5.4.4.2 Data Analysis

5.4.4.2.1 Empirical Reliability Di seguito si riportano le reliability empiriche determinate per i nodi più critici che presentano un numero di tuple significativo. Come si evince dal grafico, il nodo meno reliable è *R63-M0-N0*. È bene osservare che tale nodo non è quello che presenta il maggior numero di entry del log, che come mostrato in Figura 5.64 è *R71-M0-N4*. Questo può essere dovuto al fatto che molte delle entry associate a *R71-M0-N4* sono vicine nel tempo per cui sono inserite nelle stesse tuple. In altre parole, molte di queste entry sono in realtà relative allo stesso errore.

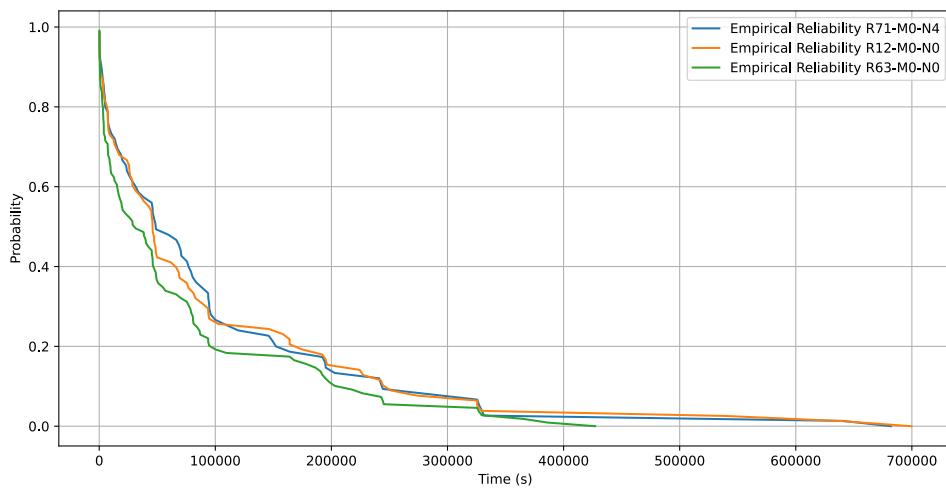


Figura 5.70: Reliability empirica

5.5 Confronto tra Mercury e Blue Gene/L

Per concludere, si evidenziano le differenze riscontrate durante la trattazione precedente tra i sistemi Mercury e Blue Gene/L.

In primo luogo è possibile affermare che il bottleneck di Mercury è rappresentato da un singolo nodo, *tg-c401*. Al contrario, il bottleneck di BGL non è rappresentato da un singolo nodo ma dall'intero sottosistema di I/O.

Inoltre, mentre gli errori di BGL sono distribuiti in maniere uniforme nello spazio e nel tempo, non è possibile affermare lo stesso nel caso di Mercury. La maggior parte delle entry relative a quest'ultimo sono infatti originate non solo sul singolo nodo *tg-401*, ma anche in un arco di tempo limitato della durata di due giorni (12/02/2007 - 14/02/2007).

In conclusione, si riportano in Figura 5.71 le reliability empiriche ottenute per i due sistemi. Dal grafico si evince che il sistema Mercury è meno reliable rispetto a Blue Gene/L.

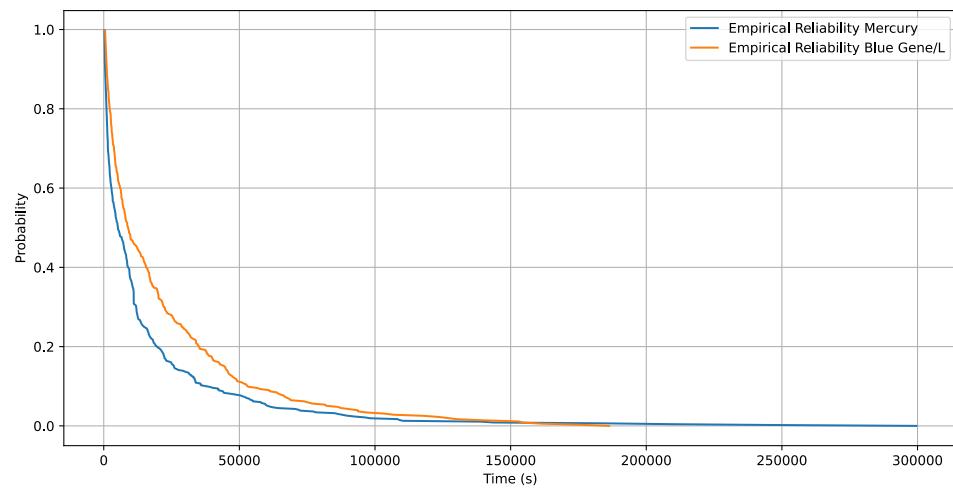


Figura 5.71: Reliability empirica

Bibliografia

- [1] SAS Institute Inc. *JMP*. Ver. 14.0. 1989-2019.
- [2] F. Pedregosa et al. “Scikit-learn: Machine Learning in Python”. In: *Journal of Machine Learning Research* 12 (2011), pp. 2825–2830.
- [3] Matthew Reid. “Reliability – a Python library for reliability engineering”. In: *Journal of Machine Learning Research* (2020).
- [4] Skipper Seabold e Josef Perktold. “statsmodels: Econometric and statistical modeling with python”. In: *9th Python in Science Conference*. 2010.