



Documentazione dei Progetti di Impianti di Elaborazione

Orlando Gian Marco M63/001326
Perillo Marco M63/001313
Russo Diego M63/001335

Indice

Benchmark	4
1.1 Traccia	4
1.2 Configurazioni	4
1.3 Risultati attesi	5
1.4 Svolgimento	6
1.4.1 Raccolta dei campioni	6
1.4.2 Calcolo del COV e della media campionaria	6
1.4.3 Stima del Sample Size	7
1.4.4 Esecuzione dei test e risultati ottenuti	8
Web Server	13
2.1 Traccia	13
2.2 Capacity Test	13
2.2.1 Configurazione e metriche	13
2.2.2 Configurazione Test Plan	14
2.2.3 Analisi dei dati e risultati ottenuti	17
2.2.4 Indice di equità	21
2.3 Workload Characterization	22
2.3.1 Raccolta dei dati per la generazione del workload	23
2.3.2 Workload Characterization	26
2.3.3 Confronto LL e LL'	30
2.4 Design of Experiment	33
2.4.1 Configurazione	33
2.4.2 Analisi dell'importanza	35
2.4.3 Analisi della significatività	36
PCA e Clustering	39

3.1 Traccia	39
3.2 Data Understanding e Pre-processing dei dati	39
3.3 Principal Component Analysis (PCA)	40
3.4 Clustering	42
3.5 Caratterizzazione del workload sintetico	45
Regessione Lineare	47
4.1 Esercizio 1	47
4.1.1 Traccia	47
4.1.2 Svolgimento	47
4.2 Esercizio 2	50
4.2.1 Traccia	50
4.2.2 Svolgimento	50
4.3 Esercizio 3	53
4.3.1 Traccia	53
4.3.2 Svolgimento	53
4.4 Esercizio 4	65
4.4.1 Traccia	65
4.4.2 Svolgimento	65
Reliability	76
5.1 Esercizio 1	76
5.1.1 Traccia	76
5.1.2 Svolgimento	76
5.2 Esercizio 2	78
5.2.1 Traccia	78
5.2.2 Svolgimento	79
5.3 Esercizio 3	81
5.3.1 Traccia	81

5.3.2 Svolgimento	82
5.4 Esercizio 4	84
5.4.1 Traccia	84
5.4.2 Svolgimento	84
5.5 Esercizio 5	88
5.5.1 Traccia	88
5.5.2 Svolgimento	90
FFDA	93
6.1 Traccia	93
6.2 Field Failure Data Analysis (FFDA)	93
6.3 Mercury	94
6.3.1 Data Manipulation	95
6.3.2 Data Analysis	97
6.4 Blue Gene/L	102
6.4.1 Data Manipulation	103
6.4.2 Data Analysis	104
6.5 Confronto dell'affidabilità di Mercury e Blue Gene	109
6.6 Ulteriori analisi: i bottleneck di Mercury e Blue Gene	109

Capitolo 1

Benchmark

1.1 Traccia

Il simulatore n-body realizza la simulazione dell'evoluzione di un sistema di N corpi sotto l'influenza della forza gravitazionale. Esso è un buon benchmark poiché stressa:

- *il sottosistema di floating point;*
- *le chiamate ricorsive;*
- *lo stack.*

Utilizzare il simulatore n-body per comparare più sistemi con lo stesso sistema operativo e differenti processori.

1.2 Configurazioni

Il System Under Test (SUT) di tale esercizio sarà il processore. In particolare, considerando ciò che il benchmark n-body va a stressare, i Component Under Study (CUS) della nostra analisi saranno sicuramente la floating-point unit che gestisce le operazioni a virgola mobile (ed è uno specifico sotto-componente della ALU) e lo Stack Management Unit (SMU) che, invece, gestisce le chiamate ricorsive.

Di seguito, dunque, vengono innanzitutto riportate le configurazioni delle macchine a nostra disposizione su cui è stata effettuata l'analisi.

La macchina 1 presenta le seguente caratteristiche:

- Processore: 11th Gen Intel(R) Core(TM) i7-11800H - 2.30 GHz
- Memoria RAM: 16 GB 3200Mhz
- Sistema Operativo: Windows 11

La macchina 2 presenta le seguente caratteristiche:

- Processore: AMD Ryzen 7 5800H - 3.20 GHz
- Memoria RAM: 16 GB 3200Mhz
- Sistema Operativo: Windows 11

La macchina 3 presenta le seguenti caratteristiche:

- Processore: Intel(R) Core(TM) i7-8750H CPU - 2.20 GHz
- Memoria RAM: 16 GB 2667MHz
- Sistema Operativo: Windows 11

1.3 Risultati attesi

Considerando le configurazioni presentate, ci aspettiamo che i tempi di esecuzione dei processori testati siano statisticamente diversi tra loro. Analizzeremo le ragioni che ci hanno portato a questa conclusione confrontando le diverse configurazioni a coppie:

- **Macchina 1 - Macchina 2:** entrambe le macchine utilizzano processori relativamente recenti rilasciati nell'anno 2021 e presentano lo stesso numero di core e thread (8 core e 16 thread). Tuttavia, appartengono a famiglie diverse (rispettivamente Intel Core i7 e AMD Ryzen 7) di produttori diversi, pertanto ci aspettiamo che i dati siano statisticamente differenti;
- **Macchina 1 - Macchina 3:** entrambe le macchine utilizzano processori della famiglia Intel Core i7 ma di generazioni diverse (rispettivamente, undicesima generazione ed ottava generazione) rilasciate a distanza di circa 4 anni. Questo è un intervallo di tempo importante dal punto di vista tecnologico che ha visto l'introduzione di una nuova architettura caratterizzata da un upgrade da 6 ad 8 core e da 12 a 16 thread. Pertanto, ci aspettiamo che i dati siano statisticamente differenti;
- **Macchina 2 - Macchina 3:** in questo ultimo confronto, le macchine utilizzano non solo processori di diverse case produttrici, ma anche rilasciati a circa 4 anni di distanza l'uno dall'altro. In generale questi processori non presentano considerevoli caratteristiche comuni, dunque, a maggior ragione rispetto ai confronti precedenti, ci aspettiamo che i dati siano statisticamente differenti.

1.4 Svolgimento

Nei seguenti sotto-paragrafi verranno descritti i passaggi effettuati al fine di realizzare il benchmarking in esame.

1.4.1 Raccolta dei campioni

In questa fase abbiamo raccolto i campioni dei diversi processori effettuando delle misurazioni dirette in maniera indipendente. E' stata garantita l'indipendenza dei campioni attraverso il riavvio delle macchine (al fine di ridurre l'influenza della cache sulle prestazioni del processore) e l'attesa di un minuto dopo il riavvio (al fine di escludere l'influenza che eventuali operazioni di boot avrebbero potuto avere sulle prestazioni del processore) prima di effettuare una nuova misurazione.

Per ogni esperimento effettuiamo 5 ripetizioni. Sono stati condotti 30 esperimenti indipendenti con un numero di corpi pari a 50.000 ed ulteriori 30 esperimenti indipendenti con un numero di corpi pari a 300.000 in modo da avere evidenze per il confronto delle 3 macchine sia con "basso carico" che con "alto carico". Per entrambi i due tipi di carico, abbiamo scelto di effettuare 30 esperimenti in modo da poter rispettare le ipotesi del Teorema del Limite Centrale. Il Teorema del Limite Centrale, infatti, afferma che se si considerano dei campioni composti da osservazioni IID, ovvero delle osservazioni indipendenti ed identicamente distribuite, ed il numero di campioni n è molto grande ($n \geq 30$), posso concludere che la distribuzione delle medie campionarie si comporta come una distribuzione normale con:

- media uguale alla media della popolazione μ ;
- deviazione standard uguale a $\frac{\sigma}{\sqrt{n}}$.

Non abbiamo ritenuto necessario effettuare analisi sulla distribuzione dei dati proprio perché il Teorema del Limite Centrale ci garantisce che la distribuzione sia normale.

1.4.2 Calcolo del COV e della media campionaria

Al fine di iniziare la fase di analisi dei dati raccolti, abbiamo ritenuto opportuno innanzitutto calcolare la media e la deviazione standard dei campioni a nostra disposizione in modo da poter calcolare il cosiddetto **Coefficiente di Variazione** (COV) che, per definizione, indica di quanto la media si discosta dal campione ed è dato dal seguente rapporto:

$$COV = \frac{\text{deviazione standard}}{\text{media}}$$

Ora, se il valore ottenuto effettuando tale rapporto risulta essere minore di 0.5, allora questo ci indica che possiamo scegliere ancora la media al fine di effettuare le nostre analisi,

altrimenti dovremmo scegliere la mediana in quanto il risultato ottenuto ci indicherebbe che uno o più outlier hanno modificato troppo la deviazione standard.

Nella seguente tabella si riportano i risultati ottenuti:

- *50.000 corpi*

	Media	Deviazione Standard	COV
Macchina 1	8553,69	1039,68	0,12
Macchina 2	10171,79	518,58	0,05
Macchina 3	15813,11	4444,06	0,28

- *300.000 corpi*

	Media	Deviazione Standard	COV
Macchina 1	55490,1	8816,5	0,16
Macchina 2	59596,7	2606,76	0,044
Macchina 3	87697,49	22839,33	0,26

Sia nel caso di 50.000 corpi che in quello di 300.000 corpi, il coefficiente di variazione è risultato essere ben al di sotto rispetto alla soglia di 0.5 e, per questa ragione, è stato possibile - come detto - calcolare la media campionaria di ogni osservazione anziché la mediana al fine di poter proseguire con l'analisi.

1.4.3 Stima del Sample Size

A questo punto, abbiamo deciso di effettuare una stima del Sample Size, ossia del numero minimo di campioni indipendenti necessari per poter approssimare la media e la varianza del campione a quelli della popolazione. Tale operazione risulta necessaria in quanto noi non conosciamo tali parametri rispetto all'intera popolazione ma dobbiamo inferirli dai campioni a nostra disposizione.

Per fare ciò, dunque, abbiamo scelto un valore di errore del 10% rispetto alla media ed un intervallo di confidenza del 95%. La scelta del valore di errore è stata guidata proprio dal fatto che se avessimo scelto un valore di errore più stringente (ad esempio, del 5% rispetto alla media) la macchina con la configurazione “più datata” (Macchina 3) avrebbe richiesto un

numero di campioni indipendenti troppo elevato (superiore anche alle centinaia sia nel caso di 50.000 corpi che in quello di 300.000 corpi).

La formula utilizzata al fine di poter stimare tale numero minimo di campioni necessari è la seguente:

$$n = \left(\frac{z_{\alpha/2} \cdot \sigma}{E} \right)^2$$

Notiamo che - come detto - la deviazione standard della popolazione σ è stata sostituita con la deviazione standard del campione S . Inoltre, il valore di $z_{\alpha/2} = 1.96$. Tale valore è ottenuto dall'apposita tabella di riferimento relativa al calcolo dei quantili normali.

Di seguito riportiamo i risultati ottenuti sia per il caso di 50.000 corpi che nel caso di 300.000 corpi:

	50.000 corpi	300.000 corpi
Macchina 1	6	10
Macchina 2	1	1
Macchina 3	30	26

Considerando che i risultati ottenuti hanno evidenziato una dimensione campionaria minore o, al massimo, uguale al numero di campioni indipendenti da noi già raccolti, abbiamo ritenuto ridondante effettuare osservazioni aggiuntive e, per questo, abbiamo deciso di proseguire l'analisi servendoci dei campioni già a nostra disposizione.

1.4.4 Esecuzione dei test e risultati ottenuti

Terminato il calcolo del Sample Size, siamo passati alla comparazione tra le 3 macchine. Innanzitutto, abbiamo caricato i dataset (uno per il caso di 50.000 corpi e l'altro per il caso di 300.000 corpi) nel software di analisi statistica JMP. Tali dataset - come spiegato in precedenza nell'apposito paragrafo di questa documentazione - sono stati ottenuti calcolando le medie campionarie di ogni singola osservazione. Entrambi i dataset, dunque, sono composti da 30 righe e 3 colonne (una riga per ogni osservazione ed una colonna per ogni macchina a nostra disposizione).

Notiamo, inoltre, che siccome non esiste una corrispondenza uno ad uno tra le coppie di campioni ed è stata assicurata l'indipendenza dei campioni prelevati, le osservazioni che andremo ad effettuare durante la nostra analisi sono di tipo "unpaired".

Dunque, caricando i dataset in JMP ed utilizzando l'apposito strumento di analisi, otteniamo i seguenti risultati:

- 50.000 corpi

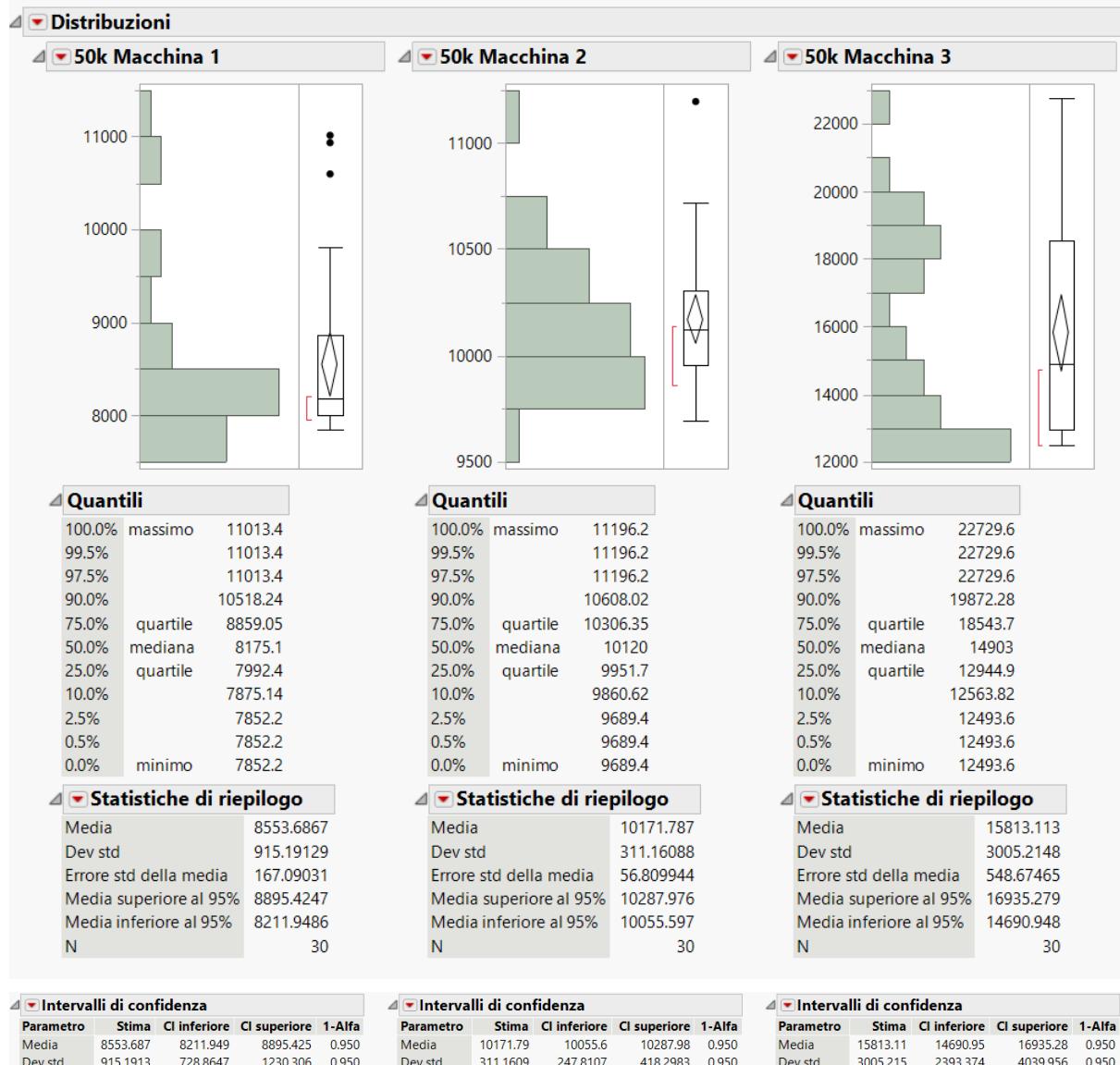


Figura 1.1: Analisi macchine 50.000 corpi

- 300.000 corpi



Figura 1.2: Analisi macchine 300.000 corpi

Avendo scelto un livello di confidenza del 95%, abbiamo innanzitutto effettuato il calcolo degli intervalli di confidenza secondo la seguente formula:

$$\hat{X} - \frac{\sigma \cdot z_{\alpha/2}}{\sqrt{n}} \leq \mu \leq \hat{X} + \frac{\sigma \cdot z_{\alpha/2}}{\sqrt{n}}$$

Dunque, i risultati ottenuti per entrambi i casi sono i seguenti:

- 50.000 corpi

$$T_1 = 8553,69 \pm 341,74$$

$$T_2 = 10171,79 \pm 116,19$$

$$T_3 = 15813,11 \pm 1122,16$$

Da una semplice analisi visiva, dunque, notiamo subito che gli intervalli di confidenza non si sovrappongono in nessuno dei tre possibili confronti. Per questo, dunque, **possiamo subito escludere** - almeno nel caso di “basso carico” - l’appartenenza dei campioni alla stessa popolazione d’origine e, per questo, possiamo dire che **tali campioni risultano essere statisticamente differenti**.

- *300.000 corpi*

$$T_1 = 55999,17 \pm 3014,33$$

$$T_2 = 59596,7 \pm 596,17$$

$$T_3 = 87697,49 \pm 6795,72$$

Per quanto riguarda, invece, il caso di 300.000 corpi notiamo che - come prima - **per quanto riguarda i confronti con la macchina 3** si vede subito che gli intervalli di confidenza non si sovrappongono e, per questo, **possiamo già affermare che i campioni risultano essere statisticamente differenti**.

Per ciò che concerne, invece, il confronto tra la macchina 1 e la macchina 2 è necessario fare alcune considerazioni aggiuntive in quanto gli intervalli di confidenza (seppur di poco) risultano essere sovrapposti ma la media di una macchina non risulta essere inclusa nell’intervallo di confidenza della reciproca macchina e, per questo, il test visivo non è conclusivo e, dunque, sarà necessario effettuare un t-test.

Tra i grafici a nostra disposizione, ci siamo soffermati innanzitutto sui box plot riportati sopra. Il box plot, infatti, permette di rendersi subito conto se le varianze sono simili. Tale informazione sarà utile per poter scegliere il corretto test d’ipotesi da applicare per il confronto delle due macchine. In particolare, a partire da tale analisi visiva è possibile subito notare che il box plot della prima macchina risulta essere molto più grande rispetto a quello della seconda e ciò implica che la varianza di tale macchina è molto più elevata rispetto a quella della seconda macchina. Per questo motivo, non è stato necessario effettuare un test alternativo a quello visivo (come, ad esempio, il *vartest2(x,y)* disponibile in Matlab).

Dunque, siccome le due varianze sono - appunto - diverse tra loro, abbiamo scelto di effettuare il **Two-Sample t-test unpoole**d. Su Matlab tale test viene realizzato utilizzando il comando *ttest2(x,y)* con l’aggiunta dei parametri *vartype* e *unequal*. Questi due parametri vengono aggiunti in modo che non si faccia nessuna assunzione sul fatto che le varianze debbano essere uguali. In tale test, l’ipotesi nulla è quella che afferma che i due vettori in

input (ossia le medie campionarie delle 30 osservazioni effettuate sulle due macchine) provengano da popolazioni con la stessa media.

Di seguito, dunque, viene riportato il risultato del Two-Sample t-test unpooledd effettuato:

```
>> [h, p] = ttest2(macchina1,macchina2, 'Vartype','unequal')

h =
1

p =
0.0228
```

Figura 1.3: Two-Sample t-test

Dal risultato ottenuto su Matlab è possibile evincere che l'ipotesi nulla viene rigettata ($h=1$). Inoltre, il p-value presenta un valore molto inferiore al valore di 0.05 (tale valore dipende dall'intervallo di confidenza scelto che - ricordiamo - è del 95%) e ciò rafforza proprio il fatto che l'ipotesi nulla venga rigettata. In conclusione, dunque, otteniamo che anche in quest'ultimo caso **i campioni misurati sono statisticamente differenti e provengono da distribuzioni diverse**.

I risultati ottenuti in tutti i confronti effettuati risultano essere, quindi, in linea con quelli che ci aspettavamo descritti nell'apposito paragrafo *"Risultati attesi"* di questa documentazione sia nel caso di "basso carico" che in quello di "alto carico".

Capitolo 2

Web Server

2.1 Traccia

Analizzare le performance di un web server, effettuando le seguenti attività:

- **Capacity Test:** valutazione delle performance del sistema al variare del carico di lavoro imposto;
- **Workload Characterization:** estrapolare un workload sintetico a partire da un workload reale e verificarne la significatività statistica dei rispettivi workload di basso livello;
- **Design of Experiment:** verifica dell'influenza dei fattori sul response time del sistema.

2.2 Capacity Test

Il Capacity Test è uno dei test più utilizzati nell'ambito della Performance Analysis. Tramite questo test, infatti, è possibile trovare il punto di funzionamento ideale del System Under Test (SUT, ossia il sistema che si intende testare attraverso uno stimolo esterno). In generale, un sistema non scala linearmente le proprie performance all'aumentare del carico e, per questo, trovare tale punto di funzionamento ideale può essere utile al fine di effettuare attività di capacity planning, capacity management ed anche performance tuning.

2.2.1 Configurazione e metriche

Il sistema testato è un **Web Server Apache2** eseguito su una macchina virtuale Oracle VM VirtualBox V 6.1 con sistema operativo Linux (in particolare, la distribuzione Linux installata è **Ubuntu 20.04**). La macchina host su cui è stata fatta eseguire la macchina virtuale presenta le seguenti risorse hardware:

- **Processore:** 11th Gen Intel(R) Core(TM) i7-11800H - 2.30 GHz;
- **Memoria RAM:** 16 GB 3200Mhz;
- **Sistema Operativo:** Windows 11.

Notiamo, inoltre, che nel caso di test di capacità relativo ad un Web Server, le metriche tipicamente utilizzate per caratterizzare le sue performance sono:

- **Response Time**: intervallo di tempo che intercorre tra la richiesta di una risorsa ed il completamento della risposta del sistema;
- **Throughput**: il numero di richieste per unità di tempo che il server è in grado di soddisfare.

Infine, notiamo che - considerando le metriche fin qui elencate - al fine di effettuare il test di capacità, dovremo individuare due punti notevoli per il funzionamento del web server stesso:

- **Knee Capacity**: è il punto in cui si ha il miglior trade-off tra throughput e response time ed infatti corrisponde al punto in cui il rapporto tra le due metriche è massimo. Tale punto, inoltre, corrisponde al cosiddetto punto di **Power**;
- **Usable Capacity**: è il punto in cui il throughput arriva al massimo possibile ed il tempo di risposta è altissimo (seppur non vengano superati i vincoli imposti proprio sul tempo di risposta). Superato questo punto, il sistema inizia a non funzionare più correttamente.

2.2.2 Configurazione Test Plan

Per effettuare l'analisi del capacity test è stato elaborato un Test Plan utilizzando il tool Apache JMeter. Il componente fondamentale di un test realizzato con tale tool è il **Thread Group**, necessario per simulare le richieste degli utenti verso il server. Un Thread Group è costituito da una serie di parametri che abbiamo impostato nel seguente modo:

- **Numero di threads: 50** (simula l'esatto numero di utenti che effettuano le richieste);
- **Ramp-up period: 1 secondo** (è il tempo di avvio di un thread e ci consente, dunque, di schedulare l'avvio dei vari thread);
- **Loop count: infinito** (indica il numero di volte che un thread deve effettuare una richiesta, in questo caso non abbiamo impostato un limite di richieste per ogni thread);
- **Duration: 300 secondi** (indica la durata dell'osservazione).

The screenshot shows the configuration of a Thread Group in JMeter. The 'Name' field is set to 'Thread Group'. Under 'Action to be taken after a Sampler error', the 'Continue' option is selected. In the 'Thread Properties' section, the 'Number of Threads (users)' is set to 50, 'Ramp-up period (seconds)' is 1, and 'Loop Count' is set to 'Infinite'. Several checkboxes are checked: 'Same user on each iteration', 'Specify Thread lifetime', and 'Duration (seconds)' is set to 300. The 'Startup delay (seconds)' field is empty.

Figura 2.1: Configurazione del Thread Group

Un altro elemento fondamentale da aggiungere al Test Plan sono i **Sampler**. I Sampler da noi utilizzati sono di tipo *HTTP Request* ed indicano a JMeter di inviare una serie di richieste al server ed attendere una risposta dal server stesso. In particolare, abbiamo aggiunto un Sampler per ogni risorsa presente nel nostro Web Server. Notiamo che abbiamo considerato 3 tipologie diverse di risorse in base alla loro dimensione: 7 file “*Small*” di dimensione compresa tra 1KB e 3KB, 6 file “*Medium*” di dimensione compresa tra 1MB e 3MB ed, infine, 6 file “*Large*” di dimensione compresa tra 10MB e 16MB. Tali risorse sono state memorizzate nel path di default del Web Server */var/www/html/* in modo da non dover specificare il path della risorsa stessa all’interno del componente di JMeter (come vedremo a breve, è bastato inserire solo il nome della risorsa).

I parametri impostati per i Sampler delle richieste HTTP sono i seguenti:

- **Nome della richiesta;**
- **Protocollo:** il protocollo è HTTP, l’IP viene assegnato da Virtual Box mentre la porta è quella standard di HTTP (80);
- **Tipo di richiesta:** le richieste vengono effettuate con metodo GET;
- **Path della risorsa:** è stato inserito solo il nome della risorsa in quanto - come detto - tutte le risorse sono state memorizzate nel path di default.

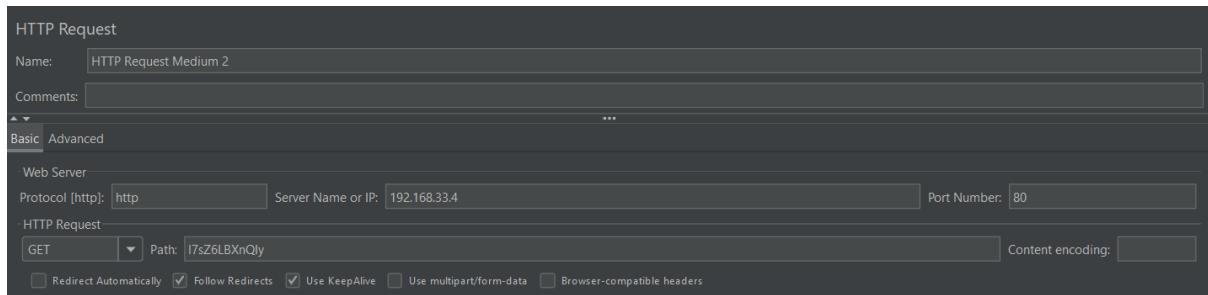


Figura 2.2: Configurazione del Sampler HTTP Request

Un altro componente fondamentale è il **Controllore**. Tale componente ci permette di dare una logica un po' più complessa al nostro esperimento. In particolare, abbiamo utilizzato un **Random Order Controller**. Tramite tale elemento, ogni volta che viene attivato un thread, quest'ultimo richiede tutte le risorse in ordine casuale. Le risorse, quindi, vengono sempre richieste tutte ma è l'ordine in cui queste vengono richieste ad essere casuale (a differenza, ad esempio, di un Random Controller in cui, invece, viene chiesta randomicamente una sola risorsa). Facciamo ciò al fine di rendere il piano di test quanto più realistico possibile.

Un altro elemento chiave di un Test Plan è il **Timer**, in questo caso è stato utilizzato il Constant Throughput Timer (CTT). Il CTT stabilisce il throughput che tutti i thread in un gruppo devono raggiungere impostando il parametro "*all active thread in current thread group*", ovvero tutti i thread di quel gruppo devono fare assieme un certo numero di richieste al minuto. Tutti i thread vengono avviati contemporaneamente ed il numero di richieste effettuate da ciascun thread può variare. Durante l'analisi, il valore del CTT è stato man mano aumentato, passando gradualmente da 500 fino ad arrivare a 3500. Infine, i risultati ottenuti sono stati raccolti tramite due tipi di **Listener**: il Summary Report ci ha permesso di raccogliere - tra gli altri - i valori di throughput ottenuti mentre il Simple Data Writer quelli relativi all'elapsed time.

Per completezza, riportiamo di seguito la struttura del Test Plan e tutti i componenti che lo compongono descritti fin qui:



Figura 2.3: Struttura del Test Plan

2.2.3 Analisi dei dati e risultati ottenuti

I dati sono stati raccolti effettuando 3 osservazioni indipendenti per ogni valore di CTT. Al fine di assicurare l'indipendenza di tali osservazioni, la macchina virtuale che ospita il Web Server è stata riavviata tra un'osservazione e l'altra.

Una volta raccolti i valori di ogni osservazione, considerando un certo valore di CTT, il valore di Response Time rappresentativo è stato calcolato effettuando innanzitutto la mediana della colonna Elapsed dei 3 report ottenuti dalle 3 osservazioni effettuate. Fatto ciò, abbiamo calcolato il coefficiente di variazione in modo da scegliere se utilizzare la media o la

mediana dei 3 valori ottenuti. Per tutti i differenti valori di CTT è stata sempre selezionata la media.

Per quanto riguarda il Throughput, invece, ne abbiamo raccolto direttamente il valore dai 3 Summary Report ottenuti da ogni osservazione e ne abbiamo calcolato la media in modo da ottenere un valore rappresentativo per ogni CTT considerato.

Fatto ciò, abbiamo utilizzato Matlab per disegnare i grafici rispettivamente di Response Time, Throughput e Power (definita come rapporto tra Throughput e Response Time). Tramite tali grafici è stato possibile individuare i punti di Knee Capacity ed Usable Capacity precedentemente descritti.

Riportiamo, di seguito, i grafici ottenuti:

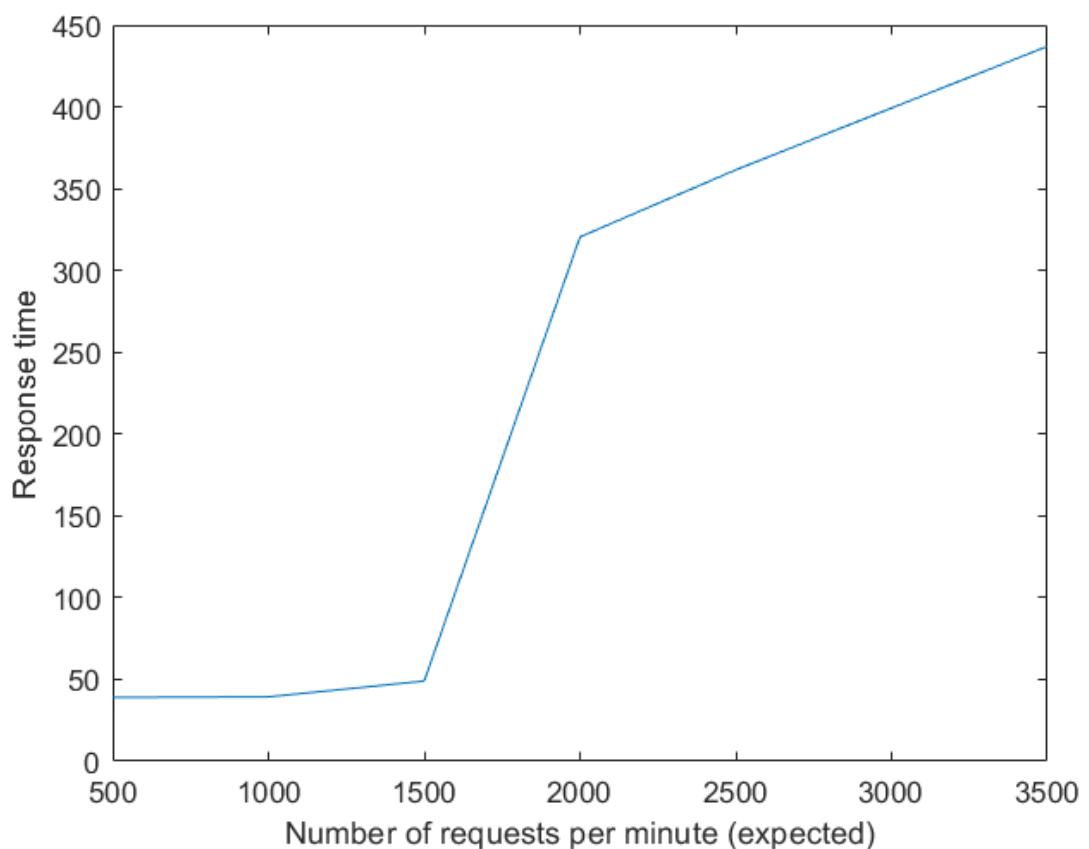


Figura 2.4: Response Time

Per quanto riguarda il tempo di risposta, notiamo che all'inizio presenta uno stretch molto basso e poi, arrivato ad un valore di 1500 richieste al minuto, tende ad aumentare esponenzialmente. Notiamo, inoltre, che tale andamento esponenziale termina una volta superate le 2000 richieste al minuto in quanto abbiamo notato una saturazione anche lato

client che non riesce ad inviare il numero di richieste attese (seppur il valore di Response Time continui comunque ad aumentare).

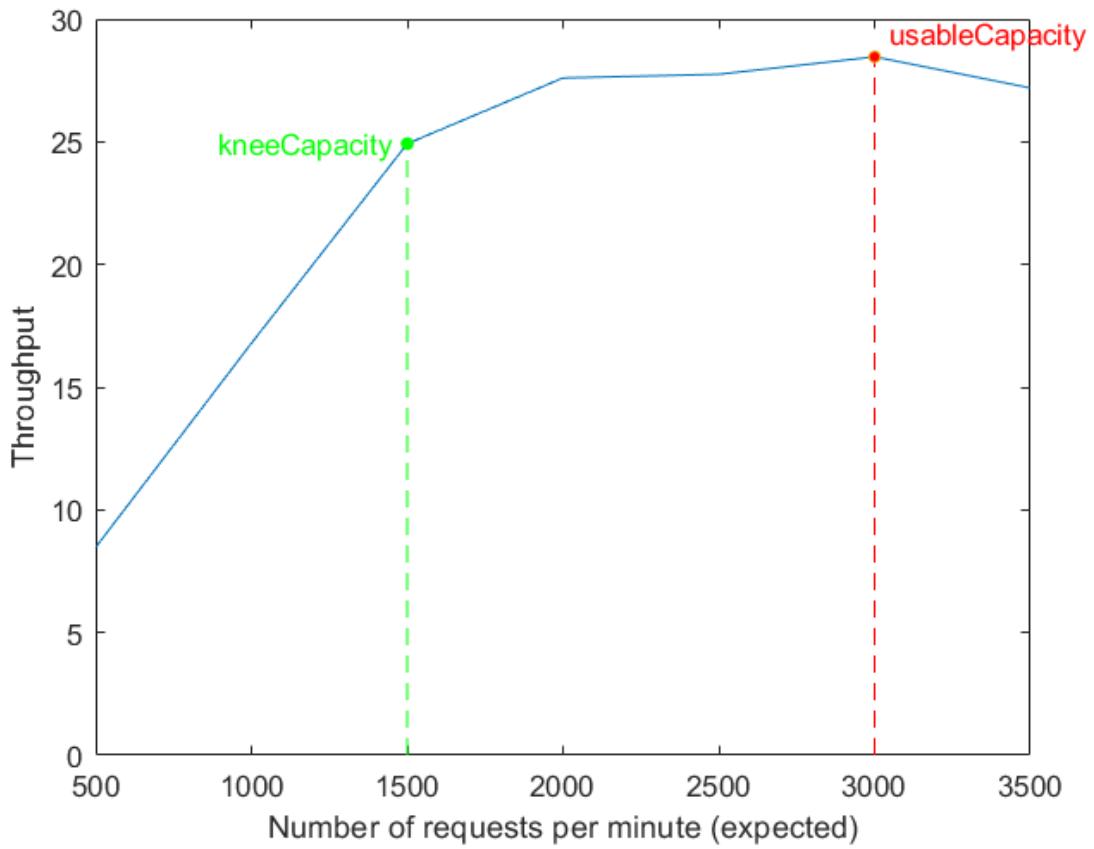


Figura 2.5: Throughput

Per quanto riguarda, invece, l'analisi del grafico relativo al throughput, notiamo che all'inizio il suo valore cresce con una certa velocità per poi tendere a stabilizzarsi. Arrivati, poi, ad un valore di CTT di 3000 richieste al minuto, il throughput inizia a decrescere: tale comportamento ci ha permesso di individuare facilmente il punto di **Usable Capacity**, fissato a **3000 richieste al minuto**.

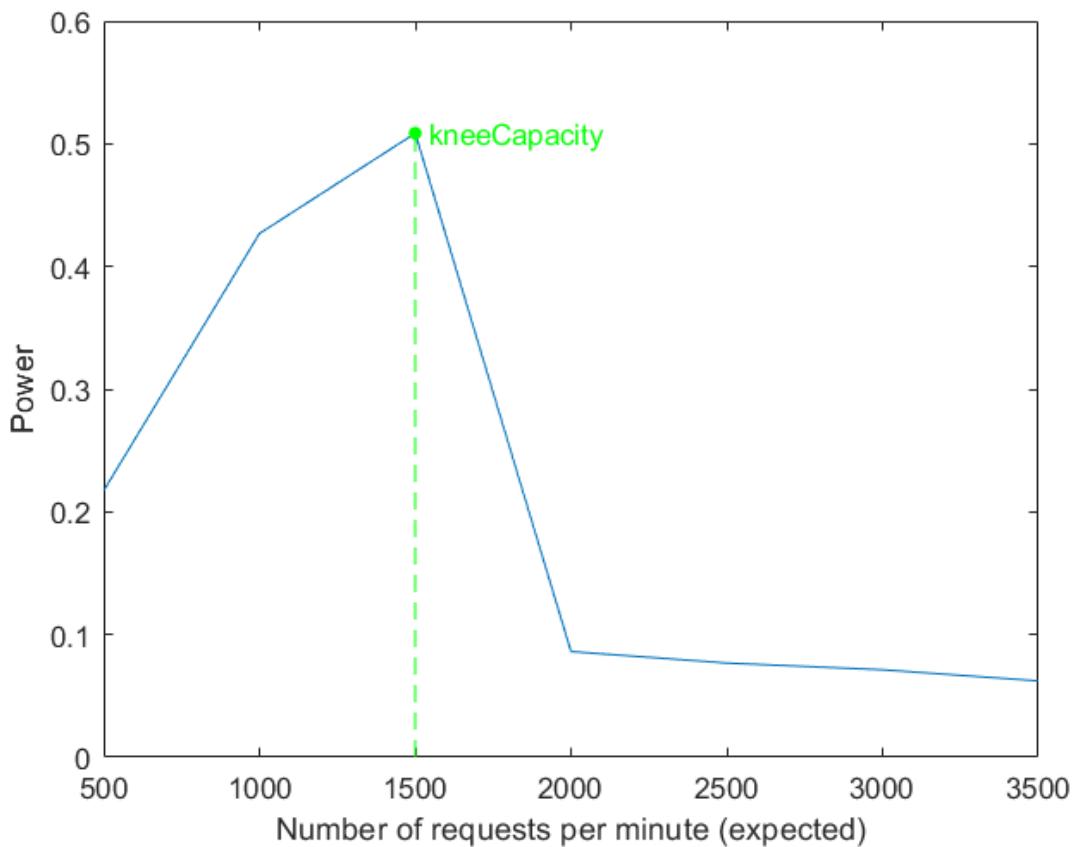


Figura 2.6: Power

Infine, il grafico della Power risulta essere utile al fine di individuare il punto di Knee Capacity. Tale punto, infatti, è individuato dal valore massimo della curva ottenuta effettuando il rapporto tra il throughput ed il response time. Come evidenziato anche dai grafici, il punto di **Knee Capacity** è risultato essere posto ad un valore di **1500 richieste al minuto**. In tale punto, infatti, notiamo che il sistema presenta un alto throughput ed un basso tempo di risposta. Ovviamente non conviene mantenere il sistema sempre al massimo throughput possibile e quindi non bisogna mai farlo lavorare nel punto di “*Usable Capacity*”. Il sistema, infatti, dovrebbe lavorare sempre nel punto di “*Knee Capacity*” perché in tale punto si preserva un buon throughput ed un buon tempo di risposta, ma non solo, infatti abbiamo anche buoni margini di “elasticità” che possono aiutare il sistema nel momento in cui può incorrere in picchi di carico ed avere, in questo modo, un sistema che sia sempre abbastanza performante.

2.2.4 Indice di equità

L'indice di equità (Fairness Index) indica quanto equamente si sta distribuendo una risorsa su più utenti. Nello specifico, quando la fairness è uguale ad 1 il sistema si sta comportando nella maniera più equa possibile. Quando, invece, la fairness si allontana da 1 fino ad arrivare a 0, il sistema risulta essere sempre “meno equo”.

Tramite JMeter, dunque, abbiamo realizzato un nuovo Test Plan costituito da 3 Thread Group differenti. Tali Thread Group (costituiti da 50 utenti ciascuno), richiedono al Web Server in maniera concorrente le 19 risorse precedentemente descritte. In particolare, il CTT dei 3 Thread è stato impostato come segue:

- Thread Group 1: 200 richieste/minuto;
- Thread Group 2: 1000 richieste/minuto;
- Thread Group 3: 2000 richieste/minuto.

Tali valori rappresentano, dunque, il throughput “desiderato” (*fair*).

Ad ogni Thread Group è stato associato un Summary Report che ci ha consentito di raccogliere il parametro di throughput misurato durante le osservazioni. Sono stati, dunque, raccolti i seguenti valori di throughput:

- Thread Group 1: 248,33 richieste/minuto;
- Thread Group 2: 710,78 richieste/minuto;
- Thread Group 3: 858,5 richieste/minuto.

Calcolando il rapporto tra throughput misurati e quelli “fair” otteniamo, invece, i seguenti risultati:

- Thread Group 1: 1,242 richieste/minuto;
- Thread Group 2: 0,71 richieste/minuto;
- Thread Group 3: 0,429 richieste/minuto.

Infine, dunque, è possibile calcolare l'indice di fairness secondo la seguente formula:

$$f(x_1, x_2, \dots, x_n) = \frac{(\sum_{i=1}^n x_i)^2}{n \sum_{i=1}^n x_i^2} = \frac{(1,242 + 0,71 + 0,429)^2}{3(1,242^2 + 0,71^2 + 0,429^2)} = 0,847$$

Possiamo concludere, quindi, che il Web Server risulta essere fair con i Thread Group caratterizzati da un CTT basso. Appena aumentiamo il valore del CTT all'interno di un certo Thread Group, invece, il throughput che è stato misurato diventa sempre più basso rispetto a

quello assegnato su JMeter al Thread Group stesso evidenziando il fatto che il Web Server si comporterà in maniera sempre meno equa con valori di CTT più elevati.

2.3 Workload Characterization

La workload characterization mira a comprendere e definire le proprietà statistiche del workload. Attraverso un'analisi approfondita, i parametri che descrivono gran parte della variabilità del dataset possono essere estratti e utilizzati per caratterizzare il workload in modo sintetico. Il processo di workload characterization si compone di diverse fasi:

- **Fase 1** → dopo aver creato il workload reale, esso viene utilizzato per testare il SUT e vengono raccolti due tipi di parametri, i parametri di basso livello (LL) ed i parametri di alto livello (HL). I parametri di alto livello sono tutti quei parametri che sono stati raccolti lato client come, ad esempio, il response time e che non forniscono una chiara indicazione su cosa sta accadendo lato server. I parametri di basso livello, invece, sono tutti quei parametri che sono stati recuperati direttamente dal SUT e consistono in informazioni riguardanti lo stato del server come, ad esempio, l'utilizzo dei processori e della memoria. Dopo aver raccolto questi parametri, essi vengono caratterizzati applicando tecniche di PCA e clustering. Come risultato, si ottengono i parametri di basso livello caratterizzati (LL_c) ed i parametri di alto livello caratterizzati (HL_c);
- **Fase 2** → a partire dai parametri di alto livello caratterizzati, viene generato un workload sintetico che, come nella fase 1, viene inviato al medesimo SUT al fine di ottenere nuovi parametri di basso livello che verranno caratterizzati con le medesime operazioni di PCA e Clustering eseguite precedentemente in modo da ottenere una nuova caratterizzazione dei parametri di basso livello (LL'_c);
- **Fase 3** → in questa fase bisogna verificare se non ci sono differenze statistiche tra i dati caratterizzati dal workload reale ed i dati caratterizzati dal workload sintetico. Per fare ciò andremo ad utilizzare i parametri di basso livello (LL_c e LL'_c). Vengono utilizzati tali parametri, e non quelli di alto livello, in quanto se vogliamo confrontare due o più workload dobbiamo fare riferimento al sistema che esegue il workload (il server) e non a quello che riceve le risposte (il client).

Le operazioni illustrate in precedenza sono state rappresentate graficamente nella figura sottostante:

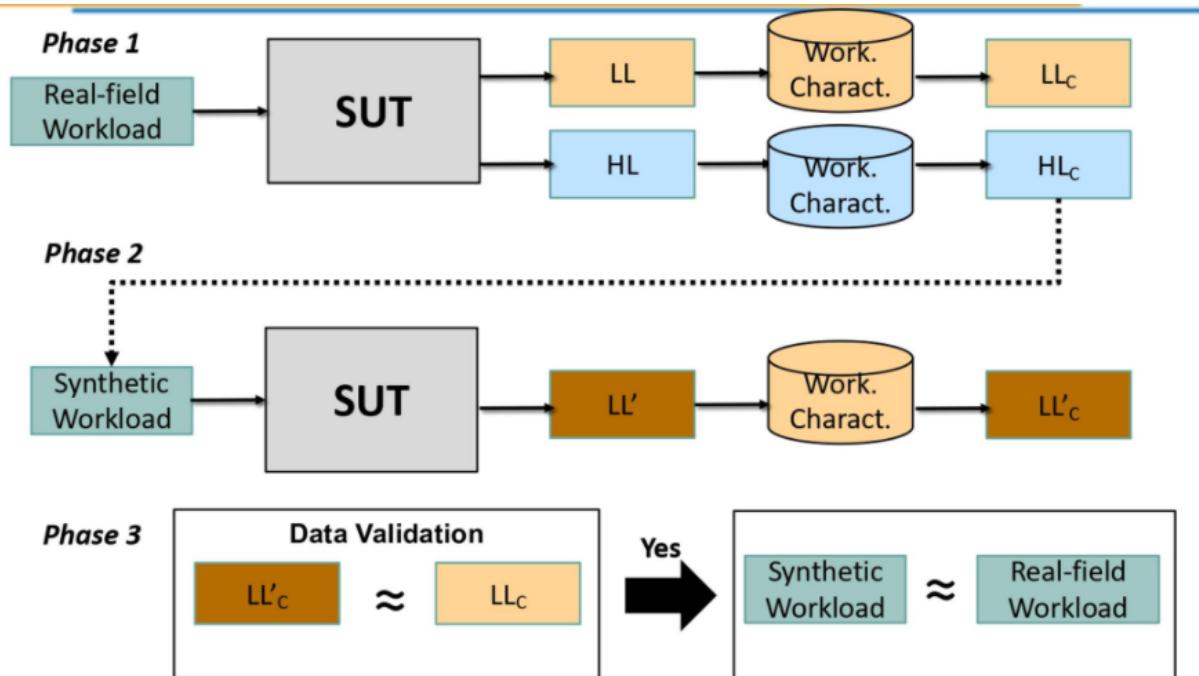


Figura 2.7 - Fasi della workload characterization

2.3.1 Raccolta dei dati per la generazione del workload

Il System Under Test (SUT) che consideriamo in questo caso è lo stesso Web Server di cui abbiamo realizzato il test di capacità spiegato dettagliatamente nell'apposito paragrafo di questa trattazione. Dunque, stiamo considerando nuovamente un Web Server Apache2 eseguito su una macchina virtuale Oracle VM VirtualBox V 6.1 con sistema operativo Linux (in particolare, la distribuzione Linux installata è Ubuntu 20.04). A differenza del precedente test di capacità, però, abbiamo modificato il numero di risorse memorizzate all'interno del server: in questo caso, infatti, abbiamo posizionato nella cartella di default *var/www/html* 50 elementi di dimensioni differenti che variano da 1KB fino ad arrivare a 20MB. In particolare, abbiamo realizzato la seguente distribuzione delle risorse:

- 5 risorse della dimensione di 1KB;
- 5 risorse della dimensione di 2KB;
- 5 risorse della dimensione di 3KB;
- 5 risorse della dimensione di 1MB;
- 5 risorse della dimensione di 2MB;
- 5 risorse della dimensione di 3MB;

- 5 risorse della dimensione di 10MB;
- 5 risorse della dimensione di 13MB;
- 5 risorse della dimensione di 16MB;
- 5 risorse della dimensione di 20MB.

Ora, al fine di poter realizzare correttamente la caratterizzazione del workload che vogliamo poi confrontare con una sua versione “sintetica” (come anticipato nel precedente paragrafo) abbiamo raccolto due diverse tipologie di parametri, quelli di alto livello (lato client) e quelli di basso livello (lato server). Spieghiamo, adesso, nel dettaglio come sono stati raccolti entrambi.

Parametri di alto livello

I parametri di alto livello sono stati raccolti nuovamente tramite l'utilizzo del tool Apache JMeter. In questo caso sono stati creati 5 Thread Group differenti ed il Test Plan è stato opportunamente configurato per fare in modo che tali Thread Group vengano eseguiti in maniera sequenziale in modo da rendere le loro esecuzioni indipendenti le une dalle altre. Ognuno di questi Thread Group è costituito da 10 Sampler HTTP Request differenti. Ora, siccome l'osservazione ha avuto una durata complessiva di 5 minuti, questo significa che ogni Thread Group è stato impostato per avere una durata di 60 secondi ciascuno. Inoltre, ogni Thread Group è costituito da 20 thread ed è caratterizzato da ramp-up unitario. Infine, sono stati aggiunti al Test Plan anche 5 Constant Throughput Timer (uno per ogni Thread Group) impostati al valore 1500 ed un listener di tipo Simple Data Writer al fine di raccogliere i seguenti parametri di alto livello:

- *timestamp* → istante temporale UNIX a cui la richiesta viene effettuata;
- *elapsed* → tempo trascorso tra l'invio della richiesta del client e la ricezione dell'ultimo pacchetto della risposta del server;
- *label* → etichetta testuale della richiesta effettuata;
- *responseCode* → codice di risposta del server;
- *responseMessage* → messaggio di risposta del server;
- *threadName* → nome del Thread Group a cui appartiene il thread che ha effettuato la richiesta;
- *dataType* → tipo di dato richiesto;

- *success* → esito della richiesta;
- *failureMessage* → messaggio di fallimento;
- *bytes* → bytes ricevuti dal client;
- *sentBytes* → bytes emessi dal client;
- *grpThread* → numero di threads attivi all'interno del gruppo;
- *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;
- *idleTime* → tempo nel quale il server è attivo e disponibile ma non sta effettuando nessuna operazione produttiva;
- *connect* → tempo necessario per instaurare la connessione con il server.

Dunque, al termine dell'osservazione è stato ottenuto un dataset composto da 6158 istanze (righe) e 17 componenti (colonne).

Parametri di basso livello

Per quanto concerne, invece, i parametri di basso livello, ci si è avvalsi del comando “*vmstat*” eseguito tramite terminale direttamente sulla macchina server. In particolare, tramite l'esecuzione del seguente comando

```
vmstat -n 1 320 | tr -s ' ' | tr '\n' ',' >> low.csv
```

è stato possibile raccogliere i parametri del sistema per un tempo di 320 secondi (leggermente maggiore dei 300 secondi impostati lato client in modo da gestire opportunamente la natura asincrona dell'esecuzione del test) ed un tempo di campionamento di 1 secondo.

I parametri di basso livello raccolti, dunque, sono stati i seguenti:

- *processi* → “*r*” indica i processi in esecuzione, “*b*” il numero di processi bloccati;
- *memoria* → “*swpd*” indica la quantità di memoria virtuale utilizzata, “*free*” la quantità di memoria libera, “*buff*” la quantità di memoria utilizzata come buffer, “*cache*” la quantità di memoria utilizzata come cache;

- *swap* → "si" indica la quantità di memoria su cui il disco ha effettuato degli swap, "so" indica la quantità di memoria su cui è stato effettuato uno swap nel disco;
- *I-O* → "bi" indica la quantità di blocchi inviati ad un dispositivo, "bo" indica la quantità di blocchi ricevuti da un dispositivo;
- *sistema* → "in" indica il numero di interrupt al secondo, "cs" il numero di context switch effettuati al secondo;
- *CPU* → "us" è la percentuale di utilizzo CPU da parte di processi in user space, "sy" la percentuale di utilizzo CPU da parte del kernel, "id" la percentuale di idle della CPU, "wa" è il tempo trascorso in attesa di input o output, "st" indica il tempo di CPU occupato da un'altra macchina virtuale.

Dunque, al termine dell'osservazione è stato ottenuto un dataset composto da 320 istanze (righe) e 17 componenti (colonne).

2.3.2 Workload Characterization

La caratterizzazione dei parametri sia di alto che di basso livello è costituita dalle seguenti fasi:

- pre-processing → eliminazione di eventuali attributi nominali, costanti o altamente correlate;
- PCA e Clustering → individuazione del miglior numero di componenti principali e del numero di cluster creati al fine di conservare quanta più varianza possibile riducendo, però, la dimensionalità del dataset;
- creazione del workload sintetico → selezione dell'istanza più rappresentativa da ognuno dei cluster.

Workload Characterization HL

Per quanto riguarda i parametri di alto livello, abbiamo innanzitutto condotto una fase di filtraggio dei dati. In particolare, abbiamo eseguito le seguenti operazioni sul dataset iniziale:

- sono state rimosse le colonne *timestamp*, *responseCode* e *idleTime* in quanto costituite da tutti valori costanti;
- sono state rimosse le colonne *responseMessage*, *success* e *failureMessage* in quanto costituite sia da valori costanti che categorici;

- sono state rimosse le colonne *dataType* e *URL* in quanto costituite da valori categorici;
- è stata rimossa la colonna *allThreads* in quanto risulta essere identica alla colonna *grpThreads* e, dunque, ovviamente altamente correlata.

Successivamente, abbiamo applicato la PCA dalla quale sono stati considerate 3 componenti principali, mantenendo una varianza spiegata del 97%.

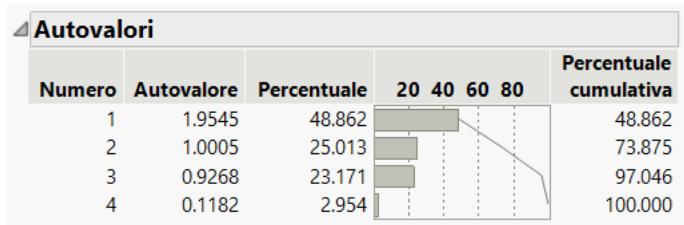


Figura 2.8 - PCA Workload HL

Al termine della PCA, abbiamo effettuato la clusterizzazione scegliendo un numero di cluster pari a 6. Dopo queste operazioni si è mantenuto il 90,70% della varianza originale.

Cronologia di clusterizzazione			
Numero di cluster	Distanza	Leader	Subordinato
1	75.85466342	1	14
2	71.08228101	1	8
3	69.87197415	8	13
4	63.88214916	13	16
5	56.37214499	8	28
6	33.45169177	13	4985
7	27.40297699	8	12
8	26.31087369	16	37
9	25.70320131	28	32
10	22.84418534	16	25
11	22.43095019	12	17
12	18.09610184	12	20
13	14.73148874	1	6
14	14.22945815	28	67
15	14.20253184	8	15
16	13.48430845	13	1213
17	12.97360307	37	59
18	12.25492231	1	2
19	12.24136639	32	1351
20	12.09837049	67	98
21	10.19450848	20	34
22	10.15736966	25	33
23	9.94035919	17	2501
24	9.57513877	1	1192
25	8.78854951	8	1205
26	8.56390136	16	22
27	8.51616695	59	81
28	8.47366863	14	24
29	7.92381884	6	1181
30	7.90506000	13	29
31	7.88012821	28	96

Figura 2.9 - Clustering Workload HL

Considerando, dunque, la caratterizzazione dei parametri di alto livello ottenuta a valle delle operazioni di PCA e Clustering, è stato sviluppato un workload sintetico che comprende una singola richiesta per ciascun cluster. Per fare ciò abbiamo realizzato un workflow nel tool Knime al fine di identificare la richiesta più frequente per ogni cluster.

Workload Characterization LL

Come spiegato anche per quanto riguarda i parametri di alto livello, una volta ottenuto il dataset “low-level” abbiamo innanzitutto effettuato una prima fase di filtraggio. In particolare, abbiamo deciso di rimuovere le righe che vanno da 2 a 4 e le righe che vanno da 306 a 321 in quanto corrispondenti proprio ai 20 secondi extra con cui sono stati raccolti i dati lato server necessari - come spiegato in precedenza - per poter gestire la natura asincrona del test. Tali righe, infatti, presentavano valori della colonna “*id*” (percentuale in cui la CPU non è occupata) molto elevati, indicando proprio che le richieste lato client dovevano ancora iniziare oppure erano già terminate. Inoltre, anche le colonne “*wa*” e “*st*” sono state rimosse perché presentavano tutti valori costanti.

Fatto ciò, abbiamo applicato la PCA al dataset così ottenuto. Abbiamo scelto di selezionare 7 componenti principali in modo da conservare quasi il 96% della varianza.

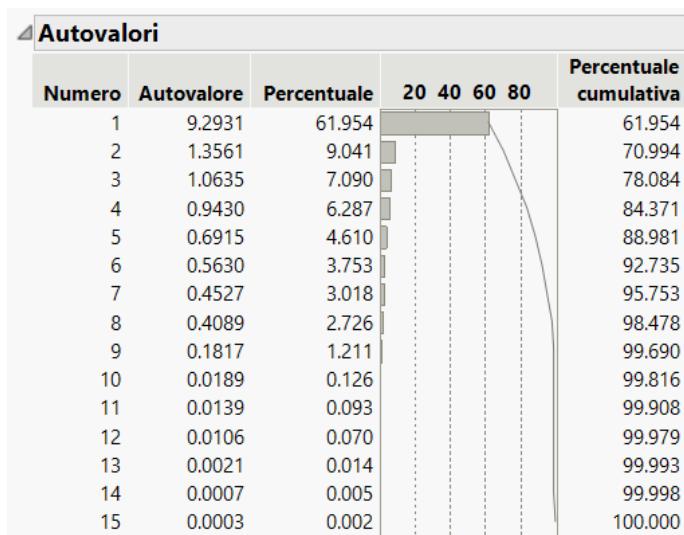


Figura 2.10 - PCA Workload LL

In seguito, abbiamo effettuato la clusterizzazione, scegliendo di raggruppare le istanze del dataset in 8 cluster al fine di conservare, a valle delle operazioni di PCA e clustering, l'81,82% della varianza iniziale.

Cronologia di clusterizzazione			
Numero di cluster	Distanza	Leader	Subordinato
1	17.09889658	1	6
2	14.60397631	1	5
3	14.35572922	6	8
4	14.24962565	6	10
5	13.49445438	1	3
6	13.37360492	6	12
7	10.29297610	12	50
8	8.36891575	8	61
9	7.23294100	12	23
10	6.98483319	3	4
11	6.65827690	6	24
12	6.46601622	10	47
13	5.93831277	4	15
14	5.12647008	24	52
15	4.80717658	23	49
16	4.75545262	52	126
17	4.33759518	10	63
18	4.33005046	5	42
19	3.95223352	50	73
20	3.71398516	6	69
21	3.65220821	47	211
22	3.46598197	24	53
23	3.42653316	8	123
24	3.33434550	1	2
25	3.30951185	12	13
26	3.26570771	73	214
27	3.16406421	47	85
28	2.77451995	50	76
29	2.74893877	23	28
30	2.62268809	6	66
31	2.51534459	126	129
32	2.44146812	63	217
33	2.41136406	61	182
34	2.40551099	4	18
35	2.35813336	47	54

Figura 2.11 - Clustering Workload LL

Workload Characterization LL'

Dopo la creazione del workload sintetico di alto livello, tale carico è stato sottoposto nuovamente al nostro SUT mediante la creazione di un nuovo test plan per mezzo del tool Apache JMeter. In seguito, attraverso l'utilizzo del comando `vmstat`, sono stati raccolti i dati di basso livello *LL'* come effettuato in precedenza. Una volta ottenuti i dati, è stata eseguita una fase di filtraggio. In particolare sono state rimosse le righe da 2 a 13 e da 315 a 321 poiché corrispondenti ai secondi extra di raccolta dei dati lato server, come già spiegato in precedenza. Inoltre, come già effettuato nel workload *LL*, sono state eliminate le colonne *wa*

e st. Dopo la fase di filtraggio è stata applicata la stessa workload characterization applicata ai parametri LL , ovvero sono state scelte 7 componenti principali e 8 cluster.

2.3.3 Confronto LL e LL'

Definiti i workload LL ed LL' , dobbiamo adesso andare a confrontarli per verificare se essi appartengono alla stessa popolazione. Infatti, nel caso in cui i test che mostreremo di seguito avranno come risultato il fatto che i due workload non sono statisticamente differenti, allora potremo affermare che le richieste del workload sintetico approssimano bene le richieste del workload reale di partenza.

A questo punto, dunque, la data validation dei due workload è caratterizzata innanzitutto da un test sulla normalità delle 7 componenti principali. Successivamente - in base al risultato del suddetto test di normalità - sarà necessario applicare un test parametrico o non parametrico al fine di verificare che le popolazioni non siano statisticamente differenti.

Riportiamo, dunque, di seguito i grafici dei QQ-Plot ottenuti dall'analisi delle componenti principali del workload LL :

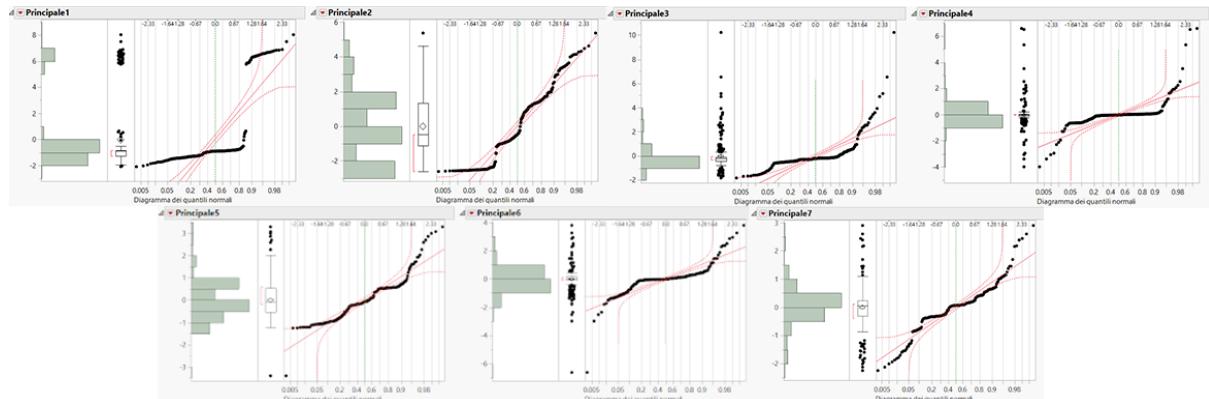


Figura 2.12 - Diagramma dei quantili normali delle componenti principali di LL

In questo caso, dunque, il test visivo è stato sufficiente per stabilire la non normalità di tutte le 7 componenti principali del workload LL .

Di seguito, invece, riportiamo i QQ-Plot ottenuti dall'analisi delle componenti principali del workload LL' :

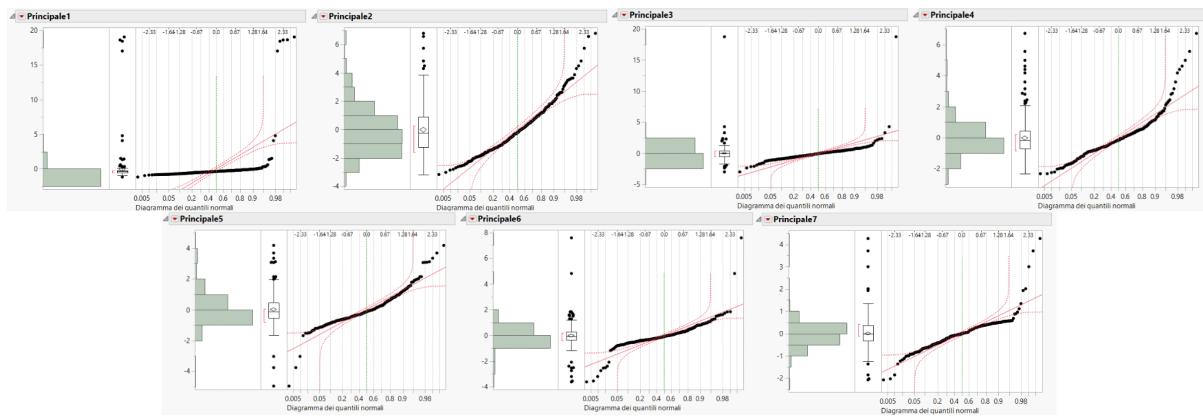


Figura 2.13 - Diagramma dei quantili normali delle componenti principali di LL'

In questo caso, l'unica componente per cui il test visivo ci lascia qualche perplessità rispetto alla sua normalità è la seconda. Dunque, anche se la non normalità di tutte le altre - sia per il workload LL che per il workload LL' - imporrebbe comunque l'utilizzo di un test non parametrico, per completezza abbiamo voluto realizzare un'ulteriore verifica della non normalità della seconda componente principale del workload LL' attraverso il cosiddetto test di Kolmogorov-Smirnov. Tale test, effettuato tramite l'apposita funzione Matlab, ha dato come risultato il valore 1, il quale implica il rigetto dell'ipotesi nulla secondo la quale la distribuzione è normale (rigettando tale ipotesi nulla, dunque, il test ci assicura che anche in questo caso la distribuzione è non normale).

Considerando, dunque, la non normalità di tutte le componenti principali, abbiamo applicato un test non parametrico ed, in particolare, il cosiddetto test di Wilcoxon. L'ipotesi nulla di questo test consiste nell'affermare che i campioni passati all'interno di due vettori provengono da distribuzioni continue con mediane uguali. Abbiamo effettuato tale test nuovamente tramite l'apposita funzione Matlab e, per ogni campione che è stato confrontato, riporta in output un p-value maggiore del livello di significatività fissato ($\alpha = 0.05$) ed un valore di uscita pari a 0, indicando dunque che **l'ipotesi nulla non può essere rigettata**. Questo significa che **i campioni non sono statisticamente differenti**.

The screenshot shows the MATLAB interface with two tabs open: 'deviance.m' and 'Wilcoxon.m'. The 'Wilcoxon.m' tab is active, displaying the following code and output:

```

12
13 x4 = [8,0,416344,61756,2000,141416,200,404,5904,404,9966,1272,2,96,1,0.911856433,2.015776523,-1.03490
14 y4 = [1,0,536044,38904,3416,155072,0,0,0,0,11865,404,1,41,58,-0.162036393,-1.770580563,0.555309013,0.
15
16 x5 = [4,0,416856,75496,1740,135204,0,0,0,20,13624,853,0,95,5,1.309444859,-0.276718321,-0.346565706,0.
17 y5 = [5,0,535532,34472,2708,142788,0,0,2052,0,16418,823,4,74,19,-0.168858979,1.192450361,-0.298177979
18

```

Command Window output:

```

p =
0.7071

h =
logical
0

>> [p, h] = ranksum(x2, y2)

p =
0.1656

h =
logical
0

```

Figura 2.14 - Test di Wilcoxon

Per completezza, riportiamo di seguito i valori dei p-value ottenuti:

p-value
0,7071
0,1656
0,3910
0,1727
0,8228
0,1451
0,8780
0,2628

In conclusione, tramite questo test possiamo affermare che il workload sintetico generato a partire dalla caratterizzazione statistica del workload di alto livello produce effetti statisticamente equivalenti a quello reale sul sistema in analisi. Questo significa che il workload sintetico creato a partire da quello di alto livello risulta essere ben rappresentativo del carico imposto al sistema.

2.4 Design of Experiment

Il design of experiment (*DoE*) permette di identificare i fattori che maggiormente vanno ad influire sulla variabile d'uscita del sistema. In generale i fattori si dividono in due categorie, controllabili e non controllabili. Nel nostro caso andremo a considerare soltanto i fattori controllabili, in quanto cercheremo di capire come - al loro variare - cambierà la variabile d'uscita. Dunque, bisogna considerare i contributi che ogni singolo fattore dà sulla variabile d'uscita, i cosiddetti effetti, tramite i quali possiamo determinare i "fattori importanti", che contribuiscono maggiormente. Per poter determinare correttamente l'effetto di ogni fattore è necessario considerare un modello, prestando però attenzione all'errore che necessariamente verrà prodotto. A questo punto l'importanza di ogni fattore potrà essere calcolata tramite il concetto di varianza spiegata, al quale però bisogna affiancare la significatività statistica del fattore.

2.4.1 Configurazione

La configurazione del DoE è stata effettuata sul tool JMP attraverso la creazione di un piano personalizzato che presenta un design con due fattori da tre livelli differenti ognuno:

- **Dimensione Pagina:** le dimensione delle pagine delle richieste HTTP, possono essere Small (1KB-3KB), Medium (1MB-3MB) o Large (13MB-20MB);
- **Intensità:** il carico imposto al sistema in termini di CTT su JMeter, può essere di 750 (25% di usable capacity), 1500 (50% di usable capacity) o 2250 (75% di usable capacity) richieste per minuto.

La variabile di risposta è il **Response Time**, ovvero il tempo impiegato dal server per servire una richiesta. La configurazione prevede anche un numero di ripetizioni pari a 5, dunque in totale per coprire tutte le possibili combinazioni per un **full factorial design** sarà necessario effettuare 45 esperimenti.

Di seguito si riporta la tabella degli esperimenti con i relativi Response Time:

	Intensità	Dimensione Pagina	Response Time
1	750	Small	2
2	750	Small	1
3	750	Small	1
4	750	Small	2
5	750	Small	2
6	750	Medium	21.14
7	750	Medium	20
8	750	Medium	22.02
9	750	Medium	35.13
10	750	Medium	37.82
11	750	Large	461
12	750	Large	2166.16
13	750	Large	1774.76
14	750	Large	2496
15	750	Large	2202.3
16	1500	Small	2
17	1500	Small	1
18	1500	Small	2
19	1500	Small	1
20	1500	Small	1
21	1500	Medium	16
22	1500	Medium	15
23	1500	Medium	16
24	1500	Medium	36
25	1500	Medium	35
26	1500	Large	2918.25
27	1500	Large	2300.66
28	1500	Large	1997.31
29	1500	Large	2442.13
30	1500	Large	2172.54
31	2250	Small	1
32	2250	Small	1

Figura 2.15 - Configurazione DoE

2.4.2 Analisi dell'importanza

Dopo aver configurato il DoE e dopo aver eseguito tutti gli esperimenti richiesti, dobbiamo, innanzitutto, andare a verificare l'importanza dei fattori. **Un fattore si definisce importante quando esprime una buona percentuale di variazione rispetto alla variazione totale del sistema.** Per conoscere l'importanza dei diversi fattori tramite JMP si è svolto il test degli effetti e l'analisi della varianza.

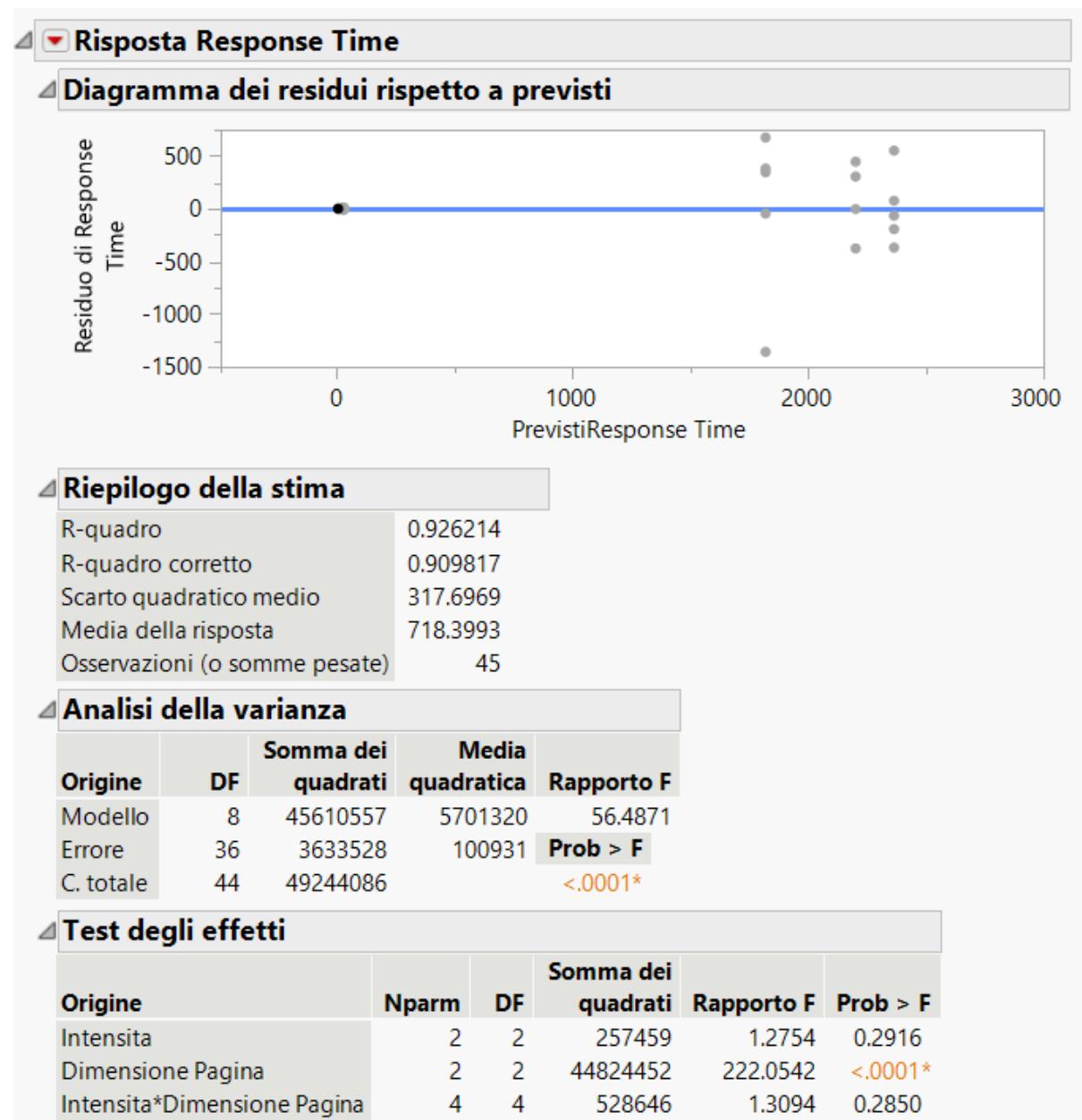


Figura 2.16 - Analisi della varianza e test degli effetti

Dalle tabelle, riportate nella figura 2.16, è stato possibile ottenere i seguenti parametri:

- **SSA (sum of square factor A)**, percentuale di varianza spiegata dalla dimensione della pagina;

- **SSB** (*sum of square factor B*), percentuale di varianza spiegata dall'intensità;
- **SSAB** (*sum of square interaction AB*), percentuale di varianza spiegata dall'interazione tra la dimensione della pagina e l'intensità;
- **SSE** (*sum of square error E*), percentuale di varianza spiegata dall'errore.

Andiamo, ora, a calcolare la percentuale di variazione associata ad ogni singolo fattore (compresi quelli di interazione e dell'errore):

$$\frac{SSA}{SST} = \frac{44824452}{49244086} = 91,02\% \text{ (Dimensione Pagina)}$$

$$\frac{SSB}{SST} = \frac{257459}{49244086} = 0,5\% \text{ (Intensità)}$$

$$\frac{SSAB}{SST} = \frac{528646}{49244086} = 1,1\% \text{ (Dimensione Pagina · Intensità)}$$

$$\frac{SSE}{SST} = \frac{3633528}{49244086} = 7,38\% \text{ (Errore)}$$

Da queste analisi si ottiene che **l'unico fattore importante è “Dimensione Pagina”** in quanto spiega più del 90% della varianza complessiva.

2.4.3 Analisi della significatività

A questo punto, l'ultimo aspetto da considerare è l'analisi della **significatività** dei fattori. La significatività di un fattore è definita come la percentuale di variabilità spiegata da un fattore rispetto a quella spiegata dall'errore e, dunque, a differenza dell'importanza che è un concetto puramente relativo ed a totale giudizio dell'analista, **la significatività è un concetto statistico**. Proprio per questo motivo, la significatività di un fattore non è determinata da un risultato interpretabile bensì - come vedremo - viene stabilita da un vero e proprio test.

Generalmente, dunque, il tipo di analisi che si va a fare per il calcolo della significatività di un fattore rientra nella cosiddetta **ANOVA** (Analysis of Variance). Lo specifico test da applicare dipende da alcune caratteristiche della distribuzione dei residui:

- normalità → la distribuzione dei residui è approssimabile ad una distribuzione normale;
- omoschedasticità → la deviazione standard dei residui è costante.

Innanzitutto, per poter verificare tali proprietà è stato necessario calcolare proprio i residui dei valori precedentemente raccolti tramite l'apposito strumento di JMP.

Fatto ciò, abbiamo valutato la normalità della distribuzione tramite un test visivo effettuato sul seguente QQ-plot:

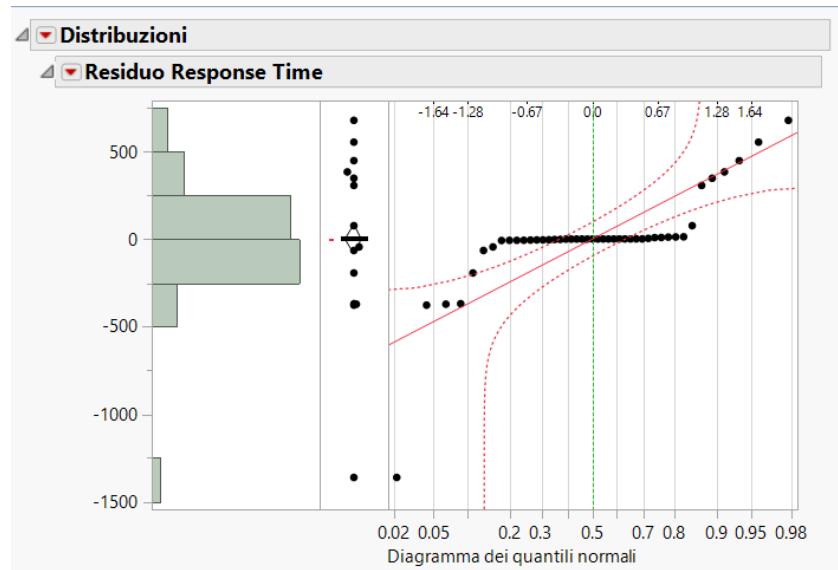


Figura 2.17 - Diagramma dei quantili normali dei residui di Response Time

Come è possibile notare, dunque, la distribuzione dei residui risulta essere **non normale**.

Ora, siccome la distribuzione dei residui risulta essere non normale, non sarà necessario effettuare la verifica dell'omoschedasticità in quanto in ogni caso - sia se i residui dovessero risultare omoschedastici che eteroschedastici - dovremo effettuare il cosiddetto **test non parametrico di Kruskal-Wallis**.

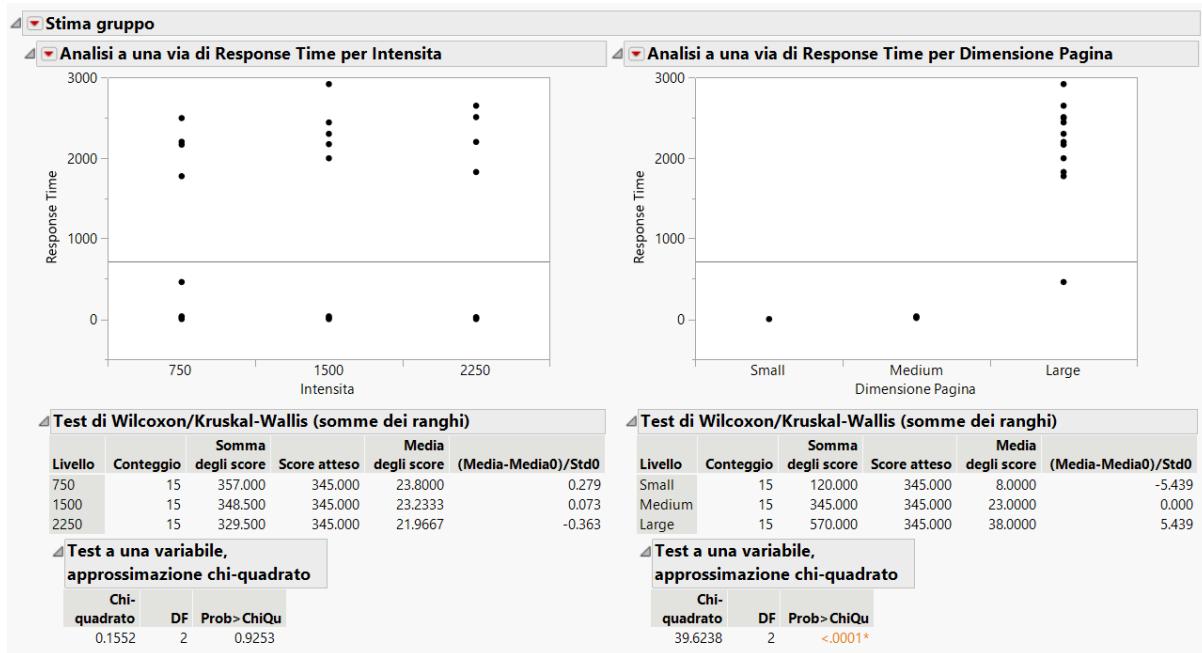


Figura 2.18 - Test non parametrico di Wilcoxon/Kruskal-Wallis

Come è possibile notare, il test di Kruskal-Wallis evidenzia la non significatività del fattore “*Intensità*” mentre, al contrario, il fattore “*Dimensione Pagina*” risulta essere significativo.

In conclusione, dunque, a valle delle analisi fatte possiamo affermare che **il fattore “Dimensione Pagina” risulta essere sia importante che significativo**. Al contrario, invece, **il fattore “Intensità” risulta essere non importante e non significativo**.

Capitolo 3

PCA e Clustering

3.1 Traccia

A partire da un workload reale, si effettui la caratterizzazione di tale workload utilizzando le tecniche di PCA (Principal Component Analysis) e Clustering. Successivamente si vada ad estrarre un workload sintetico da quello reale.

3.2 Data Understanding e Pre-processing dei dati

Per garantire una corretta analisi del workload è stata condotta una fase di comprensione approfondita dei dati (data understanding). Il workload reale è composto da 4995 campioni (righe) che presentano ognuno 24 parametri (colonne). Tali parametri sono i seguenti:

- **VmPeak**: dimensione di picco della memoria virtuale;
- **VmSize**: dimensione della memoria virtuale;
- **VmHWM**: picco della porzione di memoria virtuale occupata da un processo;
- **VmRSS**: dimensione della porzione di memoria virtuale occupata dal processo;
- **VmPTE**: dimensione della tabella di paginazione della memoria virtuale;
- **Threads**: numero di thread nel processo che contiene questo thread;
- **MemFree**: quantità di RAM fisica inutilizzata dal sistema;
- **Buffers**: quantità di RAM fisica utilizzata per i buffer di file;
- **Cached**: quantità di RAM fisica utilizzata come memoria cache;
- **Active**: memoria che è stata usata più recentemente. Non viene recuperata a meno che non sia assolutamente necessario;
- **Inactive**: memoria che è stata usata meno di recente. È più idonea ad essere recuperata per altri scopi;
- **Dirty**: memoria in attesa di essere riscritta sul disco;
- **Writeback**: dati attivamente riscritti in memoria;

- **AnonPages**: memoria utilizzata dalle pagine non supportate da file mappati in tabelle di pagina nello spazio utente;
- **Mapped**: memoria utilizzata per mappare dispositivi, file o librerie;
- **Slab**: memoria usata dal kernel sotto forma di strutture dati cache per uso esclusivo;
- **PageTables**: quantità di memoria dedicata al livello più basso di tabelle di pagina.
- **Committed_AS**: quantità di memoria attualmente allocata sul sistema;
- **NumOfAllocFH**;
- **proc-fd**;
- **avgThroughput**: throughput medio applicato sul sistema;
- **avgElapsed**: tempo medio di risposta delle richieste;
- **avgLatency**: latenza media misurata;
- **Errors**: numero di errori incorsi.

Successivamente alla comprensione del dataset, è stata condotta una fase di filtraggio dei dati (pre-processing) in modo da rimuovere le colonne costanti ed identificare eventuali outlier. Sul workload reale sono state, quindi, eseguite le seguenti operazioni:

- sono stati eliminati i parametri *anonPages*, *avgLatency*, *Errors* in quanto presentano valori costanti;
- è stata eliminata la prima riga del dataset in quanto tale riga rappresenta una fase di startup del sistema;
- è stato eliminato il parametro *Writeback* in quanto risultava essere identico al parametro *Memfree*;
- è stato eliminato il parametro *Slab* in quanto la riga che contiene l'outlier non risulta essere significativa per l'analisi. Per questo motivo eliminando tale outlier il parametro presenterebbe solo valori costanti.

Alla fine di queste operazioni otteniamo un dataset filtrato che, però, presenta ancora un numero elevato di dati da considerare. Per questo motivo bisogna ricorrere a tecniche di Data Reduction.

3.3 Principal Component Analysis (PCA)

La Principal Component Analysis è una delle principali tecniche di Data Reduction. L'idea di base è quella di trasformare lo spazio iniziale in un nuovo spazio di dimensioni ridotte

calcolando nuove componenti, dette appunto componenti principali, che ne rappresentano le direzioni ortogonali. Tali componenti sono incorrelate e linearmente indipendenti tra loro e sono ordinate in maniera tale che i primi valori spieghino la maggior parte della variabilità del dataset.

Nello specifico, ogni punto del dataset viene proiettato in questo nuovo spazio dimensionalmente ridotto selezionando solo alcune delle prime componenti principali, così da ottenere un dataset caratterizzato da una minore dimensionalità ma che conserva la maggior quantità di varianza possibile.

La Principal Component Analysis è stata effettuata tramite il tool JMP eseguendo un metodo di analisi multivariata per il calcolo delle componenti principali, ottenendo così i seguenti risultati:

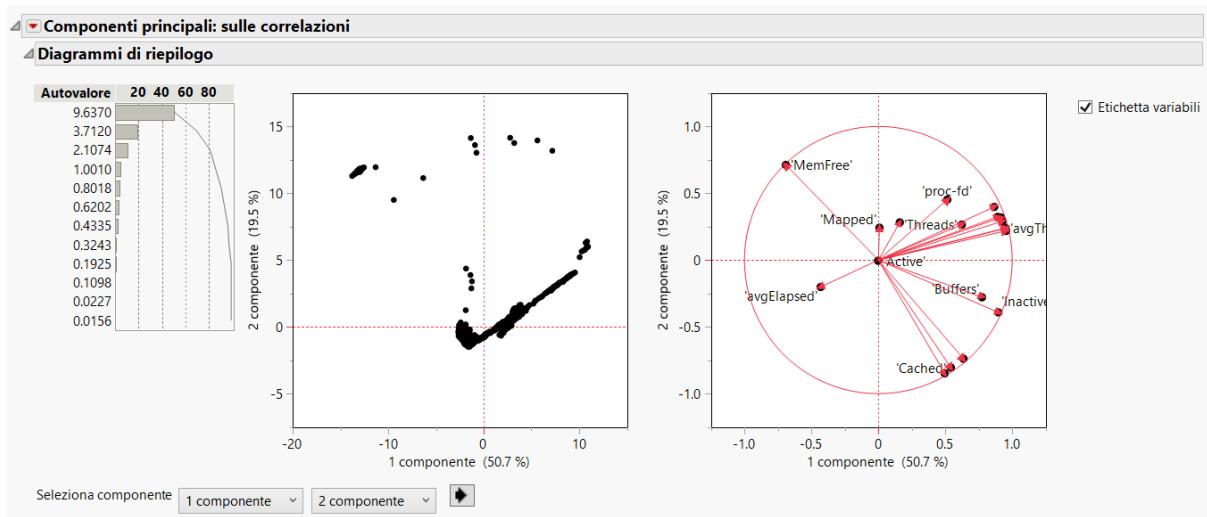


Figura 3.1: Output dell'analisi multivariata

Tra gli output forniti dall'analisi multivariata di JMP troviamo:

- Loading Plot → mostra una matrice di rappresentazione bidimensionale dei fattori di carico. Con tale grafico è possibile osservare che, se due variabili sono simili, sono rappresentate molto vicine tra loro;
- Score Plot → mostra una matrice di dispersione per coppie di componenti principali e spiega, quindi, come si distribuiscono gli elementi in base alle 2 componenti principali selezionate (di default le prime due);
- Diagramma a barre → mostra la varianza conservata da ogni componente principale.

Arrivati a questo punto risulta fondamentale andare a considerare il giusto compromesso tra numero di componenti principali e varianza spiegata (espressa nella colonna “percentuale

cumulativa" di Figura 3.2) sapendo che, anche se l'obiettivo alla base è quello di effettuare Data Reduction, scegliere un numero di componenti principali troppo basso porterebbe sicuramente ad una minore conservazione della varianza dei dati.

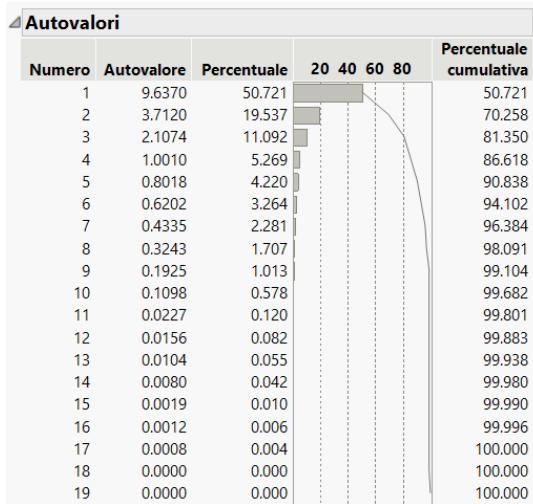


Figura 3.2: Autovalori della PCA

Nel nostro caso abbiamo ritenuto opportuno considerare soltanto le prime **5 componenti principali** in modo tale da conservare circa il **91% della varianza totale** nonostante una drastica riduzione dimensionale.

3.4 Clustering

Estratte le componenti principali dal nostro workload iniziale, il passo successivo che porta alla costruzione di un workload sintetico è quello di **clustering**. Le tecniche di clustering si basano sul mettere a punto un processo in cui organizzare insiemi di oggetti in gruppi in cui i membri di tali gruppi hanno caratteristiche simili tra loro. In particolare, siccome un workload è inizialmente composto da un grande numero di componenti, risulta utile classificare tali componenti in un ridotto numero di cluster. Fatto ciò, potremo scegliere un membro rappresentativo di ognuno di questi cluster al fine di ridurre ulteriormente il dataset, conservando comunque buona parte della varianza iniziale. Mentre con la PCA, dunque, andavamo a diminuire la dimensionalità del dataset riducendo il numero di componenti, tramite il clustering, invece, riduciamo la dimensionalità del dataset andando a ridurre il numero di istanze che compongono il dataset stesso. Per gli scopi della nostra analisi parleremo di **clustering gerarchico** ed, in particolare, abbiamo adottato un approccio

agglomerativo sfruttando il cosiddetto Metodo di Ward. Tramite tale approccio cerchiamo di minimizzare la variabilità interna ad ogni cluster (devianza intra-cluster) e massimizzare quella che intercorre tra cluster differenti (devianza inter-cluster). In questo modo, dunque, otterremo che la similarità degli elementi appartenenti allo stesso cluster sarà elevata mentre la similarità tra elementi appartenenti a cluster diversi sarà bassa.

L'approccio fin qui descritto è facilmente realizzabile in JMP e ci permette di ottenere in output un **dendrogramma** ossia un grafo utilizzato per fornire una rappresentazione grafica del processo di raggruppamento delle istanze. Tenendo conto, dunque, delle 5 componenti principali ottenute dalla PCA, riportiamo di seguito il dendrogramma ottenuto:

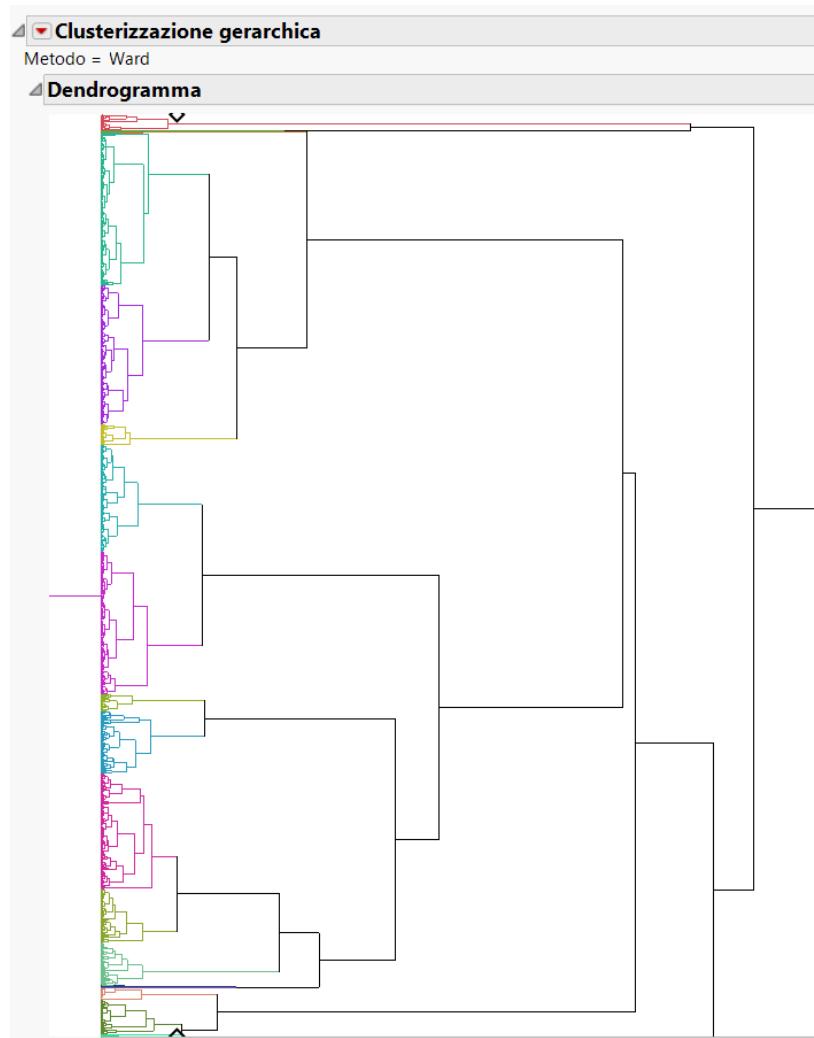


Figura 3.3: Dendrogramma

A questo punto, sfruttiamo la “*Cronologia di Clusterizzazione*” presente in JMP al fine di analizzare le distanze che intercorrono tra i diversi cluster per scegliere il numero ottimale di

cluster da selezionare. In particolare, tale numero ottimale di cluster sarà quello dopo il quale il parametro di distanza non varia in maniera significativa.

Cronologia di clusterizzazione			
Numero di cluster	Distanza	Leader	Subordinato
1	69.11506387	1	91
2	64.80958437	91	4942
3	62.39667961	1	83
4	56.54767382	91	2736
5	55.21593134	91	108
6	35.76350376	108	112
7	31.06047372	112	1930
8	23.04233208	1930	3737
9	21.76961768	91	95
10	19.41036912	83	89
11	18.85036242	1930	1938
12	14.29729308	95	192
13	14.28547899	3737	4139
14	12.21560395	2736	2739
15	11.34954105	95	103
16	10.90269615	112	1925
17	10.64640606	108	1078
18	8.47026309	2739	2843
19	8.05197211	1930	2087
20	7.03793734	1	6
21	5.51821051	2739	2794
22	5.33825567	1930	3645
23	5.22427669	1925	1991
24	4.99869296	95	1081
25	4.96218320	3737	3786
26	4.83949231	1078	1859
27	4.70743664	2739	2801
28	4.45473492	91	92
29	4.45389998	1930	1967
30	4.42987748	1081	1082
31	4.33912399	2087	2266
32	4.32113068	103	120
33	4.29129539	1938	2097

Figura 3.4: Cronologia di Clusterizzazione

Osservando i valori presenti nella finestra “Cronologia di Clusterizzazione”, dunque, la nostra scelta riguardo al **numero ottimale di cluster da selezionare è ricaduta su 8** in quanto - superato tale valore - la distanza inter-cluster resta pressoché invariata nel processo di aggiunta di un ulteriore cluster.

A questo punto, salvando il numero di cluster selezionati, JMP associa automaticamente una nuova colonna alle diverse istanze presenti nel dataset. Il valore presente in tale nuova colonna definisce il cluster di appartenenza di ogni singola istanza. Per completezza, riportiamo di seguito un istogramma che evidenzia bene la distribuzione delle istanze all'interno degli 8 cluster selezionati:

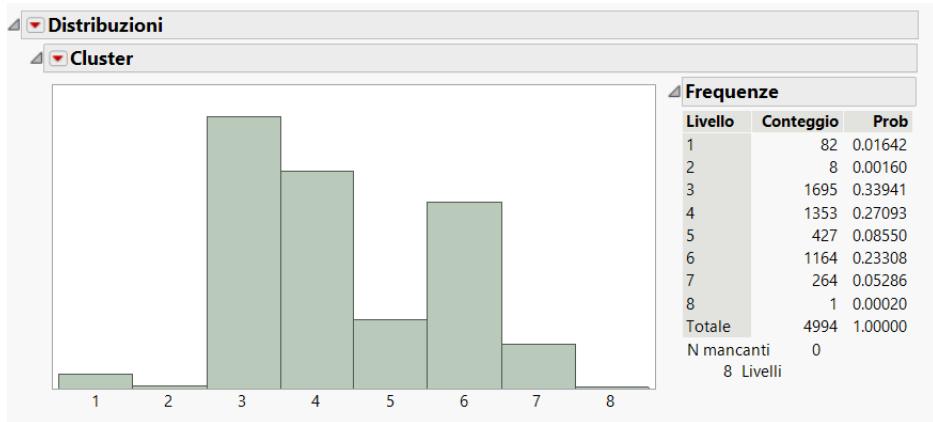


Figura 3.5: Distribuzione dei cluster

Tramite l'utilizzo di un apposito script Matlab, abbiamo ottenuto che la devianza persa effettuando la scelta di 8 cluster risulta essere del 8,458%. Conoscendo tale valore, dunque, è possibile calcolare il valore della varianza complessivamente persa a valle delle operazioni di PCA e Clustering secondo la seguente formula:

$$LostPCA + Clustering = 0,09162 + 0,093111 \cdot 0,90838 = 0,1762 = 17,62\%$$

dove:

- 0,9162 → devianza persa dopo aver effettuato la PCA;
- 0,093111 → devianza intra-cluster;
- 0,90838 → varianza conservata dopo aver effettuato la PCA.

Dunque, possiamo concludere che la varianza spiegata dopo aver effettuato PCA e Clustering risulta essere del **82,38%**.

3.5 Caratterizzazione del workload sintetico

Effettuate le operazioni di PCA e Clustering, a questo punto siamo passati alla caratterizzazione del workload sintetico. Per fare ciò, dunque, abbiamo realizzato un workflow nel tool Knime al fine di individuare il centroide virtuale di ogni cluster. Riportiamo, di seguito, una tabella che mostra i centroidi così individuati:

Cluster	Principale 1	Principale 2	Principale 3	Principale 4	Principale 5
1	-12.98231919	11.6635675	-2.450013	-0.0387205	-0.23229916
2	1.159303004	13.3680358	30.360379	0.03106664	-1.43023309
3	-1.659177482	-0.8788208	0.3200701	-0.0094908	0.746108674
4	-1.331011231	-0.7495949	0.0100924	-0.0684784	-0.59619975
5	2.105829533	0.38011912	-0.454731	-0.0836169	-1.62499555
6	2.345760771	0.36796413	-0.173599	0.07282862	-0.0688425
7	7.730750261	3.22203718	-0.764318	-0.0084079	1.317303984
8	-2.133040009	-0.7924883	-0.135979	64.8163319	-1.26803019

Figura 3.6: Centroidi

Tramite una funzione di distanza euclidea abbiamo, quindi, calcolato il punto più vicino al centroide virtuale individuato per un determinato cluster ed abbiamo considerato tale punto come quello rappresentativo del cluster stesso. Di seguito, dunque, si riporta il workload sintetico così realizzato:

	Cluster 1	Cluster 2	Cluster 3	Cluster 4	Cluster 5	Cluster 6	Cluster 7	Cluster 8
VmPeak'	152272	170344	172392	172392	186864	189652	208184	172388
VmSize'	150216	170340	170340	170340	181984	184820	206904	170340
VmHWM'	25172	31144	32696	32696	41320	41968	55500	32696
VmRSS'	25104	31140	32668	32668	36352	38344	54980	31460
VmPTE'	160	200	200	200	240	248	312	200
Threads'	63	64	22	33	87	71	161	33
MemFree'	5607780	4902916	4599528	4598828	4561712	4558856	4534972	4635532
Buffers'	30796	84164	165664	165924	195576	196464	203772	132072
Cached'	334644	864428	1117452	1117460	1118032	1118048	1118364	1117192
Active'	0	0	0	0	0	0	0	111
Inactive'	141300	454384	435040	435248	480240	483744	520056	412036
Dirty'	301504	624920	933092	933148	922060	921444	909384	921028
Mapped'	20	142552	156	44	176	132	32	164
PageTables'	77356	130788	85008	85012	88920	90652	107320	83800
Committed_AS'	23772	28324	23844	23844	23848	23848	23848	23740
NumOfAllocFH'	27928	95252	110416	110416	113788	114120	113304	109580
proc-fd'	7192	8024	7208	7204	7248	7256	7320	7208
avgThroughput'	294504	366024	314864	316284	330368	329312	347236	316560
avgElapsed'	1530	2040	1530	1020	510	1020	1020	1530

Figura 3.7: Workload sintetico

Capitolo 4

Regressione Lineare

L'obiettivo di questo homework è rilevare e stimare eventuali trend nelle variabili stimate, utilizzando eventualmente modelli regressivi lineari semplici, parametrici e non parametrici.

4.1 Esercizio 1

4.1.1 Traccia

Rilevare e stimare eventuali trend su ognuna delle tre variabili *nmail*, *byte rec* e *byte sent*, utilizzando modelli regressivi lineari semplici, parametrici e/o non parametrici.

4.1.2 Svolgimento

Il primo passo per il corretto svolgimento di tale esercizio è quello di eseguire le stime lineari delle variabili *nmail*, *byte rec* e *byte sent* rispetto alla variabile di predizione *observation* al fine di determinare l' R^2 . Tramite tale coefficiente siamo in grado di misurare il legame tra la variabilità dei dati e la correttezza del modello statistico utilizzato. L' R^2 , infatti, è il rapporto tra devianza totale spiegata dalla regressione (*SSR*) e la devianza totale (*SST*) e, dunque, indica la percentuale della devianza spiegata dal modello regressivo. Se, ad esempio, avessi un R^2 del 99% vuol dire che il modello regressivo spiega il 99% della varianza del campione.

Riportiamo, dunque, di seguito la stima lineare delle 3 variabili sopra descritte:



Figura 4.1: Stime lineari

Dalla Figura 4.1 è possibile notare che - come indicato alla voce “*R-quadro*” all’interno della finestra “*Riepilogo della stima*” - per tutte le variabili il valore del coefficiente R^2 risulta essere molto basso. Questo, dunque, indica che il modello lineare non è sufficiente per poter predire correttamente i dati.

Fatta tale considerazione, abbiamo successivamente calcolato i residui delle variabili *nmail*, *byte rec* e *byte sent* al fine di analizzarne le distribuzioni. Tramite l’apposito strumento del tool JMP, è stato possibile graficare i QQ-plot delle 3 variabili e verificare tramite un semplice test visivo la non normalità di tutte e 3 le distribuzioni.

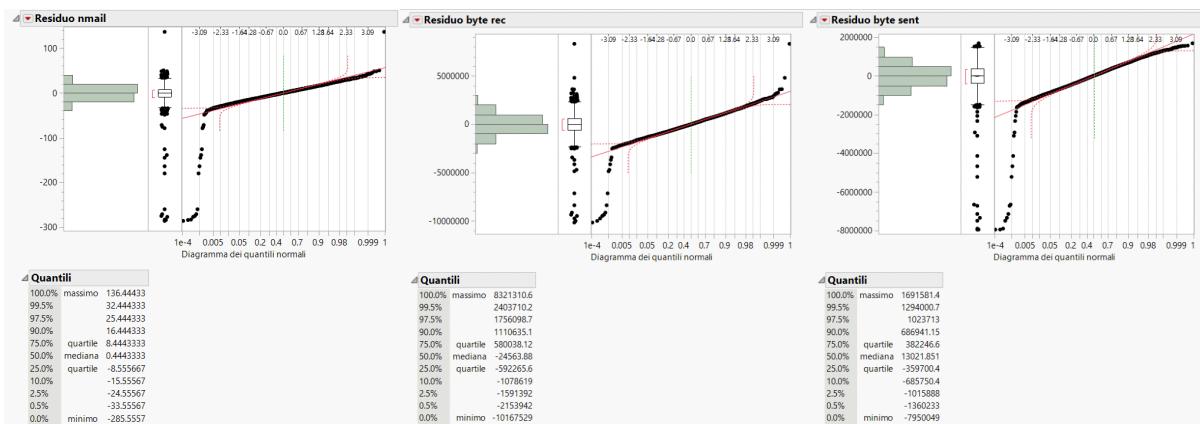


Figura 4.2: Distribuzione dei residui

Tale risultato ci ha condotto alla scelta di utilizzare un test non parametrico al fine di stabilire l’eventuale presenza di un trend nei dati. In particolare, abbiamo scelto il test non

parametrico di Mann-Kendall in quanto tale test non richiede che i dati siano lineari o distribuiti normalmente. L'output del test di Mann-Kendall è costituito da due coefficienti:

- il coefficiente di correlazione $\tau \rightarrow$ assume valori compresi tra -1 ed 1 tali che un valore negativo indica che le variabili sono inversamente correlate mentre un valore positivo indica che quando una variabile aumenta, anche l'altra aumenta;
- il p-value \rightarrow un valore minore o uguale a 0,05 indica che il risultato ottenuto è statisticamente significativo per l'analisi, ossia che l'ipotesi nulla che afferma che non è presente un trend monotono nella distribuzione viene rigettata.

Di seguito, riportiamo i risultati ottenuti a valle dell'esecuzione del test di Mann-Kendall per tutte le variabili considerate:

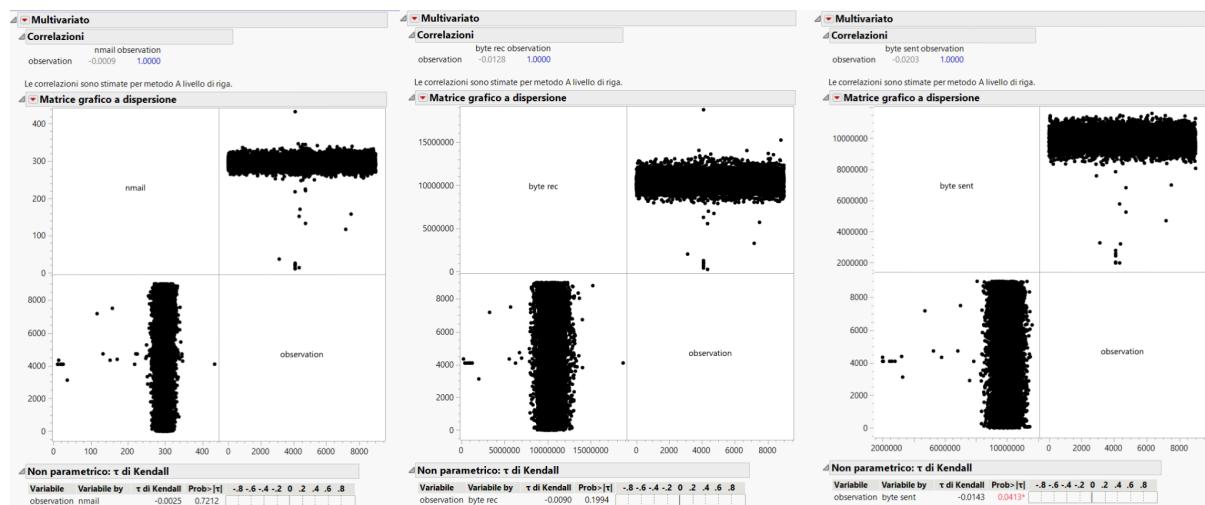


Figura 4.3: Test non parametrico Mann-Kendall

Al fine di favorire la leggibilità, riportiamo di seguito anche una tabella riassuntiva dei risultati ottenuti, evidenziando anche le variabili per cui il test di Mann-Kendall esplicita la presenza di un trend:

Variabili	Tau	P-value	Ipotesi nulla rigettata
nmail	-0.0025	0.7212	0
byte rec	-0.0090	0.1994	0
byte sent	-0.0143	0.0413*	1

Il test di Mann-Kendall (considerando il tasso di significatività standard pari ad $\alpha = 0.05$) riporta che mentre le variabili *nmail* e *byte rec* non presentano trend significativi, per quanto riguarda la variabile *byte sent*, invece, è possibile rilevare un trend.

Infine, dunque, avendo rilevato un trend, abbiamo utilizzato la procedura di Theil-Sen (implementata tramite un apposito script Python) ed abbiamo calcolato pendenza ed intercetta della retta di regressione ed il relativo intervallo di confidenza come mostrato di seguito.

```
x = np.array(x_column)
y = np.array(y_column)
slope, intercept, low, up = theilslopes(y, x, 0.95)
print("Gradiente: {}. Intervallo: [{},{}]. Intercetta: {}".format(slope, low, up, intercept))

Gradiente: -4.514783992985356. Intervallo: [-8.891050583657588, -0.13067767792081614] Intercetta: 9959247.527968435
```

Figura 4.4: Risultati Theil-Sen per la variabile byte sent

4.2 Esercizio 2

4.2.1 Traccia

Rilevare e stimare eventuali trend su ognuna delle tre variabili *nmail*, *byte rec* e *byte sent*, utilizzando modelli regressivi lineari semplici, parametrici e/o non parametrici.

4.2.2 Svolgimento

Il primo passo per il corretto svolgimento di tale esercizio è quello di eseguire le stime lineari delle variabili *nmail*, *byte rec* e *byte sent* rispetto alla variabile di predizione *observation* al fine di determinare l' R^2 .

Riportiamo, dunque, di seguito la stima lineare delle 3 variabili sopra descritte:

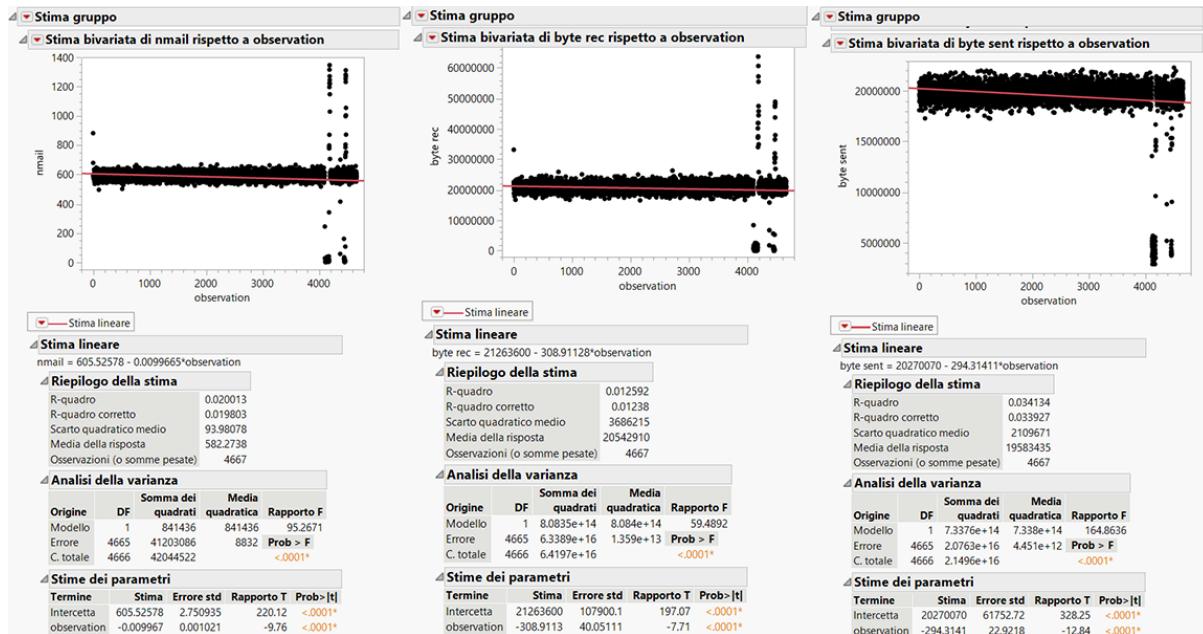
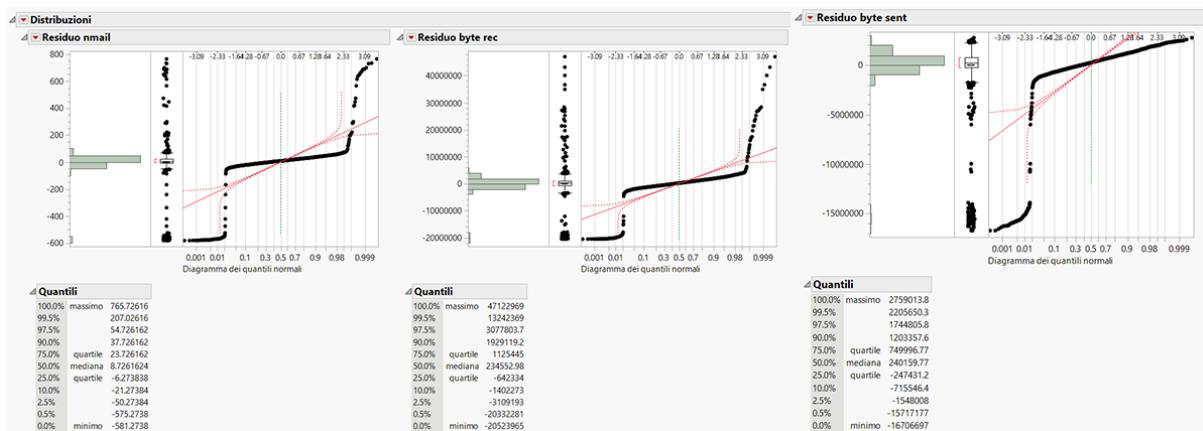


Figura 4.5: Stime lineari

Dalla Figura 4.5 è possibile notare che - come indicato alla voce “R-quadro” all’interno della finestra “Riepilogo della stima” - per tutte le variabili il valore del coefficiente R^2 risulta essere molto basso. Questo, dunque, indica che il modello lineare non è sufficiente per poter predire correttamente i dati.

Fatta tale considerazione, abbiamo successivamente calcolato i residui delle variabili *nmail*, *byte rec* e *byte sent* al fine di analizzarne le distribuzioni. Tramite l’apposito strumento del tool JMP, è stato possibile graficare i QQ-plot delle 3 variabili e verificare tramite un semplice test visivo la non normalità di tutte e 3 le distribuzioni.



Tale risultato ci ha condotto alla scelta di utilizzare un test non parametrico al fine di stabilire l’eventuale presenza di un trend nei dati. In particolare, abbiamo scelto il test non

parametrico di Mann-Kendall in quanto tale test non richiede che i dati siano lineari o distribuiti normalmente.

Di seguito, riportiamo i risultati ottenuti a valle dell'esecuzione del test di Mann-Kendall per tutte le variabili considerate:

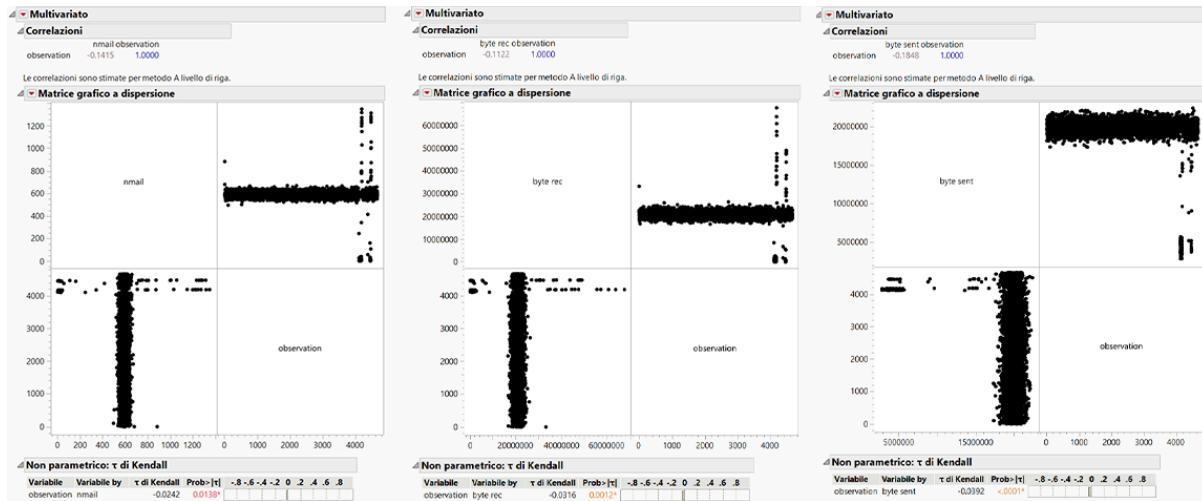


Figura 4.7: Test non parametrico Mann-Kendall

Al fine di favorire la leggibilità, riportiamo di seguito anche una tabella riassuntiva dei risultati ottenuti, evidenziando anche le variabili per cui il test di Mann-Kendall esplicita la presenza di un trend:

Variabili	Tau	P-value	Ipotesi nulla rigettata
nmail	-0.0242	0.0138*	1
byte rec	-0.00316	0.0012*	1
byte sent	-0.0392	0.0001*	1

Il test di Mann-Kendall (considerando il tasso di significatività standard pari ad $\alpha = 0.05$) riporta che tutte e 3 le variabili *nmail*, *byte rec* e *byte sent* presentano trend significativi. Infine, dunque, avendo rilevato un trend, abbiamo utilizzato la procedura di Theil-Sen (implementata tramite un apposito script Python) ed abbiamo calcolato pendenza ed intercetta della retta di regressione ed il relativo intervallo di confidenza come mostrato di seguito.

```

x = np.array(x_column)
y = np.array(y_column)
slope, intercept, low, up = theilslopes(y, x, 0.95)
print("Gradiente: {}. Intervallo: [{},{}]. Intercetta: {}".format(slope, low, up, intercept))

Gradiente: -0.000597315651135006. Intervallo: [-0.0011248593925759281,0.0] Intercetta: 592.3936678614098

```

Figura 4.8: Risultati Theil-Sen per nmail

```

Gradiente: -48.29360967184802. Intervallo: [-77.54773269689737,-18.993146773272414] Intercetta: 20890131.991364423

```

Figura 4.9: Risultati Theil-Sen per byte rec

```

Gradiente: -34.06305208650625. Intervallo: [-50.672672672672675,-17.374087591240876] Intercetta: 19903064.10051782

```

Figura 4.10: Risultati Theil-Sen per byte sent

4.3 Esercizio 3

4.3.1 Traccia

Rilevare e stimare eventuali trend sulle 5 variabili utilizzando modelli regressivi lineari semplici, parametrici e/o non parametrici. Farlo per i tre dataset *os1*, *os2* ed *os3*. Confrontare i trend individuati nei tre dataset.

4.3.2 Svolgimento

Dataset 1

Il primo passo per il corretto svolgimento di tale esercizio è quello di eseguire le stime lineari delle variabili *LIN1_VmSize*, *LIN1_VmData*, *LIN1_RSS*, *LIN1_byte_letti_I_O* e *LIN1_byte_letti_I_O1* rispetto alla variabile di predizione *TIME* al fine di determinare l' R^2 .

Riportiamo, dunque, di seguito la stima lineare delle 5 variabili sopra descritte:

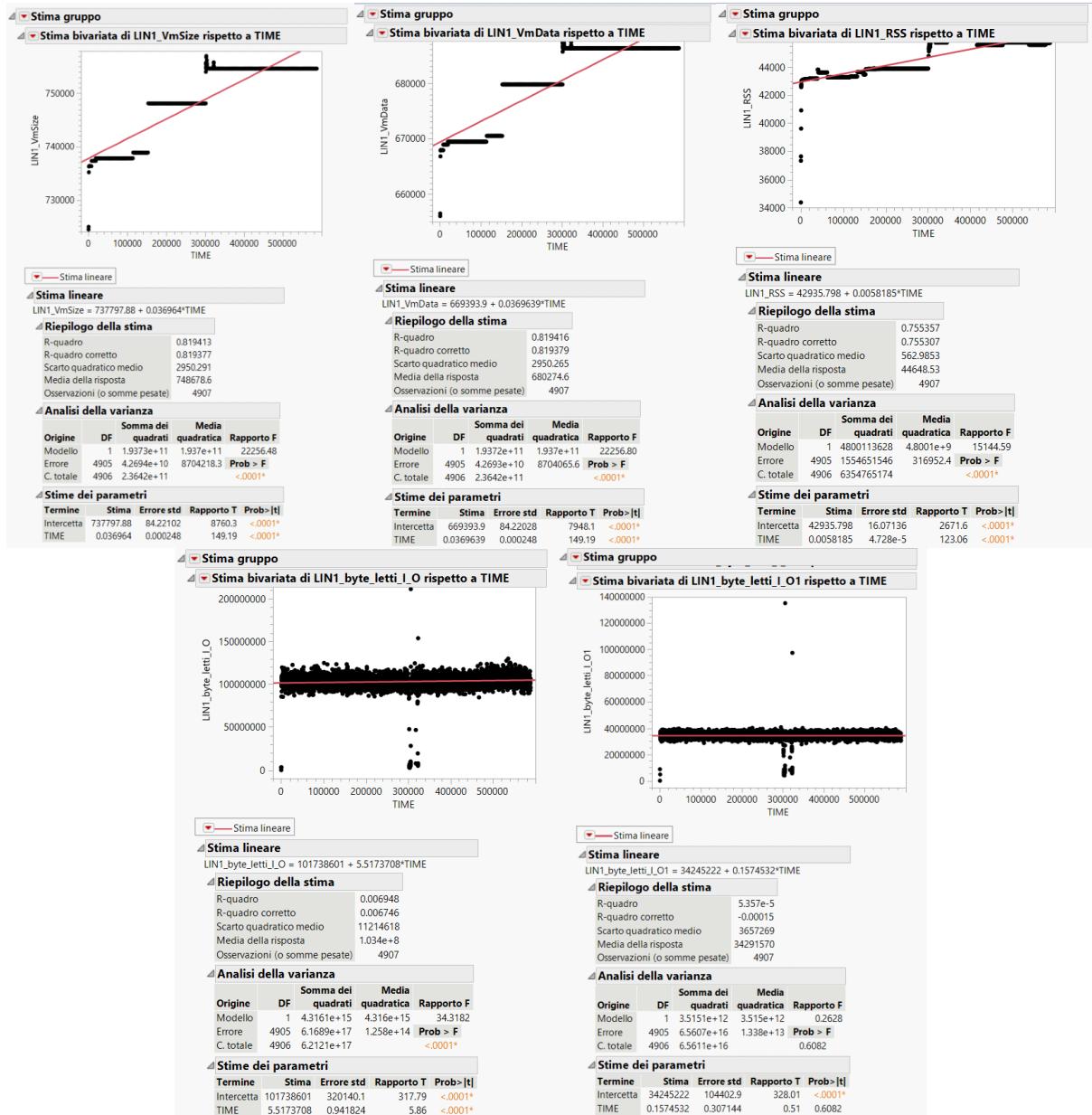


Figura 4.11: Stime lineari

Dalla Figura 4.11 è possibile notare che - come indicato alla voce “R-quadro” all’interno della finestra “Riepilogo della stima” - per le variabili *LIN1_byte_letti_I_O* e *LIN1_byte_letti_I_O1* il valore del coefficiente R^2 risulta essere molto basso mentre per le variabili *LIN1_VmSize*, *LIN1_VmData* e *LIN1_RSS* tale valore risulta essere alto. Questo, dunque, indica che nel primo caso il modello lineare non è sufficiente per poter predire correttamente i dati mentre nel secondo caso andremo a calcolare i residui proprio dal modello di regressione lineare tramite l’apposita funzione di JMP “Analizza → Modellazione Predittiva → Screening del modello”.

Fatta tale considerazione, abbiamo successivamente calcolato i residui delle variabili al fine di analizzarne le distribuzioni. Tramite l'apposito strumento del tool JMP, è stato possibile graficarne i QQ-plot e verificare tramite un semplice test visivo la non normalità di tutte le distribuzioni.

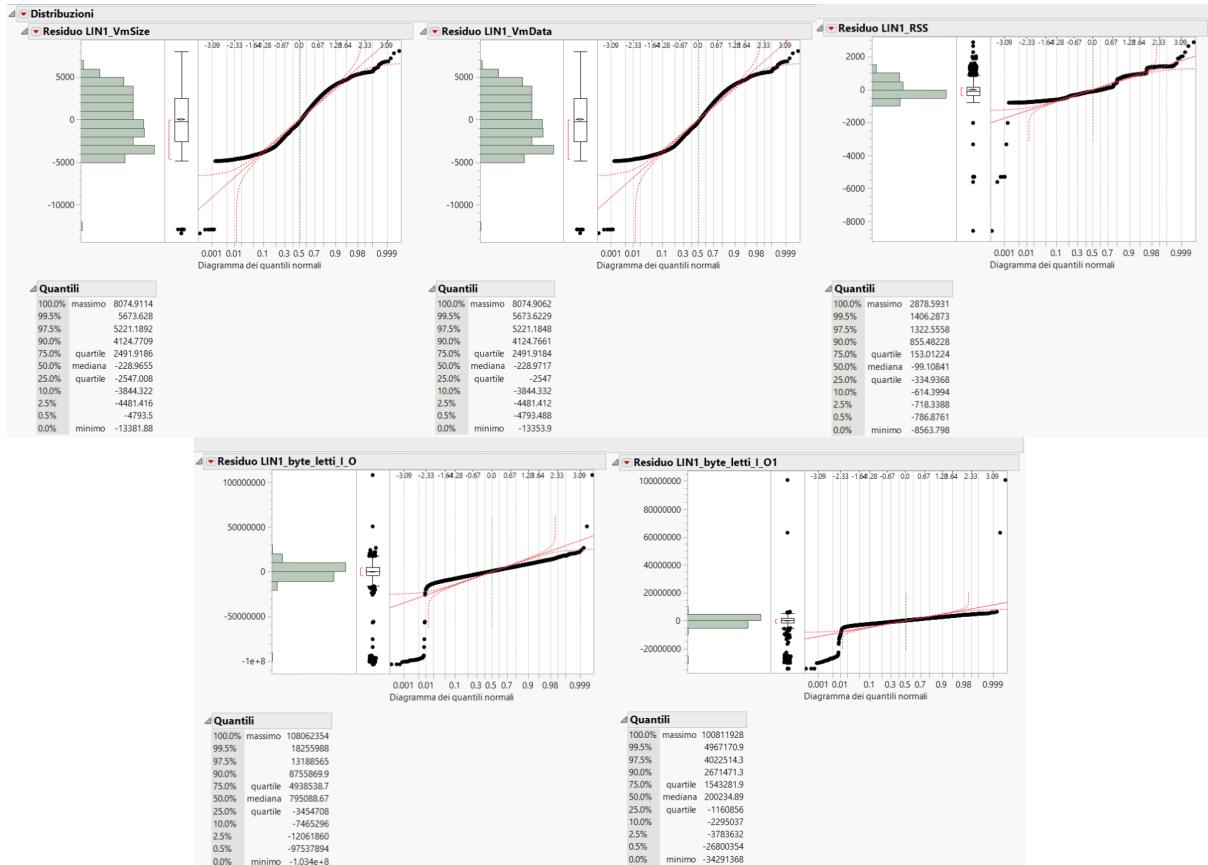


Figura 4.12: Distribuzione dei residui

Tale risultato ci ha condotto alla scelta di utilizzare un test non parametrico al fine di stabilire l'eventuale presenza di un trend nei dati. In particolare, abbiamo scelto il test non parametrico di Mann-Kendall in quanto tale test non richiede che i dati siano lineari o distribuiti normalmente.

Di seguito, riportiamo i risultati ottenuti a valle dell'esecuzione del test di Mann-Kendall per tutte le variabili considerate:

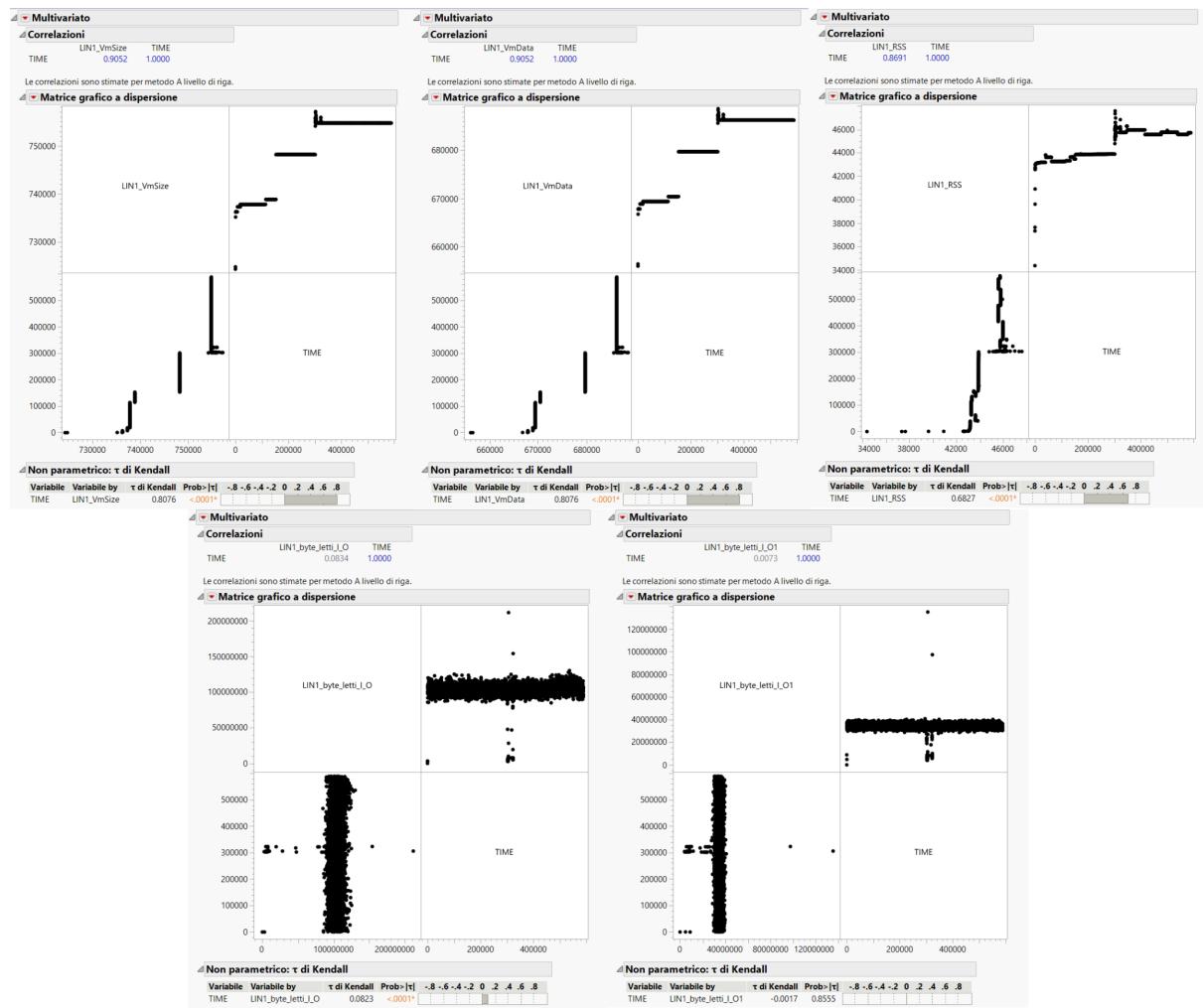


Figura 4.13: Test non parametrico Mann-Kendall

Al fine di favorire la leggibilità, riportiamo di seguito anche una tabella riassuntiva dei risultati ottenuti, evidenziando anche le variabili per cui il test di Mann-Kendall esplicita la presenza di un trend:

Variabili	Tau	P-value	Ipotesi nulla rigettata
LIN1_VmSize	0.8076	0.0001	1
LIN1_VmData	0.8076	0.0001	1
LIN1_RSS	0.6872	0.0001	1
LIN1_byte_letti_I_O	0.0823	0.0001	1
LIN1_byte_letti_I_O1	-0.0017	0.8555	0

Il test di Mann-Kendall (considerando il tasso di significatività standard pari ad $\alpha = 0.05$) riporta che tutte le variabili tranne *LIN1_byte_letti_I_O1* presentano trend significativi. Infine, dunque, avendo rilevato un trend, abbiamo utilizzato la procedura di Theil-Sen (implementata tramite un apposito script Python) ed abbiamo calcolato pendenza ed intercetta della retta di regressione ed il relativo intervallo di confidenza come mostrato di seguito.

```

0 s
x = np.array(x_column)
y = np.array(y_column)
slope, intercept, low, up = theilslopes(y, x, 0.95)
print("Gradiente: {}, Intervallo: [{},{}], Intercetta: {}".format(slope, low, up, intercept))

```

Gradiente: 0.03157224220623501. Intervallo: [0.030715952172645087, 0.03232097662647015] Intercetta: 738854.3947841726

Figura 4.14: Risultati Theil-Sen per la variabile *LIN1_VmSize*

```

Gradiente: 0.03157224220623501. Intervallo: [0.030715952172645087, 0.03232097662647015] Intercetta: 670450.3947841726

```

Figura 4.15: Risultati Theil-Sen per la variabile *LIN1_VmData*

```

Gradiente: 0.004970760233918129. Intervallo: [0.004876649454962708, 0.005055788005578801] Intercetta: 42428.80701754386

```

Figura 4.16: Risultati Theil-Sen per la variabile *LIN1_RSS*

```

Gradiente: 4.737578125. Intervallo: [3.6739155483258776, 5.80459886547812] Intercetta: 102763229.503125

```

Figura 4.17: Risultati Theil-Sen per la variabile *LIN1_byte_letti_I_O*

Dataset 2

Il primo passo per il corretto svolgimento di tale esercizio è quello di eseguire le stime lineari delle variabili *LIN2_VmSize*, *LIN2_VmData*, *LIN2_RSS*, *LIN2_byte_letti_sec* e *LIN2_byte_scritti_sec* rispetto alla variabile di predizione *TIME* al fine di determinare l' R^2 .

Riportiamo, dunque, di seguito la stima lineare delle 5 variabili sopra descritte:

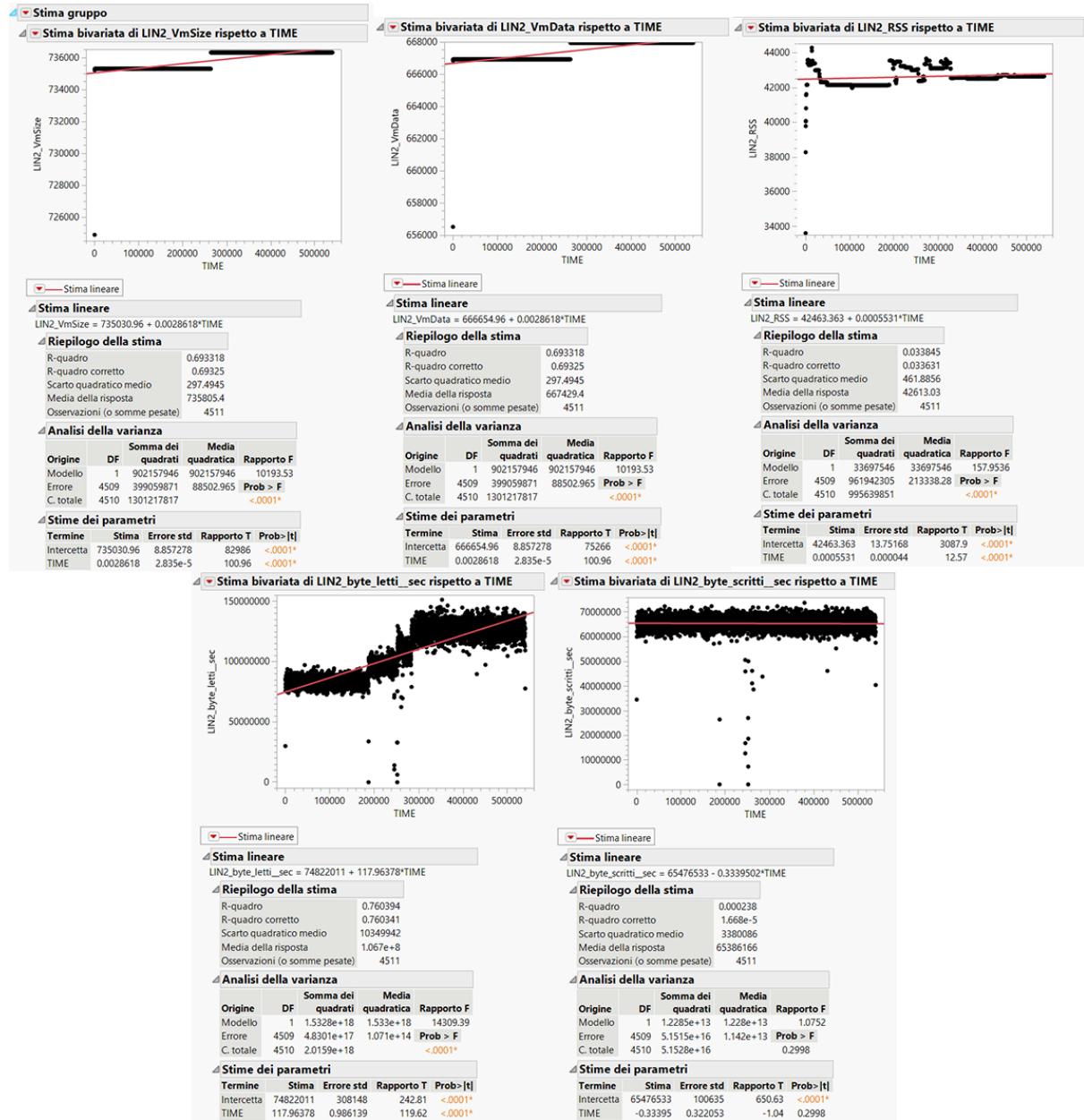


Figura 4.18: Stime lineari

Dalla Figura 4.18 è possibile notare che - come indicato alla voce “*R-quadro*” all’interno della finestra “*Riepilogo della stima*” - per le variabili *LIN2_RSS* e *LIN2_byte_scritti_sec* il valore del coefficiente R² risulta essere molto basso mentre per le variabili *LIN2_VmSize*, *LIN2_VmData* e *LIN2_byte_letti_sec* tale valore risulta essere alto. Questo, dunque, indica che nel primo caso il modello lineare non è sufficiente per poter predire correttamente i dati mentre nel secondo caso andremo a calcolare i residui proprio dal modello di regressione lineare tramite l’apposita funzione di JMP “*Analizza → Modellazione Predittiva → Screening del modello*”.

Fatta tale considerazione, abbiamo successivamente calcolato i residui delle variabili al fine di analizzarne le distribuzioni. Tramite l'apposito strumento del tool JMP, è stato possibile graficarne i QQ-plot e verificare tramite un semplice test visivo la non normalità di tutte le distribuzioni.

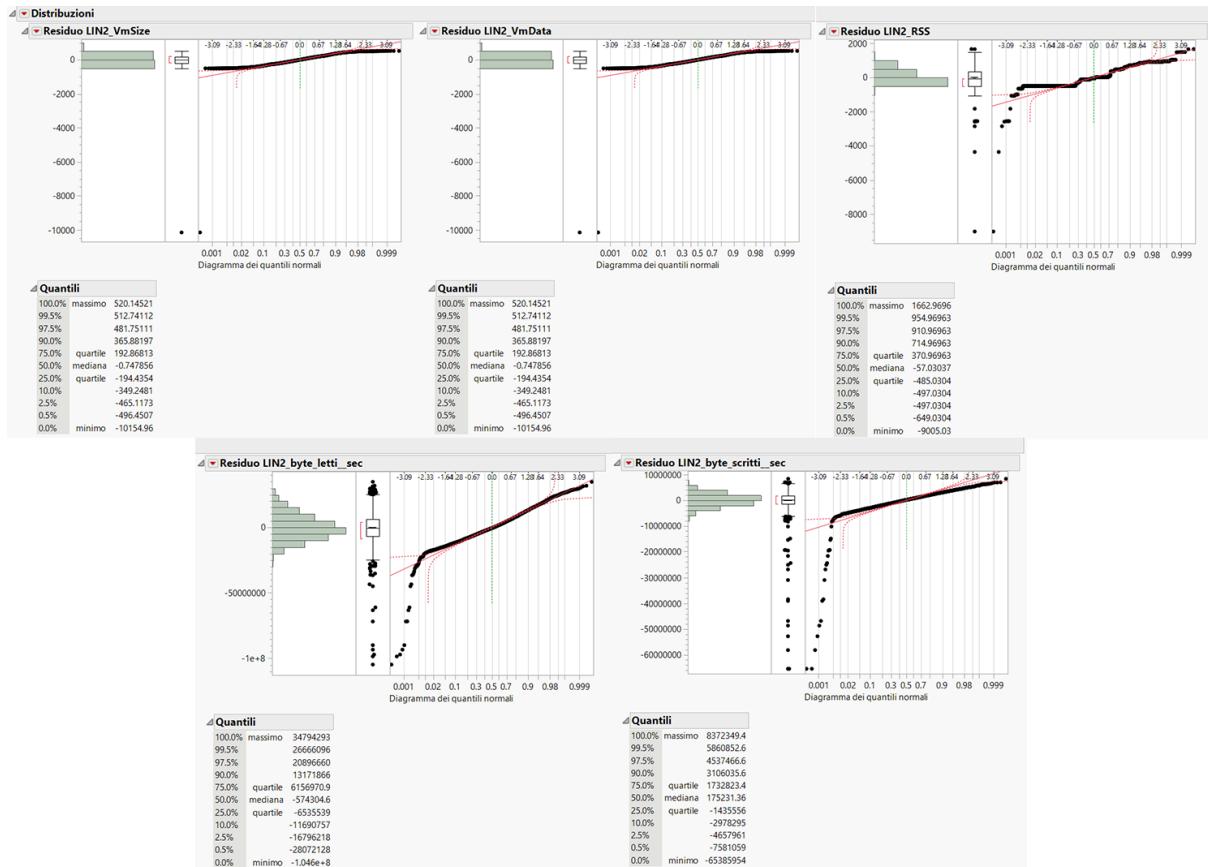


Figura 4.19: Distribuzione dei residui

Tale risultato ci ha condotto alla scelta di utilizzare un test non parametrico al fine di stabilire l'eventuale presenza di un trend nei dati. In particolare, abbiamo scelto il test non parametrico di Mann-Kendall in quanto tale test non richiede che i dati siano lineari o distribuiti normalmente.

Di seguito, riportiamo i risultati ottenuti a valle dell'esecuzione del test di Mann-Kendall per tutte le variabili considerate:

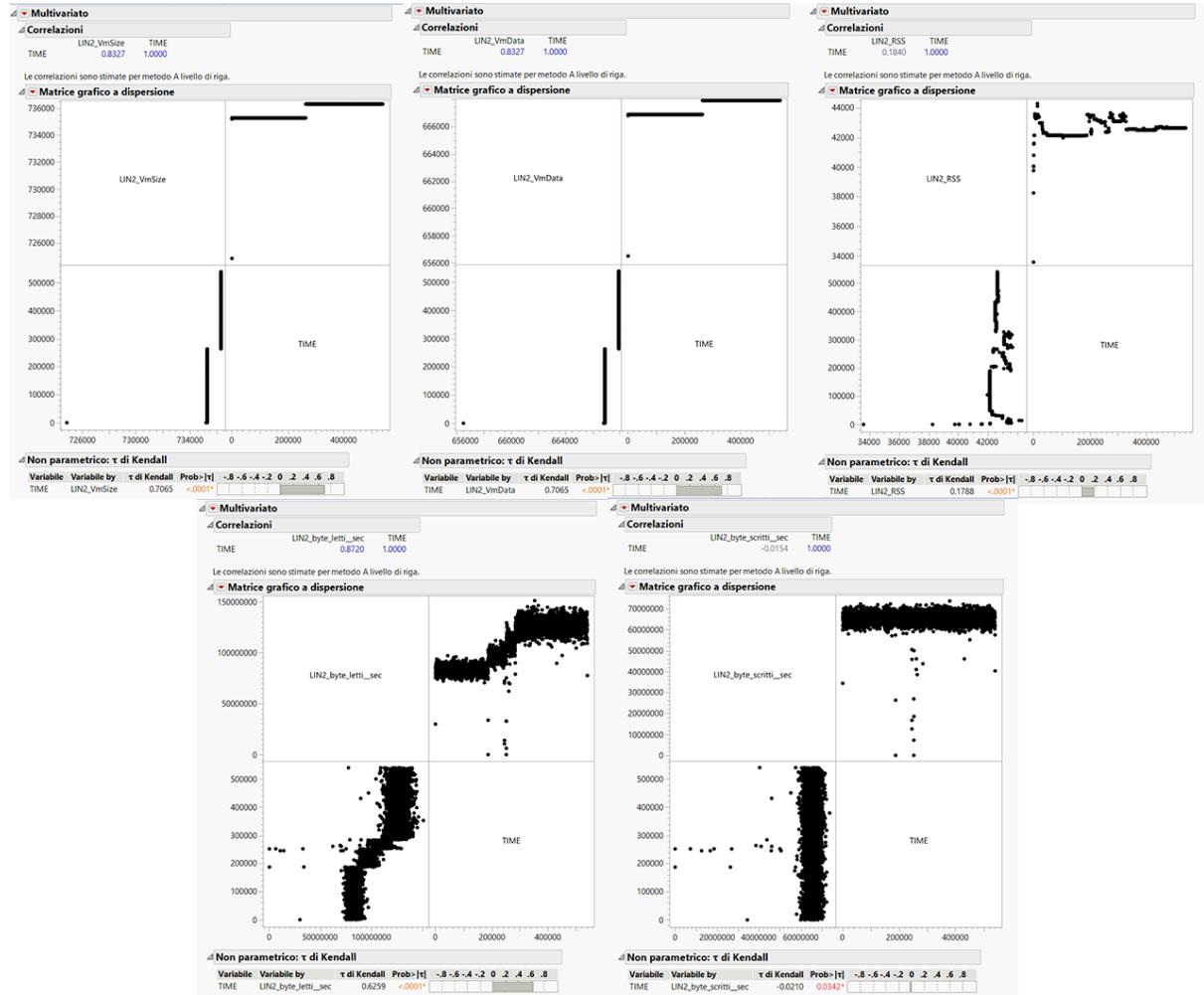


Figura 4.20: Test non parametrico Mann-Kendall

Al fine di favorire la leggibilità, riportiamo di seguito anche una tabella riassuntiva dei risultati ottenuti, evidenziando anche le variabili per cui il test di Mann-Kendall esplicita la presenza di un trend:

Variabili	Tau	P-value	Ipotesi nulla rigettata
LIN2_VmSize	0.7065	0.0001	1
LIN2_VmData	0.7065	0.0001	1
LIN2_RSS	0.1788	0.0001	1
LIN2_byte_letti_sec	0.6259	0.0001	1
LIN2_byte_scritti_sec	-0.0210	0.0342	1

Il test di Mann-Kendall (considerando il tasso di significatività standard pari ad $\alpha = 0.05$) riporta che tutte le variabili presentano trend significativi. Infine, dunque, avendo rilevato un trend, abbiamo utilizzato la procedura di Theil-Sen (implementata tramite un apposito script Python) ed abbiamo calcolato pendenza ed intercetta della retta di regressione ed il relativo intervallo di confidenza come mostrato di seguito.

```
▶ x = np.array(x_column)
y = np.array(y_column)
slope, intercept, low, up = theilslopes(y, x, 0.95)
print("Gradiente: {}. Intervallo: [{},{}]. Intercetta: {}".format(slope, low, up, intercept))

Gradiente: 0.0019128745423298214. Intervallo: [0.0,0.002085369827305311] Intercetta: 735790.3761488455
```

Figura 4.21: Risultati Theil-Sen per la variabile LIN2_VmSize

```
Gradiente: 0.0019128745423298214. Intervallo: [0.0,0.002085369827305311] Intercetta: 667414.3761488455
```

Figura 4.22: Risultati Theil-Sen per la variabile LIN2_VmData

```
Gradiente: 0.0006324666198172874. Intervallo: [0.000555555555555556,0.0007445442875481386] Intercetta: 42384.85453267744
```

Figura 4.23: Risultati Theil-Sen per la variabile LIN2_RSS

```
□ Gradiente: 114.55350983796296. Intervallo: [112.73359327217125,116.37659764826176] Intercetta: 77733384.23784722
```

Figura 4.24: Risultati Theil-Sen per la variabile LIN2_byte_letti_sec

```
□ Gradiente: -0.48861262488646684. Intervallo: [-0.9390005359056806,-0.036319163292847505] Intercetta: 65693615.57629428
```

Figura 4.25: Risultati Theil-Sen per la variabile LIN2_byte_scritti_sec

Dataset 3

Il primo passo per il corretto svolgimento di tale esercizio è quello di eseguire le stime lineari delle variabili *LIN4_VmSize*, *LIN4_VmData*, *LIN4RSS*, *LIN4_byte_letti_sec* e *LIN4_byte_scritti_sec* rispetto alla variabile di predizione *TIME* al fine di determinare l' R^2 .

Riportiamo, dunque, di seguito la stima lineare delle 5 variabili sopra descritte:



Figura 4.26: Stime lineari

Dalla Figura 4.26 è possibile notare che - come indicato alla voce “*R-quadro*” all’interno della finestra “*Riepilogo della stima*” - per le variabili *LIN4_byte_letti_sec* e *LIN4_byte_scritti_sec* il valore del coefficiente R^2 risulta essere molto basso mentre per le variabili *LIN4_VmSize*, *LIN4_VmData* e *LIN4RSS* tale valore risulta essere alto. Questo, dunque, indica che nel primo caso il modello lineare non è sufficiente per poter predire correttamente i dati mentre nel secondo caso andremo a calcolare i residui proprio dal modello di regressione lineare tramite l’apposita funzione di JMP “*Analizza → Modellazione Predittiva → Screening del modello*”.

Fatta tale considerazione, abbiamo successivamente calcolato i residui delle variabili al fine di analizzarne le distribuzioni. Tramite l’apposito strumento del tool JMP, è stato possibile

graficarne i QQ-plot e verificare tramite un semplice test visivo la non normalità di tutte le distribuzioni.

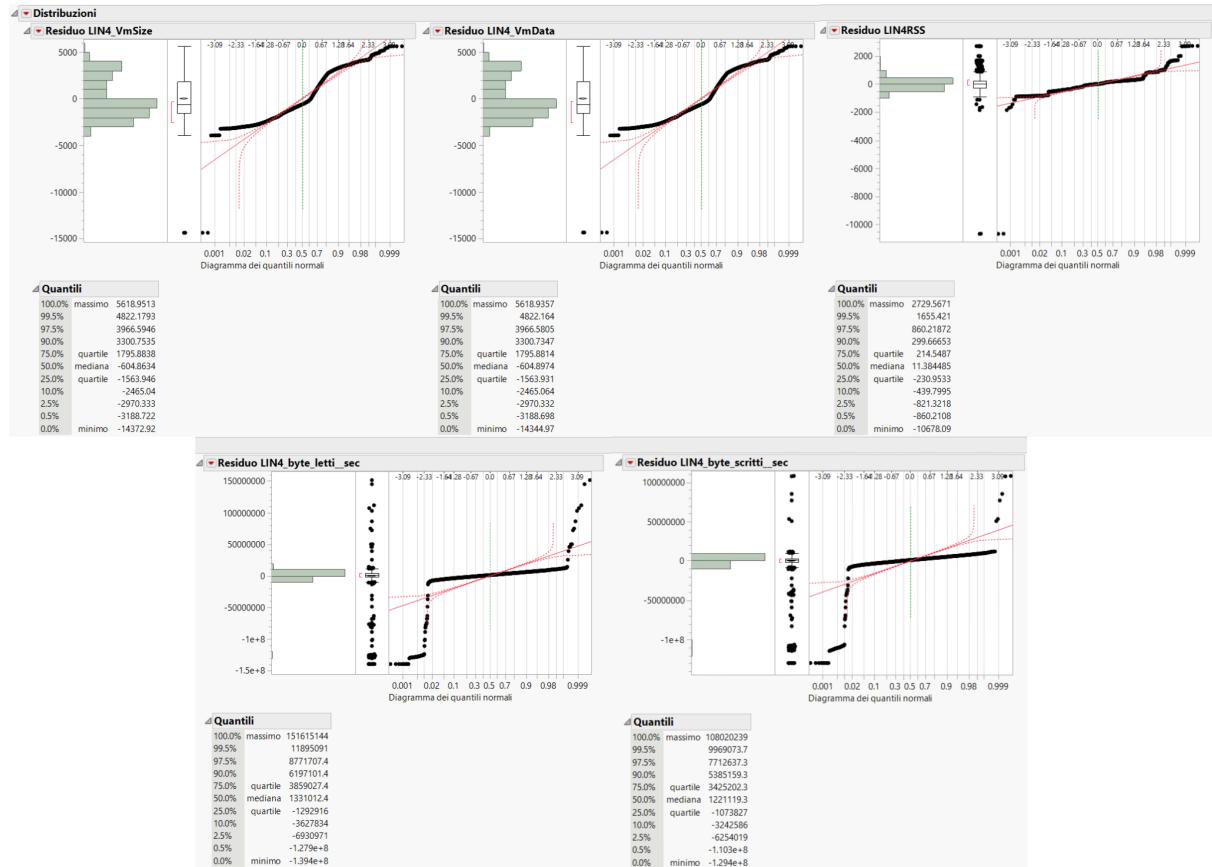


Figura 4.27: Distribuzione dei residui

Tale risultato ci ha condotto alla scelta di utilizzare un test non parametrico al fine di stabilire l'eventuale presenza di un trend nei dati. In particolare, abbiamo scelto il test non parametrico di Mann-Kendall in quanto tale test non richiede che i dati siano lineari o distribuiti normalmente.

Di seguito, riportiamo i risultati ottenuti a valle dell'esecuzione del test di Mann-Kendall per tutte le variabili considerate:

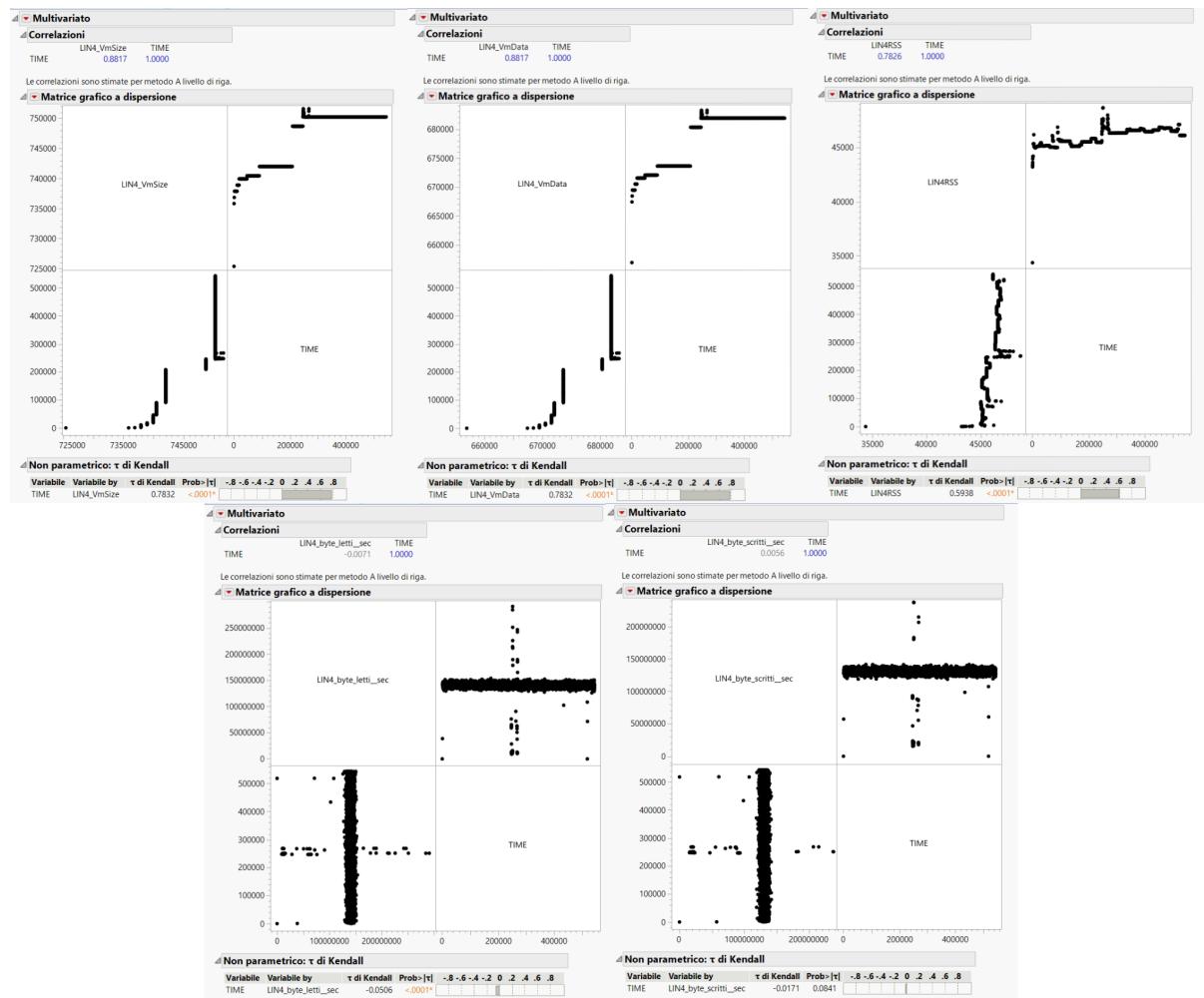


Figura 4.28: Test non parametrico Mann-Kendall

Al fine di favorire la leggibilità, riportiamo di seguito anche una tabella riassuntiva dei risultati ottenuti, evidenziando anche le variabili per cui il test di Mann-Kendall esplicita la presenza di un trend:

Variabili	Tau	P-value	Ipotesi nulla rigettata
LIN4_VmSize	0.7832	0.0001	1
LIN4_VmData	0.7832	0.0001	1
LIN4_RSS	0.5938	0.0001	1
LIN4_byte_letti_sec	-0.0506	0.0001	1
LIN4_byte_scritti_sec	-0.0171	0.0841	0

Il test di Mann-Kendall (considerando il tasso di significatività standard pari ad $\alpha = 0.05$) riporta che tutte le variabili tranne *LIN4_byte_scritti_sec* presentano trend significativi. Infine, dunque, avendo rilevato un trend, abbiamo utilizzato la procedura di Theil-Sen (implementata tramite un apposito script Python) ed abbiamo calcolato pendenza ed intercetta della retta di regressione ed il relativo intervallo di confidenza come mostrato di seguito.

```
x = np.array(x_column)
y = np.array(y_column)
slope, intercept, low, up = theilslopes(y, x, 0.95)
print("Gradiente: {}. Intervallo: [{},{}]. Intercetta: {}".format(slope, low, up, intercept))

Gradiente: 0.021659324522760644. Intervallo: [0.02127364220024293, 0.02202594299368493] Intercetta: 744374.5991189427
```

Figura 4.29: Risultati Theil-Sen per la variabile *LIN4_VmSize*

```
Gradiente: 0.021659324522760644. Intervallo: [0.02127364220024293, 0.02202594299368493] Intercetta: 675970.5991189427
```

Figura 4.30: Risultati Theil-Sen per la variabile *LIN4_VmData*

```
Gradiente: 0.0034579439252336447. Intervallo: [0.0033884297520661156, 0.0035222894881673087] Intercetta: 45386.471028037384
```

Figura 4.31: Risultati Theil-Sen per la variabile *LIN4_RSS*

```
Gradiente: -1.891048815359477. Intervallo: [-2.617641979071421, -1.16775956284153] Intercetta: 141291080.77144608
```

Figura 4.32: Risultati Theil-Sen per la variabile *LIN4_byte_letti_sec*

4.4 Esercizio 4

4.4.1 Traccia

Supponendo di avere un limite massimo alla memoria heap di 1 GByte. Rilevare un eventuale trend di consumo dello heap nell'esperimento in figura. Se rilevato il trend, stimare il tempo in cui lo heap satura (failure prediction).

4.4.2 Svolgimento

Dataset 1

Il primo passo per il corretto svolgimento di tale esercizio è quello di eseguire le stime lineari della variabile *Allocated Heap* rispetto alla variabile di predizione *T(s)* al fine di determinare l' R^2 .

Riportiamo, dunque, di seguito la stima lineare:

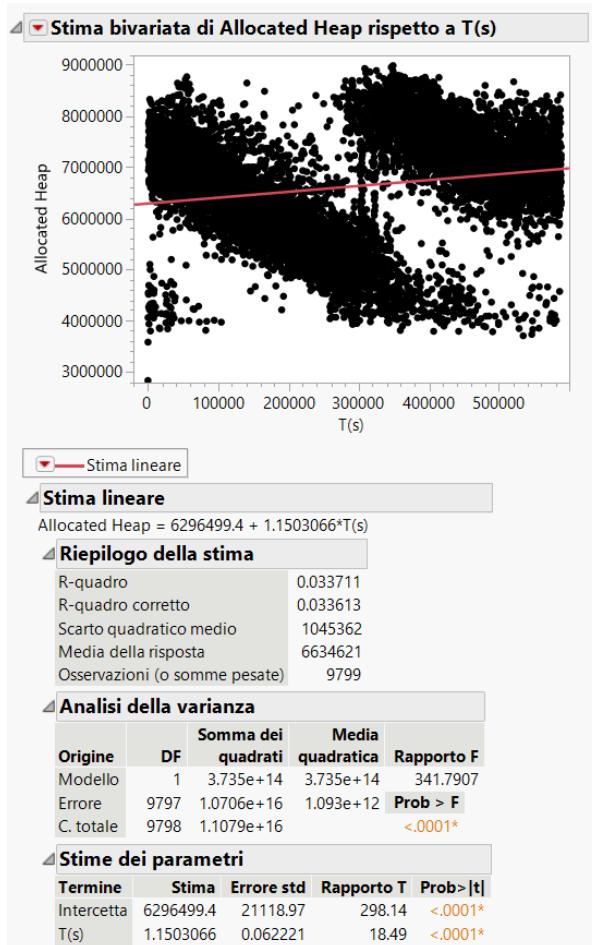


Figura 4.33: Stima lineare

Dalla Figura 4.33 è possibile notare che - come indicato alla voce “*R-quadro*” all’interno della finestra “*Riepilogo della stima*” - per la variabile *Allocated Heap* il valore del coefficiente R^2 risulta essere molto basso. Questo, dunque, indica che il modello lineare non è sufficiente per poter predire correttamente i dati.

Fatta tale considerazione, abbiamo successivamente calcolato i residui delle variabili al fine di analizzarne le distribuzioni. Tramite l’apposito strumento del tool JMP, è stato possibile graficarne i QQ-plot e verificare tramite un semplice test visivo la non normalità di tutte le distribuzioni.

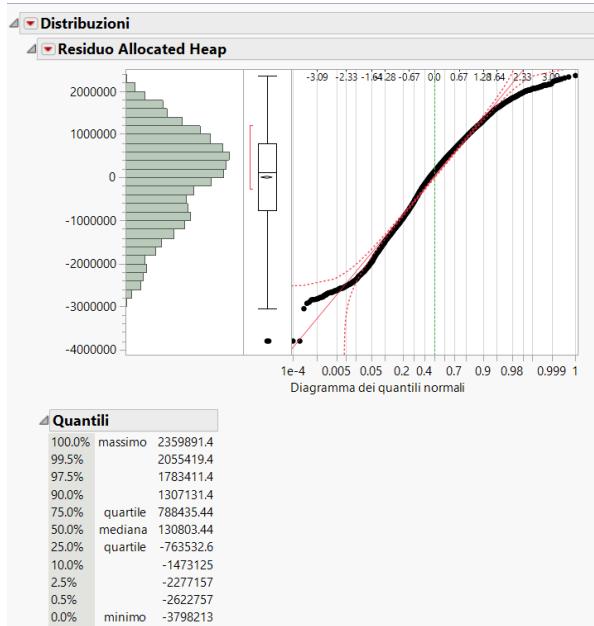


Figura 4.34: Distribuzione dei residui

Tale risultato ci ha condotto alla scelta di utilizzare un test non parametrico al fine di stabilire l’eventuale presenza di un trend nei dati. In particolare, abbiamo scelto il test non parametrico di Mann-Kendall in quanto tale test non richiede che i dati siano lineari o distribuiti normalmente.

Di seguito, riportiamo i risultati ottenuti a valle dell’esecuzione del test di Mann-Kendall per tutte le variabili considerate:

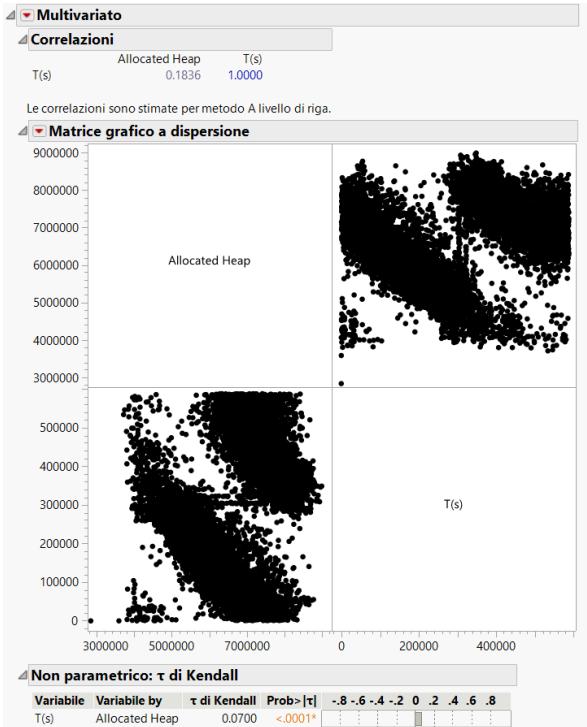


Figura 4.35: Test non parametrico Mann-Kendall

Il test di Mann-Kendall, dunque, riporta un valore di $\tau = 0.0700$ ed un $P - value = 0.0001$. Considerando il tasso di significatività standard pari ad $\alpha = 0.05$ allora possiamo affermare che il test di Mann-Kendall evidenzia la presenza di un trend significativo. Infine, dunque, avendo rilevato un trend, abbiamo utilizzato la procedura di Theil-Sen (implementata tramite un apposito script Python) ed abbiamo calcolato pendenza ed intercetta della retta di regressione ed il relativo intervallo di confidenza come mostrato di seguito. Inoltre, abbiamo realizzato un ulteriore script Python utile per poter calcolare il tempo in cui lo heap satura secondo la seguente formula:

$$\text{allocatedheap} = 6562275,46 + (0,69 \pm 0,12) \cdot T(s)$$

$$T(s) = \frac{10000000000 - 6562275,46}{0,69 \pm 0,12} = 47 \pm 9 \text{ anni}$$

Per completezza, riportiamo di seguito gli script realizzati.

```

[66] x = np.array(x_column)
y = np.array(y_column)
slope, intercept, low, up = theilslopes(y, x, 0.95)
print("Gradiente: {:.2f}. Intervallo: [{:.2f},{:.2f}] Intercetta: {:.2f}".format(slope, low, up, intercept))

Gradiente: 0.6911224682945296. Intervallo: [0.563777264562484,0.8178240089963452] Intercetta: 6562275.461669506

MAX_SIZE = 1000000000
ts1 = (MAX_SIZE - intercept)/(slope + (slope-low))
ts2 = (MAX_SIZE - intercept)/(slope - (slope-low))
media = (ts1+ts2)/2

ts1_anni = ts1 / (365.25*24*60*60)
ts2_anni = ts2 / (365.25*24*60*60)
media_anni = media / (365.25*24*60*60)
conf_anni = media_anni - ts1_anni

print("Il tempo in cui l'heap satura è: {:.2f} ± {:.2f} anni. Intervallo: [{:.2f},{:.2f}]".format(media_anni, conf_anni, ts1_anni, ts2_anni))

Il tempo in cui l'heap satura è: 47.15009850235062 ± 8.687807407831286 anni. Intervallo: [38.46229109451934,55.83790591018192]

```

Figura 4.36: Risultati Theil-Sen e stima

Dataset 2

Il primo passo per il corretto svolgimento di tale esercizio è quello di eseguire le stime lineari della variabile *Allocated Heap* rispetto alla variabile di predizione $T(s)$ al fine di determinare l' R^2 .

Riportiamo, dunque, di seguito la stima lineare:

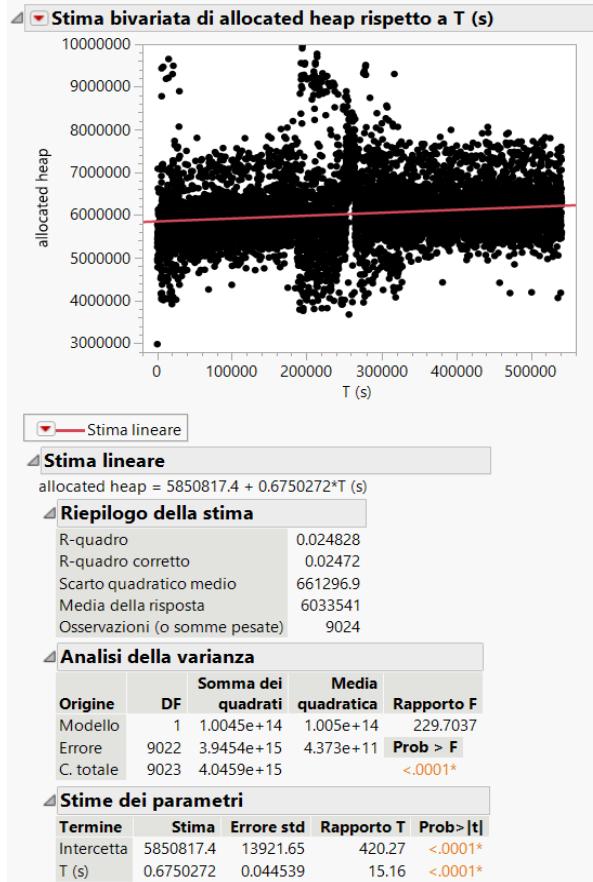


Figura 4.37: Stima lineare

Dalla Figura 4.33 è possibile notare che - come indicato alla voce “*R-quadro*” all’interno della finestra “*Riepilogo della stima*” - per la variabile *Allocated Heap* il valore del coefficiente R^2 risulta essere molto basso. Questo, dunque, indica che il modello lineare non è sufficiente per poter predire correttamente i dati.

Fatta tale considerazione, abbiamo successivamente calcolato i residui delle variabili al fine di analizzarne le distribuzioni. Tramite l’apposito strumento del tool JMP, è stato possibile graficarne i QQ-plot e verificare tramite un semplice test visivo la non normalità di tutte le distribuzioni.

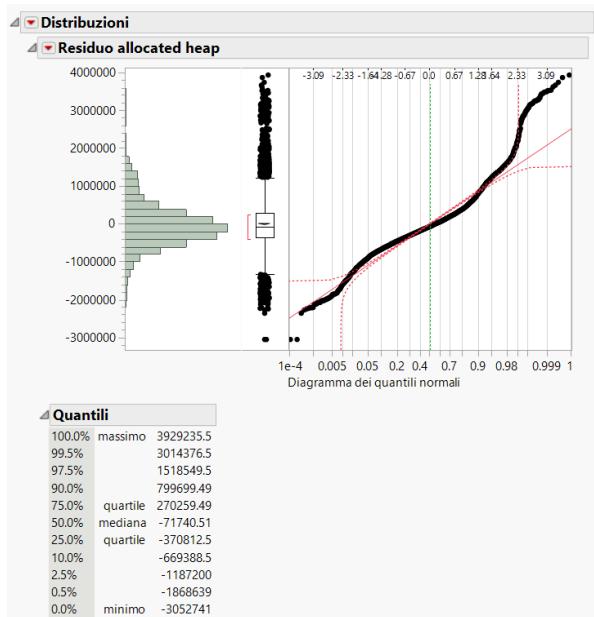


Figura 4.38: Distribuzione dei residui

Tale risultato ci ha condotto alla scelta di utilizzare un test non parametrico al fine di stabilire l’eventuale presenza di un trend nei dati. In particolare, abbiamo scelto il test non parametrico di Mann-Kendall in quanto tale test non richiede che i dati siano lineari o distribuiti normalmente.

Di seguito, riportiamo i risultati ottenuti a valle dell’esecuzione del test di Mann-Kendall per tutte le variabili considerate:

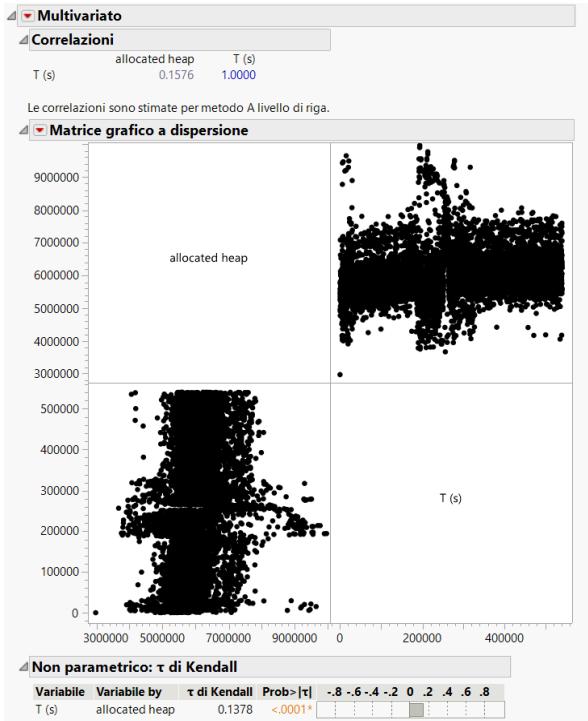


Figura 4.39: Test non parametrico Mann-Kendall

Il test di Mann-Kendall, dunque, riporta un valore di $\tau = 0.1378$ ed un $P - value = 0.0001$. Considerando il tasso di significatività standard pari ad $\alpha = 0.05$ allora possiamo affermare che il test di Mann-Kendall evidenzia la presenza di un trend significativo. Infine, dunque, avendo rilevato un trend, abbiamo utilizzato la procedura di Theil-Sen (implementata tramite un apposito script Python) ed abbiamo calcolato pendenza ed intercetta della retta di regressione ed il relativo intervallo di confidenza come mostrato di seguito. Inoltre, abbiamo realizzato un ulteriore script Python utile per poter calcolare il tempo in cui lo heap satura secondo la seguente formula:

$$\text{allocatedheap} = 5780180,56 + (0,67 \pm 0,07) \cdot T(s)$$

$$T(s) = \frac{10000000000 - 5780180,56}{0,67 \pm 0,07} = 47 \pm 5 \text{ anni}$$

Per completezza, riportiamo di seguito gli script realizzati.

```

[70] x = np.array(x_column)
y = np.array(y_column)
slope, intercept, low, up = theilslopes(y, x, 0.95)
print("Gradiente: {:.2f}. Intervallo: [{:.2f},{:.2f}] Intercetta: {:.2f}".format(slope, low, up, intercept))

Gradiente: 0.6709499304572171. Intervallo: [0.6048638415610893, 0.7366538131962297] Intercetta: 5780180.563324536

MAX_SIZE = 1000000000
ts1 = (MAX_SIZE - intercept)/(slope + (slope-low))
ts2 = (MAX_SIZE - intercept)/(slope - (slope-low))
media = (ts1+ts2)/2

ts1_anni = ts1 / (365.25*24*60*60)
ts2_anni = ts2 / (365.25*24*60*60)
media_anni = media / (365.25*24*60*60)
conf_anni = media_anni - ts1_anni

print("Il tempo in cui l'heap satura è: {:.2f} ± {:.2f} anni. Intervallo: [{:.2f},{:.2f}]".format(media_anni, conf_anni, ts1_anni, ts2_anni))

Il tempo in cui l'heap satura è: 47.41570645304874 ± 4.670271877953034 anni. Intervallo: [42.74543457509571, 52.08597833100176]

```

Figura 4.40: Risultati Theil-Sen e Stima

Dataset 3

Il primo passo per il corretto svolgimento di tale esercizio è quello di eseguire le stime lineari della variabile *Allocated Heap* rispetto alla variabile di predizione *T(s)* al fine di determinare l' R^2 .

Riportiamo, dunque, di seguito la stima lineare:

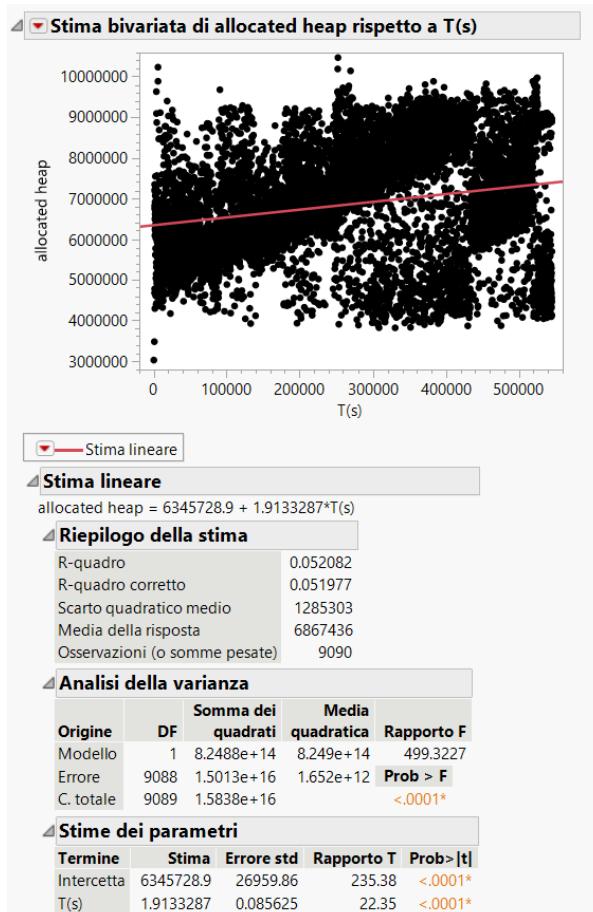


Figura 4.41: Stima lineare

Dalla Figura 4.33 è possibile notare che - come indicato alla voce “*R-quadro*” all’interno della finestra “*Riepilogo della stima*” - per la variabile *Allocated Heap* il valore del coefficiente R^2 risulta essere molto basso. Questo, dunque, indica che il modello lineare non è sufficiente per poter predire correttamente i dati.

Fatta tale considerazione, abbiamo successivamente calcolato i residui delle variabili al fine di analizzarne le distribuzioni. Tramite l’apposito strumento del tool JMP, è stato possibile graficarne i QQ-plot e verificare tramite un semplice test visivo la non normalità di tutte le distribuzioni.

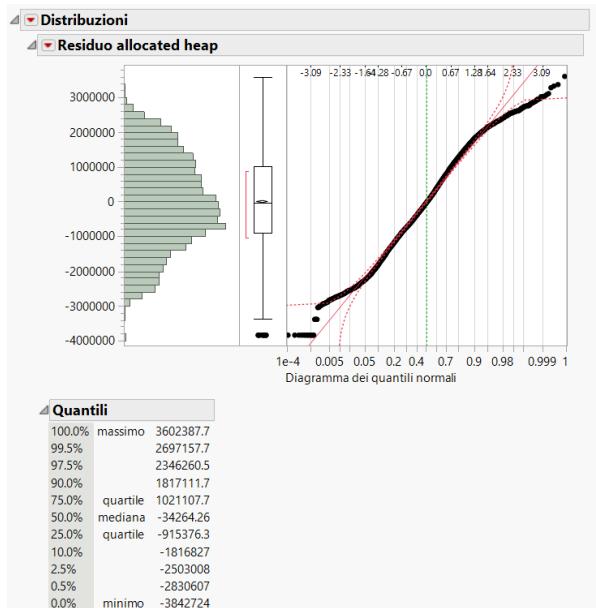


Figura 4.42: Distribuzione dei residui

Tale risultato ci ha condotto alla scelta di utilizzare un test non parametrico al fine di stabilire l’eventuale presenza di un trend nei dati. In particolare, abbiamo scelto il test non parametrico di Mann-Kendall in quanto tale test non richiede che i dati siano lineari o distribuiti normalmente.

Di seguito, riportiamo i risultati ottenuti a valle dell’esecuzione del test di Mann-Kendall per tutte le variabili considerate:

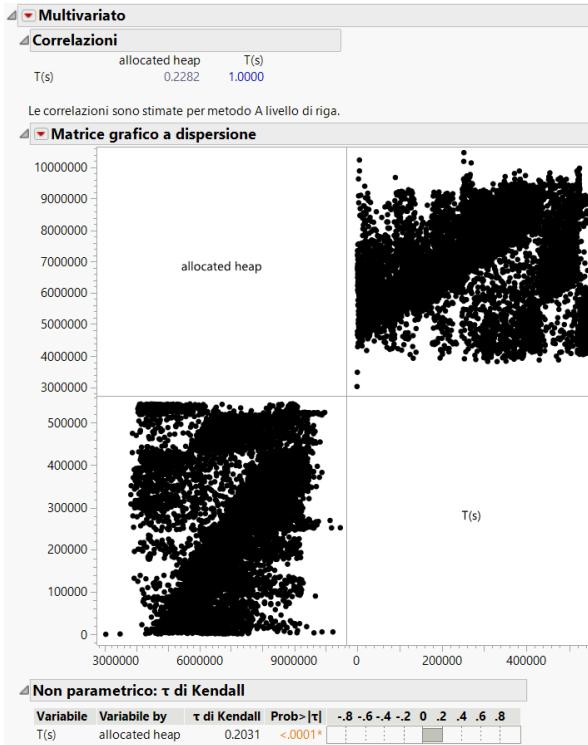


Figura 4.43: Test non parametrico Mann-Kendall

Il test di Mann-Kendall, dunque, riporta un valore di $\tau = 0.0700$ ed un $P - value = 0.0001$. Considerando il tasso di significatività standard pari ad $\alpha = 0.05$ allora possiamo affermare che il test di Mann-Kendall evidenzia la presenza di un trend significativo. Infine, dunque, avendo rilevato un trend, abbiamo utilizzato la procedura di Theil-Sen (implementata tramite un apposito script Python) ed abbiamo calcolato pendenza ed intercetta della retta di regressione ed il relativo intervallo di confidenza come mostrato di seguito. Inoltre, abbiamo realizzato un ulteriore script Python utile per poter calcolare il tempo in cui lo heap satura secondo la seguente formula:

$$\text{allocatedheap} = 6041585,56 + (2,90 \pm 0,19) \cdot T(s)$$

$$T(s) = \frac{1000000000 - 6041585,56}{2,90 \pm 0,19} = 11 \pm 1 \text{ anni}$$

Per completezza, riportiamo di seguito gli script realizzati.

```
x = np.array(x_column)
y = np.array(y_column)
slope, intercept, low, up = theilslopes(y, x, 0.95)
print("Gradiente: {}. Intervallo: [{},{}]. Intercetta: {}".format(slope, low, up, intercept))

Gradiente: 2.9030932760062105. Intervallo: [2.714759535655058, 3.0923529411764705] Intercetta: 6041585.556431387

MAX_SIZE = 1000000000
ts1 = (MAX_SIZE - intercept)/(slope + (slope-low))
ts2 = (MAX_SIZE - intercept)/(slope - (slope-low))
media = (ts1+ts2)/2

ts1_anni = ts1 / (365.25*24*60*60)
ts2_anni = ts2 / (365.25*24*60*60)
media_anni = media / (365.25*24*60*60)
conf_anni = media_anni - ts1_anni

print("Il tempo in cui l'heap satura è: {} ± {} anni. Intervallo: [{},{}].".format(media_anni, conf_anni, ts1_anni, ts2_anni))

Il tempo in cui l'heap satura è: 10.895191603744586 ± 0.7068089074280586 anni. Intervallo: [10.188382696316527, 11.60200051117264]
```

Figura 4.44: Risultati Theil-Sen e Stima

Capitolo 5

Reliability

5.1 Esercizio 1

5.1.1 Traccia

Trovare $R(t)$ e $MTTF$ per il sistema riportato di seguito. Nel calcolo di MTTF si assuma che tutte le componenti siano identiche e che falliscono casualmente con lo stesso failure rate λ .

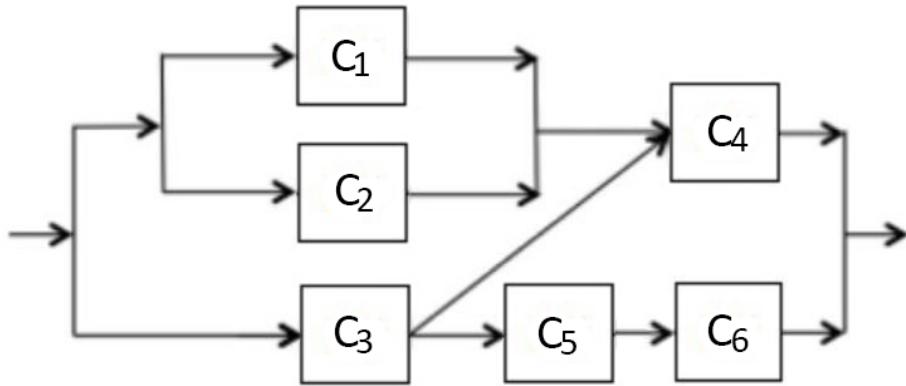


Figura 5.1: Sistema 1

5.1.2 Svolgimento

Il sistema in esame presenta sei componenti con proprietà di reliability note e uguali per tutti, pari a:

$$R(t) = e^{-\lambda t}$$

Esaminando il sistema (figura 5.1), fin da subito, risultano evidenti le relazioni di parallelismo tra le componenti C_1 e C_2 e di serie tra C_5 e C_6 . Dunque la prima cosa da fare è applicare le formule standard di riduzione serie parallelo, ottenendo così il seguente risultato:

$$R_5 - R_6 = R_5 * R_6$$

$$R_1//R_2 = 1 - (1 - R_1)(1 - R_2)$$

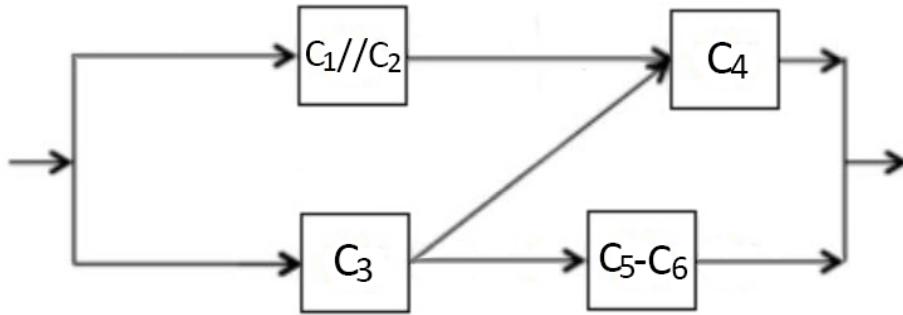


Figura 5.2: Sistema 2

Procedendo con l'analisi, possiamo notare che non basta considerare soltanto le tecniche standard di riduzione serie parallelo, infatti, arrivati a questo punto, è necessario utilizzare la tecnica del condizionamento.

Andiamo innanzitutto ad individuare, nel nuovo sistema (figura 5.2), tutti i success path presenti, ovvero:

1. $C_1 - C_4$
2. $C_2 - C_4$
3. $C_3 - C_4$
4. $C_3 - C_5 - C_6$

A questo punto, considerando i success path sopra elencati, possiamo calcolare l'upper-bound:

$$R_{sys} \leq 1 - (1 - R^2)^3(1 - R^3) \leq 3R^2 + R^3 + R^9 - 3R^5 - 3R^7$$

Successivamente, abbiamo deciso di effettuare il condizionamento rispetto al componente C_4 .

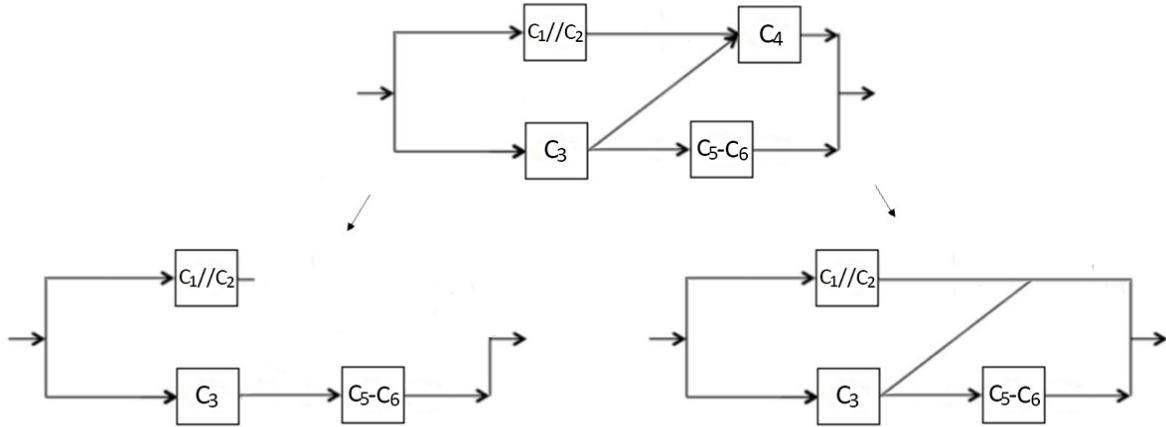


Figura 5.3: Sistema 3

A valle del condizionamento, il sistema risulta facilmente risolvibile. Infatti, condizionando il sistema con il fallimento del componente C_4 , appare evidente la relazione di serie tra il componente C_3 e il blocco $C_5 - C_6$. Condizionando, invece, con il “non fallimento” di C_4 possiamo considerare un parallelo tra $C_1 // C_2$ e C_3 .

$$P(system^{UP}) = P(S^{UP} | C_4^{UP}) * P(C_4^{UP}) + P(S^{UP} | C_4^{DOWN}) * P(C_4^{DOWN})$$

Infine, possiamo finalmente calcolare la reliability complessiva del sistema e, di conseguenza, il Mean Time To Failure (MTTF):

$$R_{sys} = R \left[1 - (1 - R)^3 \right] + (1 - R)R^3 = e^{-\lambda t} \left[1 - (1 - e^{-\lambda t})^3 \right] + (1 - e^{-\lambda t})e^{-3\lambda t} = -2e^{-3\lambda t} + 3e^{-2\lambda t}$$

$$MTTF_{sys} = \int_0^\infty R_{sys} dt = \int_0^\infty -2e^{-3\lambda t} + 3e^{-2\lambda t} dt = -\frac{2}{3\lambda} + \frac{3}{2\lambda} = \frac{5}{6\lambda}$$

5.2 Esercizio 2

5.2.1 Traccia

Confrontare due differenti schemi per l'aumento della reliability di un sistema utilizzando la ridondanza. Supponiamo che il sistema richieda s componenti identici in serie per un corretto funzionamento. Inoltre, supponiamo di avere a disposizione $m \times s$ componenti. Dato

che la reliability di una singola componente è r , si derivino le espressioni per la reliability delle due configurazioni. Per $m=5$ e $s=3$, confrontare le due espressioni come funzione del mission time t . Sia $MTTF$ della componente pari a 1400 ore.

Tra i due schemi, mostrati nella figura sottostante, quale fornirà una reliability maggiore? Modificare lo schema con minore reliability in modo da raggiungere la stessa reliability dell'altro con $MTTF=1400h$.

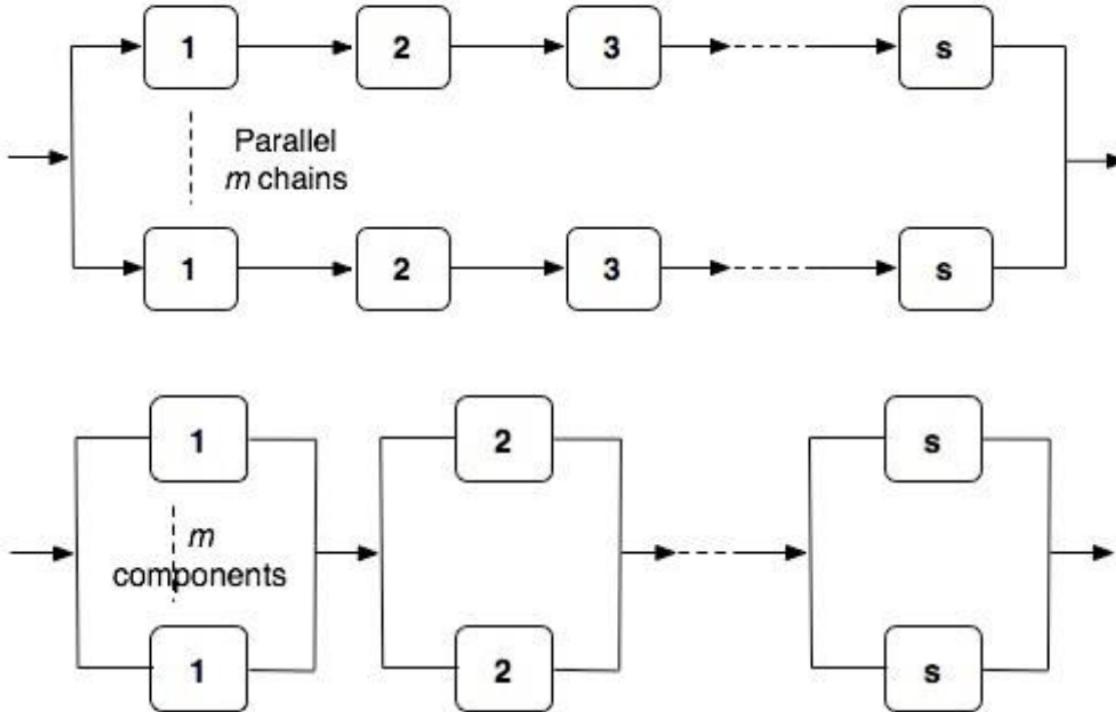


Figura 5.4: Sistema 4

5.2.2 Svolgimento

Da un'analisi preliminare possiamo affermare che il secondo sistema risulta essere più reliable del primo. Questo perché il numero di success path del secondo sistema è molto maggiore rispetto al primo sistema. Per confermare tali analisi andiamo a calcolare la reliability dei due sistemi.

Innanzitutto il primo sistema è composto dal parallelo di m catene, ognuna delle quali è composta dalla serie di s sottosistemi tutti uguali tra loro. La reliability di ogni singola catena è pari a:

$$R_C = R^s$$

Quindi la reliability del primo sistema è uguale a:

$$R_{S_1} = [1 - (1 - R^s)^m]$$

Il secondo sistema, invece, è composto dalla serie di s paralleli di m sottosistemi. La reliability di ogni singolo parallelo è pari a:

$$R_P = 1 - (1 - R)^m$$

Quindi la reliability del secondo sistema è uguale a:

$$R_{S_2} = [1 - (1 - R)^m]^s$$

Sapendo che $s=3$, $m=5$ e $R = e^{-\lambda t}$, dove $\lambda = \frac{1}{1400h}$, possiamo allora confrontare la reliability dei due sistemi al variare del mission time t :

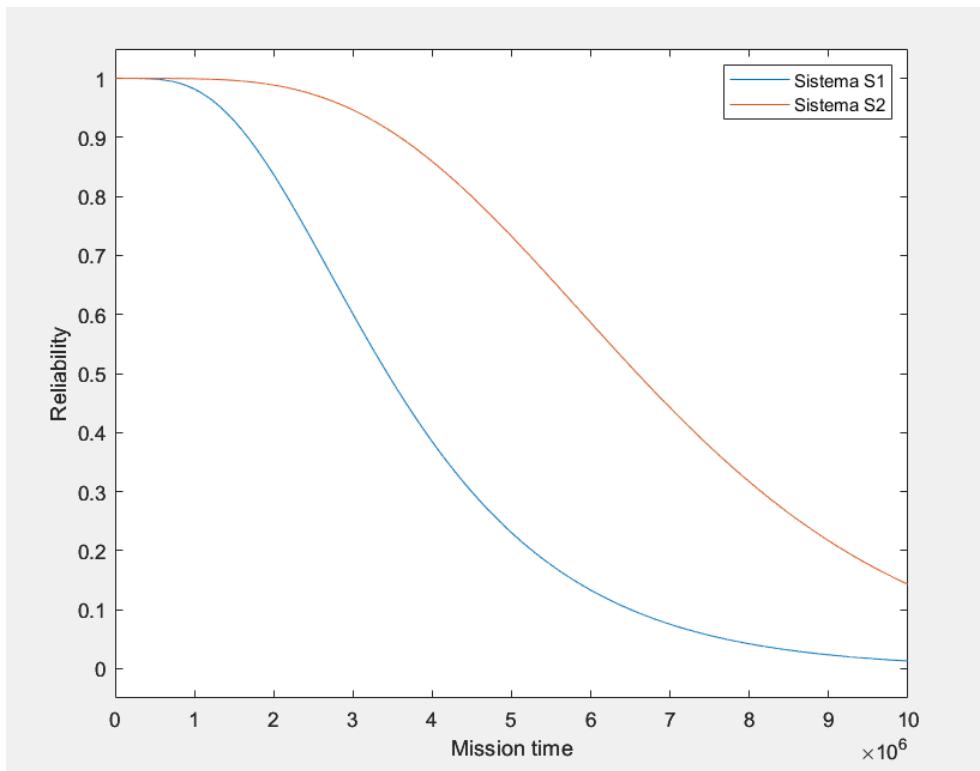


Figura 5.5: Confronto Sistemi

Dalla figura 5.5 si può notare che il secondo sistema è significativamente più reliable del primo indipendentemente dal valore del mission time t . Andiamo ora ad apportare delle modifiche al primo sistema al fine di garantire la stessa reliability del secondo per uno specifico mission time. Nel nostro caso i due sistemi devono presentare la stessa reliability a 1400 ore ($MTTF=1400h$).

Innanzitutto andiamo a calcolare la reliability del secondo sistema quando $t=1400h$:

$$R_{S_2}(1400h) = 0.727 = 72.7\%$$

Per garantire la stessa reliability del secondo sistema è necessario modificare nel primo sistema il numero m di catene in parallelo. E' possibile calcolare tale valore andando a svolgere il seguente calcolo:

$$R_{S_1} = R_{S_2} \implies 1 - (1 - R^3)^m = 0.727 \implies m \cong 26$$

In conclusione, per ottenere la stessa reliability del secondo sistema, è necessario aumentare il numero di catene del primo sistema a 26.

5.3 Esercizio 3

5.3.1 Traccia

L'architettura di una rete di computer in un sistema bancario prende il nome di skip-ring. Tale architettura è progettata per consentire ai processori di comunicare anche dopo che si sono verificati guasti sui nodi. Ad esempio, se il nodo 1 fallisce, il nodo 8 può bypassare il nodo guasto instradando i dati sulla connessione alternativa che collega i nodi 8 e 2. Supponendo che i collegamenti siano perfetti e che ciascun nodo abbia una reliability pari a R_m , derivare un'espressione per la reliability della rete. Se R_m segue la legge di fallimento esponenziale e il tasso di guasto di ciascun nodo è di 0,005 guasti all'ora, determinare la reliability del sistema dopo un periodo di 48 ore.

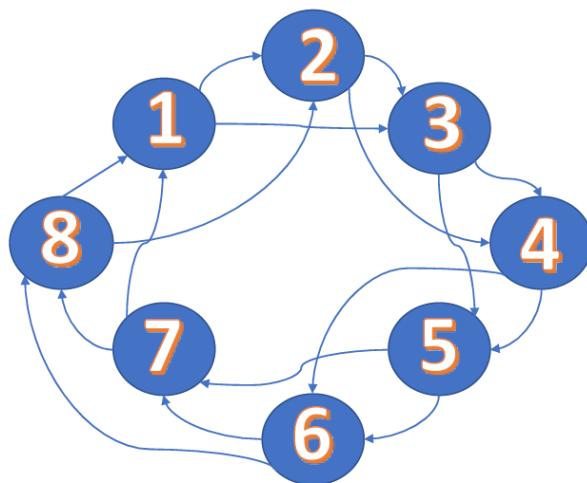


Figura 5.6: Sistema 5

5.3.2 Svolgimento

Nella topologia di rete skip-ring, ciascun nodo è connesso non solo ai nodi adiacenti, ma anche a quelli successivi. È cruciale che non si verifichi il fallimento di due nodi adiacenti affinché la rete funzioni correttamente. In tal caso, la rete non sarebbe in grado di instradare i dati su un collegamento alternativo, compromettendo la sua capacità di comunicazione. Per calcolare la reliability della rete skip-ring, si può applicare la formula per i sistemi *M-out-of-N*. Questi sistemi sono costituiti da N componenti, ciascuno con una propria reliability R_m , di cui almeno M devono essere funzionanti. In generale, la formula per la reliability di un sistema M-out-of-N è data da:

$$R_{MN} = \sum_{i=0}^{N-M} \binom{N}{i} R_M^{N-i} (1 - R_m)^i$$

Per calcolare la reliability totale della rete skip-ring, occorre considerare tutte le possibili configurazioni di funzionamento del sistema e calcolare la reliability di ognuna di esse. Successivamente, si devono sommare le reliability ottenute dalle diverse configurazioni per ottenere, quindi, la reliability totale della rete. Andiamo ad analizzare tutte le possibili configurazioni di funzionamento del sistema:

- *Nessun nodo fallito*, la reliability di questa configurazione è la seguente:

$$\binom{8}{0} R_m^8 = R_m^8$$

- *1 nodo fallito*, la reliability di questa configurazione è la seguente:

$$\binom{8}{1} R_m^7 (1 - R_m) = 8R_m^7 (1 - R_m)$$

- *2 nodi falliti*, in questa configurazione esistono 8 combinazioni in cui il sistema non funziona. Ciò avviene quando i due nodi falliti sono adiacenti. La reliability di questa configurazione è la seguente:

$$\left[\binom{8}{2} - 8 \right] R_m^6 (1 - R_m)^2 = 20R_m^6 (1 - R_m)^2$$

- *3 nodi falliti*, in questa configurazione esistono 40 combinazioni in cui il sistema non funziona in quanto:

- 8 combinazioni si ottengono quando tutti i nodi falliti sono adiacenti;
- 32 combinazioni si ottengono quando solo 2 nodi falliti sono adiacenti;

La reliability di questa configurazione è la seguente:

$$\left[\binom{8}{3} - 40 \right] R_m^5 (1 - R_m)^3 = 16 R_m^5 (1 - R_m)^3$$

- *4 nodi falliti*, in questa configurazione il sistema funziona solo in due circostanze:
 - quando i nodi 1, 3, 5, 7 funzionano ed i nodi 2, 4, 6, 8 non funzionano;
 - quando i nodi 1, 3, 5, 7 non funzionano ed i nodi 2, 4, 6, 8 funzionano.

La reliability di questa configurazione è la seguente:

$$2R_m^4 (1 - R_m)^4$$

- *5, 6, 7, 8 nodi falliti*, il sistema non funziona per queste configurazioni.

Ora, supponendo che la reliability del singolo componente segua la legge di fallimento esponenziale con un failure rate pari a $\lambda = 0.005$ guasti all'ora, è possibile riscrivere R_m nel seguente modo:

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

La reliability totale della rete è la seguente:

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

Ora, calcoliamo la reliability a 48 ore:

$$R_{sys}(48h) = 0.72888 \cong 0.729$$

Di seguito riportiamo il grafico della reliability di R_{sys} :

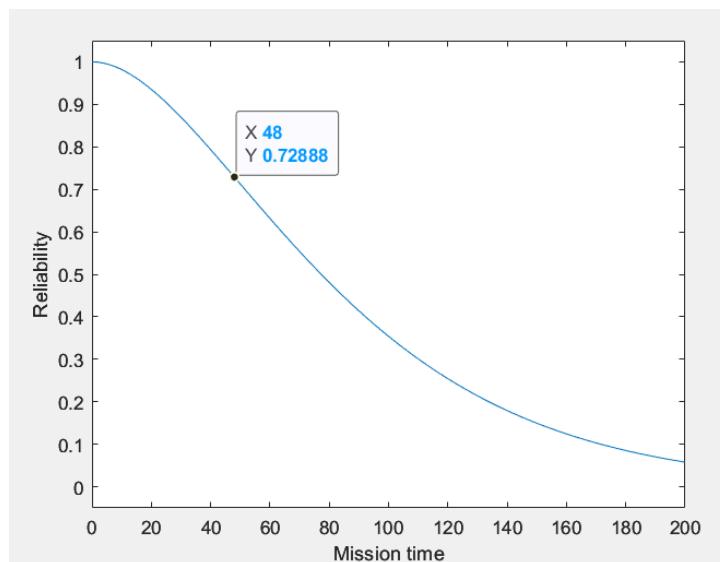


Figura 5.7: Grafico della reliability

5.4 Esercizio 4

5.4.1 Traccia

Confrontare la reliability dei seguenti schemi, supponendo una distribuzione dei guasti esponenziale con i seguenti valori:

- $MTTF_A = 900 \text{ h}$
- $MTTF_B = 7000 \text{ h}$
- $MTTF_C = 1000 \text{ h}$

5.4.2 Svolgimento

Schema 1



Figura 5.8: Schema 1

Osservando la Figura 5.8, fin da una prima analisi qualitativa possiamo affermare che il sistema 1 risulterà essere più reliable in quanto, considerando il fatto che il componente a presenta una replica, il fallimento di tale componente non causerà il fallimento dell'intero sistema. Nel sistema 2, invece, tale ridondanza non è presente.

Ora, analizzando quantitativamente i sistemi, possiamo affermare che il sistema 1 è composto dal parallelo di due serie. La reliability del sistema 1, dunque, sarà data da:

$$R_1 = 1 - (1 - R_a R_b)(1 - R_a R_c)$$

Il sistema 2, invece, è composto dalla serie del componente a ed il parallelo dei componenti b e c . La reliability del sistema 2, dunque, sarà data da:

$$R_2 = R_a(1 - (1 - R_b)(1 - R_c))$$

A questo punto, è possibile particolarizzare i valori delle reliability sopra mostrate ponendo $R = e^{-\lambda t}$. Facendo ciò, abbiamo plottato il risultato ottenuto tramite un apposito script Matlab. Riportiamo il grafico di seguito.

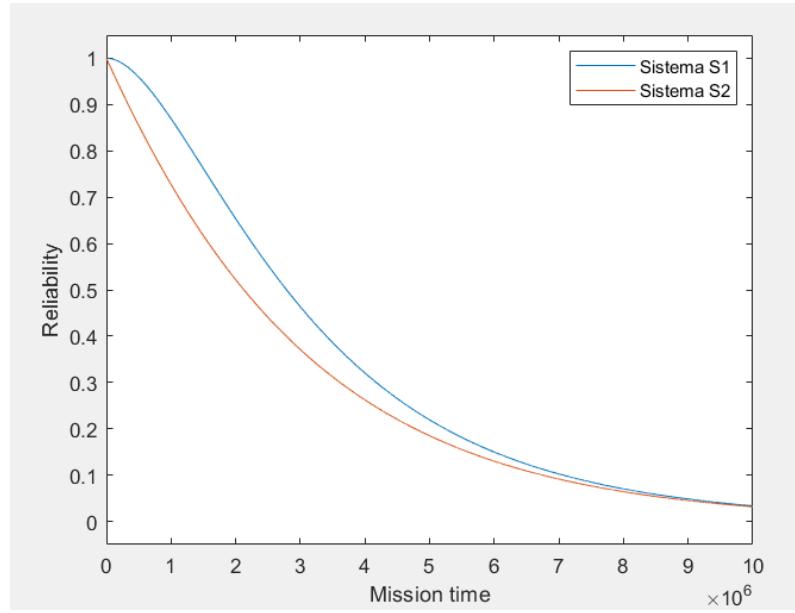


Figura 5.9: Confronto Schema 1

Schema 2

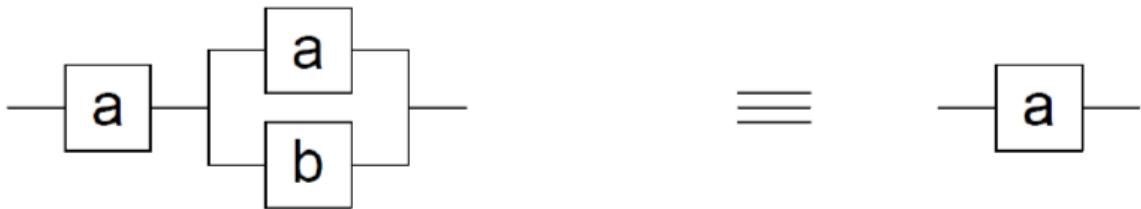


Figura 5.10: Schema 2

Anche in questo caso, già da una prima analisi qualitativa potremmo affermare che la reliability del sistema 2 sarà migliore rispetto a quella del sistema 1 in quanto sappiamo che la reliability della serie di più elementi è sicuramente sempre minore (o al più uguale) della reliability dell'elemento meno affidabile della serie.

Passando ad un'analisi quantitativa, tale risultato atteso viene confermato. Infatti, la reliability del sistema 1 sarà data da:

$$R_1 = R_a(1 - (1 - R_a)(1 - R_b))$$

La reliability del sistema 2, invece, è facilmente calcolabile come $R_2 = R_a$.

Il grafico ottenuto tramite lo script Matlab - che riportiamo di seguito - conferma che il secondo sistema funziona meglio del primo.

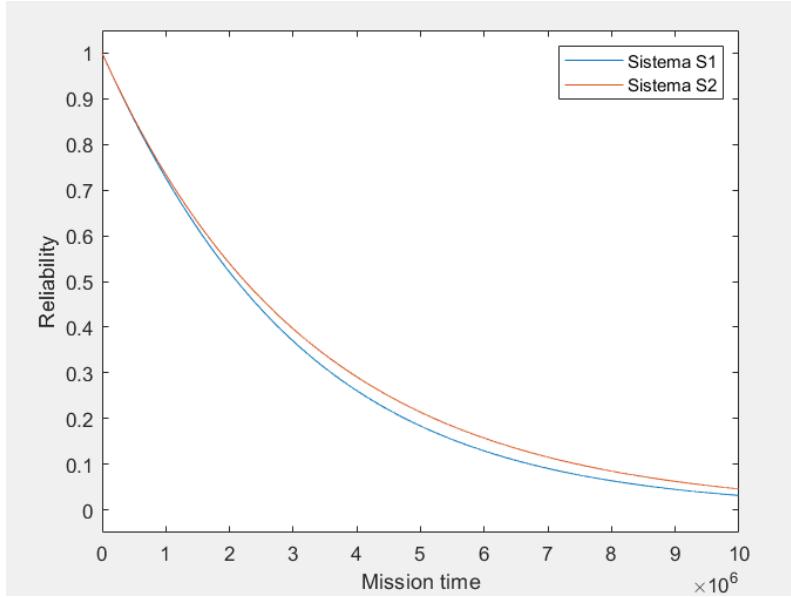


Figura 5.11: Confronto Schema 2

Schema 3

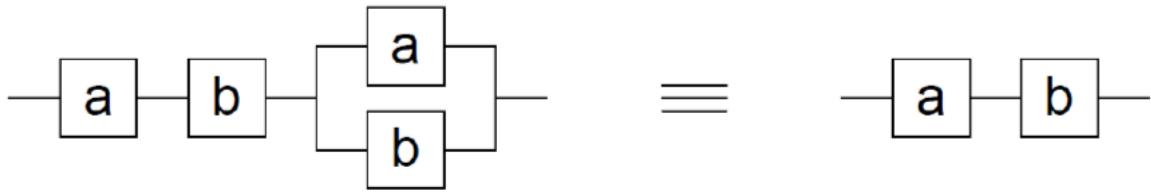


Figura 5.12: Schema 3

Le stesse considerazioni fatte per il precedente esercizio possono essere applicate anche in questo caso. Dalla Figura 5.12, infatti, notiamo che la reliability del sistema 2 sarà sicuramente migliore rispetto a quella del sistema 1 in quanto quest'ultimo aggiunge in serie ai componenti a e b anche un parallelo tra a e b .

La reliability del sistema 1, dunque, sarà data da:

$$R_1 = R_a R_b (1 - (1 - R_a)(1 - R_b))$$

La reliability del sistema 2, invece, sarà data da:

$$R_2 = R_a R_b$$

Plotando tali risultati, si ottiene il seguente grafico.

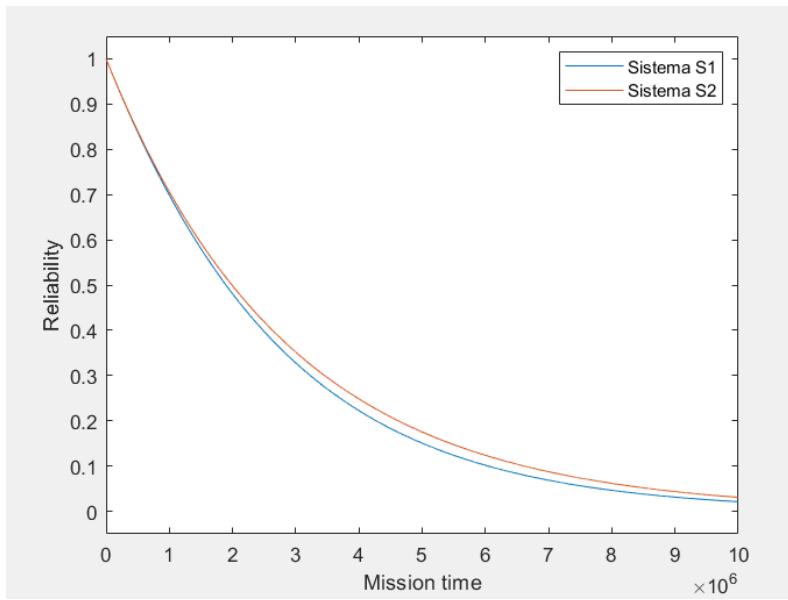


Figura 5.13: Confronto Schema 3

Schema 4



Figura 5.14: Sistema 4 - Schema 4

In quest'ultimo caso, già da una prima analisi qualitativa ci aspettiamo che il sistema 1 risulti più reliable del secondo in quanto presenta ridondanza del componente a rispetto al sistema 2.

Analiticamente, infatti, otteniamo che la reliability del sistema 1 sarà data da:

$$R_1 = 1 - (1 - R_a)(1 - R_a R_b)$$

Per quanto riguarda, invece, la reliability del sistema 2 otteniamo:

$$R_2 = R_a$$

Come atteso, anche graficando tali risultati otteniamo che il sistema 1 risulta essere più reliable del sistema 2:

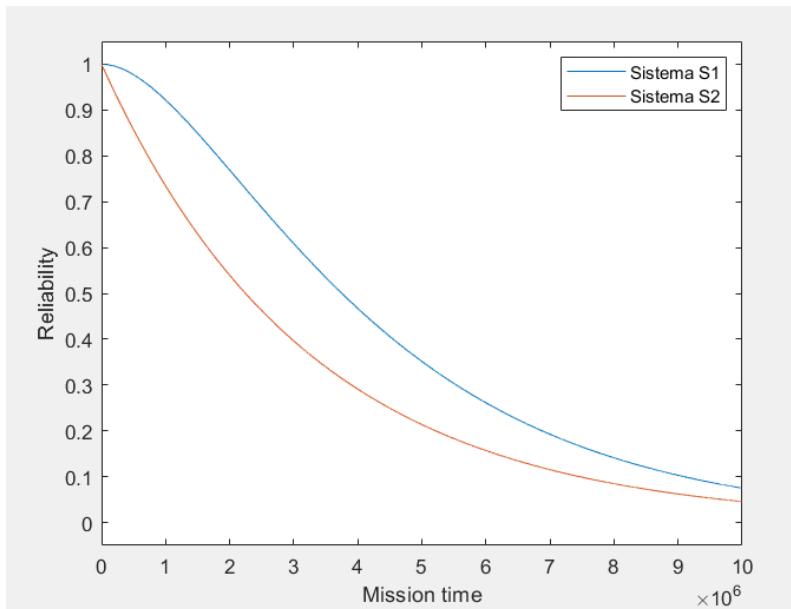


Figura 5.15: Confronto Schema 4

5.5 Esercizio 5

5.5.1 Traccia

Il sistema mostrato nella figura sottostante è un sistema di elaborazione per un elicottero. Il sistema ha processori e terminali di controllo remoto ridondanti. Due bus sono utilizzati nel sistema. Ogni bus è anche ridondante. La parte interessante del sistema è l'equipaggiamento di navigazione. L'aereo può essere completamente pilotato utilizzando il sistema di navigazione inerziale (INS). Se l'INS fallisce, l'aereo può essere pilotato utilizzando la combinazione del sistema di navigazione Doppler e dell'altitudine del sistema di riferimento di direzione (AHRS). Il sistema contiene tre unità AHRS, di cui ne è necessaria solo una. Questo è un esempio di ridondanza funzionale in cui i dati provenienti dall'AHRS e dal sistema Doppler possono essere utilizzati al posto dell'INS, in caso di fallimento dell'INS. A causa degli altri sensori e strumentazioni, entrambi i bus sono necessari affinché il sistema funzioni correttamente, indipendentemente dalla modalità di navigazione che viene utilizzata.

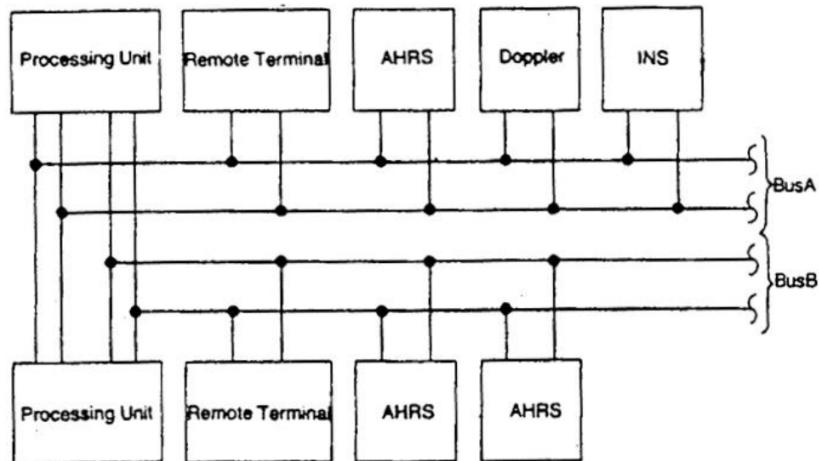


Figura 5.16: Sistema 6

Per questo esercizio, svolgere i seguenti punti:

- Disegnare il diagramma a blocchi della reliability del sistema;
- Disegnare l'albero delle guasti del sistema e analizzare i minimal cutsets;
- Calcolare la reliability per un volo di un'ora utilizzando i valori MTTF dati nella tabella sottostante. Supporre che si applichi la legge di fallimento esponenziale e che la copertura dei guasti sia perfetta;

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

- Ripetere il punto c, ma questa volta, incorporare un fattore di copertura per il rilevamento dei guasti e la riconfigurazione delle unità di elaborazione. Utilizzando gli stessi dati di guasto, determinare il valore approssimativo della copertura dei guasti necessaria per ottenere una affidabilità (alla fine di un'ora) di 0,99999.

5.5.2 Svolgimento

Punto a

Le proprietà fondamentali di questo sistema sono le seguenti:

- il sistema è dotato di due processori ridondanti, in modo che possa continuare a funzionare anche se uno dei due dovesse fallire;
- il sistema ha due terminali di controllo remoto ridondanti;
- il sistema dispone di due bus ridondanti, denominati Bus A e Bus B;
- il sistema di navigazione è basato sull'Inertial Navigation System (INS). In caso di emergenza è possibile sostituirlo con il sistema di navigazione formato dal Doppler e da uno dei tre moduli AHRS presenti nel sistema;

Da queste informazioni possiamo costruire il reliability block diagram del sistema:

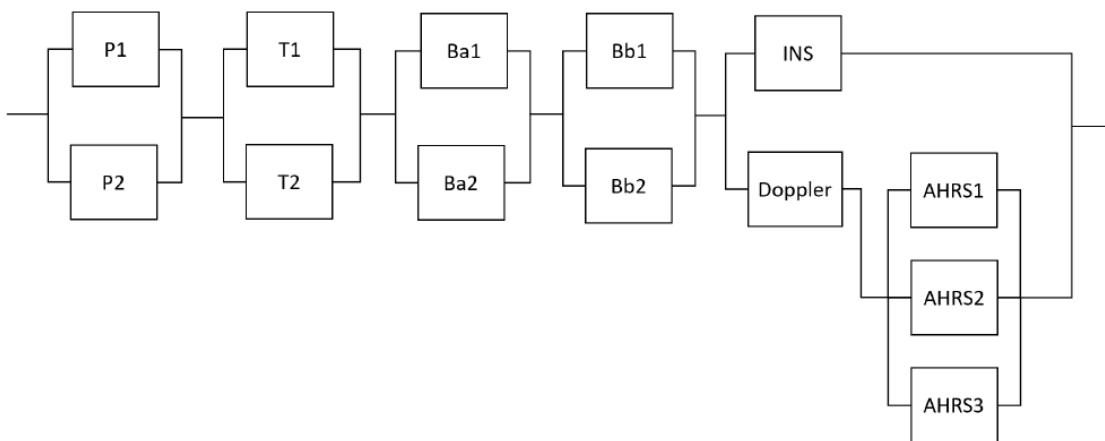


Figura 5.17: Reliability Block Diagram

Dal modello possiamo quindi calcolare la reliability totale del sistema:

$$R_{sys} = \left[1 - (1 - R_P)^2\right] \cdot \left[1 - (1 - R_T)^2\right] \cdot \left[1 - (1 - R_{BusA})^2\right] \cdot \left[1 - (1 - R_{BusB})^2\right] \cdot \left[1 - (1 - R_{INS}) \left(1 - R_D (1 - R_{AHRS})^3\right)\right]$$

Punto b

Andiamo, ora, a modellare il sistema con l'utilizzo dei fault trees. Inoltre andiamo ad individuare i minimal cutsets che provocano il failure del sistema:

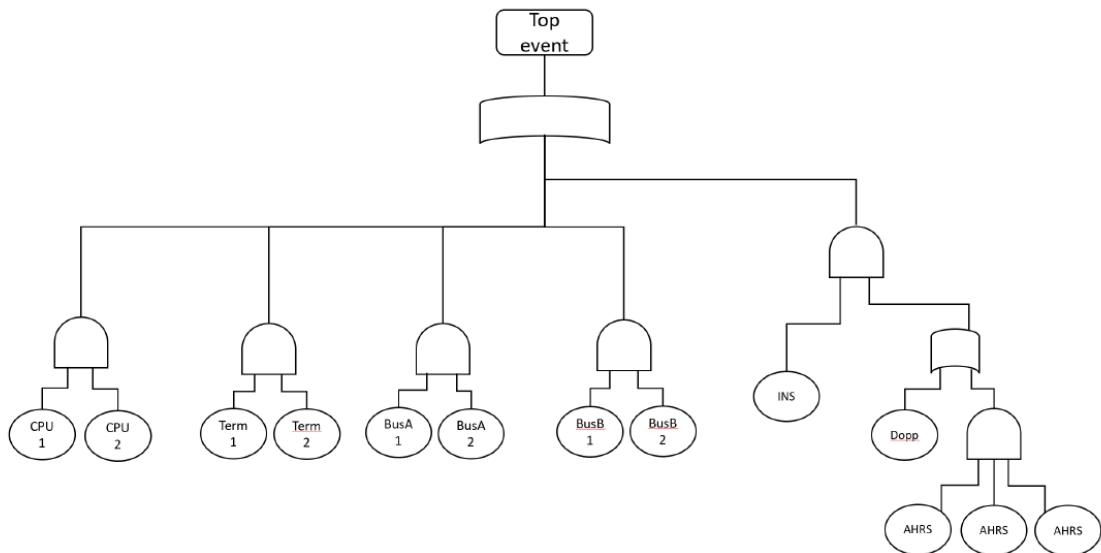


Figura 5.18: Fault tree

I minimal cutsets individuati sono i seguenti:

- CPU 1 - CPU 2;
- Terminal 1 - Terminal 2;
- Bus A1 - Bus A2;
- Bus B1 - Bus B2;
- INS - AHRS 1 - AHRS 2 - AHRS 3;
- INS - Doppler.

Punto c

In questo punto bisogna calcolare la reliability del sistema con mission time pari ad 1h e con MTTF dei singoli componenti uguali a:

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

Supponendo che la reliability di ogni singolo componente segua la legge di fallimento esponenziale avremo che il valore della reliability ad un'ora di volo è pari a:

$$R_{sys}(1h) = 0.99999894132$$

Punto d

Per quest'ultima parte dell'esercizio si richiede di non considerare un valore ideale di coverage per la fault detection bensì di calcolare il valore C tale da garantire una reliability del 99% per un mission time di un'ora.

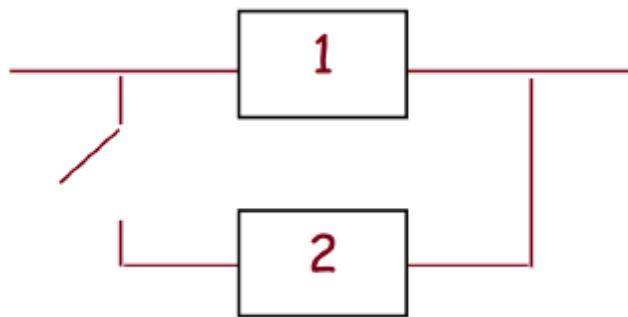


Figura 5.19: Sistema 7

A causa del fattore di coverage posto in serie ad uno dei due processori replicati, l'espressione della reliability cambia come segue:

$$\begin{aligned}
 R_{sys}(t = 1h, C) &= [1 - (1 - R_p) * (1 - R_p * C)] * \left[1 - (1 - R_t)^2\right] * \left[1 - (1 - R_{Ba})^2\right] * \\
 &\quad * \left[1 - (1 - R_{Bb})^2\right] * \left[1 - (1 - R_{INS}) * \left(1 - R_D \left(1 - (1 - R_{AHRS})^3\right)\right)\right] = 0.99
 \end{aligned}$$

A partire da tale espressione, è possibile ricavare il valore di C che risulta essere pari a:

$$C \approx 0.91.$$

Capitolo 6

FFDA

6.1 Traccia

Condurre una FFDA di tipo log-based su due file di log appartenenti ai due seguenti super-calcolatori:

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

6.2 Field Failure Data Analysis (FFDA)

Le tecniche di analisi note come Field Failure Data Analysis, sono tecniche utilizzate per valutare le proprietà di affidabilità di un sistema già sviluppato e funzionante. I dati analizzati sono dati relativi ai fallimenti del sistema e derivano da misurazioni effettuate in situazioni di carico di lavoro reale, poiché, in effetti, non esiste un modo migliore per misurare l'affidabilità di un sistema se non attraverso misurazioni dirette. Per effettuare tale analisi bisogna seguire le seguenti fasi:

- **Data Logging and Collection:** si raccolgono i dati grezzi del funzionamento del sistema in file tipicamente detti di log. In questa fase i problemi sono legati al fatto che abbiamo più sorgenti e bisogna capire cosa monitorare e prendere in base a ciò che ci interessa. Le tecniche comunemente adottate risultano “Failure reports” e “Event logs”;
- **Data Filtering:** partendo dai dati grezzi, viene effettuata una fase di filtraggio. Tale filtraggio può essere di due differenti tipi ovvero *whitelist* (quando vado a filtrare dall'intero dataset tutte le informazioni che mi interessano) oppure *blacklist* (dove vengono filtrate le informazioni che non mi interessano). Solitamente nella pratica viene utilizzata l'analisi *whitelist*. Dunque, il filtraggio viene effettuato per concentrare l'analisi solo sui dati per noi significativi;

- **Data Manipulation:** nei log c'è una fortissima ridondanza degli eventi. Bisogna quindi riconoscere nel log quali sono gli effettivi eventi di fallimento. E' necessario quindi accorpare gli eventi che sono relativi ad un unico fallimento. L'obiettivo di base della fase di Data Manipulation è quello di identificare gli errori e analizzare i log in modo da ridurre tutta la ridondanza. Una delle tecniche principali utilizzate per tale scopo è basata sul tempo e su un principio molto semplice: se gli eventi accadono in un intorno limitato nel tempo allora molto probabilmente tali eventi sono correlati tra loro e sono dovuti allo stesso fallimento, altrimenti gli eventi sono incorrelati e sono dovuti a fallimenti diversi. Dunque, su una base temporale molto semplice riesco a distinguere molto bene la ridondanza dei dati. Tale tecnica prende il nome di coalescenza temporale. Oltre ad essa esistono inoltre anche la coalescenza spaziale e content-based. Inoltre, in questa fase può essere utilizzato anche il clustering per raggruppare dati simili;
- **Data Analysis:** una volta che tutti i dati sono stati 'lavorati' e risultano disponibili, vengono sottoposti ad un'analisi utile per differenti scopi: considerare la gravità dei fallimenti, identificare la posizione del fallimento, trovare la curva di reliability del sistema, identificare la distribuzione del Time to Failure e molto altro in base a ciò che maggiormente ci interessa.

6.3 Mercury

Mercury è un super-calcolatore con architettura three layers dotato di un nodo centrale di management chiamato master che conosce come sono dislocati logicamente i vari nodi e ne governa la struttura. I nodi possono essere di tre tipi, in riferimento ai tre diversi layers: nodi front-end di login, nodi computation e nodi di storage. Dunque, ogni livello presenta una specifica responsabilità ovvero, login utilizzato per gli accessi, computation per eseguire operazioni elaborate e, infine, storage per gestire gli accessi alle memorie. Possiamo apprezzarne di seguito l'architettura:

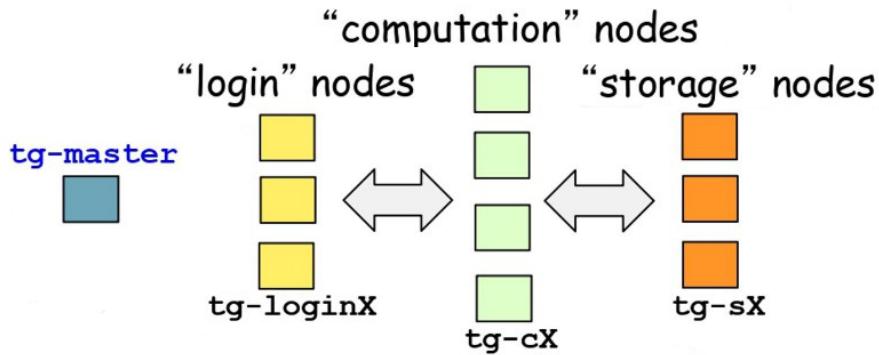


Figura 6.1: Architettura Mercury

Su ogni nodo è presente una distribuzione Linux, inoltre l'intero cluster Mercury è collegato da una rete wormhole a bassa latenza detta Myrinet.

In riferimento a tale sistema, tutti i syslog sono stati collezionati e convogliati in un server dove sono stati filtrati (per la fase di raccolta è stato considerato un tempo di circa un mese, generando un file di qualche decina di GB successivamente filtrato). Il risultato finale è il file di log *MercuryErrorLog.txt* contenente 80854 entry formattate nel seguente modo:

- **timestamp:** valore temporale in formato UNIX;
- **nodo origine:** stringa che identifica la tipologia del nodo in cui si è verificato l'errore;
- **sottosistema di origine:** stringa che identifica il sottosistema che ha originato l'errore. In totale sono presenti 5 tipi di sottosistemi: DEV, MEM, I-O, NET, e PRO;
- **messaggio:** stringa che contiene il messaggio d'errore.

6.3.1 Data Manipulation

La fase di data manipulation del log a nostra disposizione ha l'obiettivo di identificare i fallimenti avvenuti nel sistema. In particolare, il nostro scopo è quello di associare le entry che costituiscono il log ad effettivi eventi di fallimento. Per fare ciò ci serviremo di una cosiddetta **tecnica di coalescenza temporale**. In generale, la coalescenza è una tecnica per il filtraggio e la manipolazione dei dati in grado di ridurre l'insieme delle informazioni raccolte durante la fase di logging collection. Fare ciò risulta essere molto importante al fine di svolgere correttamente le operazioni di analisi di un sistema in quanto un guasto genera in un sistema più fallimenti e dal momento che gli effetti di un guasto si propagano all'interno di tutto il sistema, essi vengono rilevati più volte. Per cui si vanno a raggruppare tutti gli errori registrati relativi al medesimo guasto in un unico evento.

In questo caso, dunque, trattandosi - come detto - di coalescenza temporale, entry differenti vengono raggruppate in base alla distanza che intercorre tra di loro a livello temporale secondo la seguente formula:

*If $t(X_{i+1}) - t(X_I) < W$ then
Add X_{I+1} to the tuple*

dove:

- X_i è l'i-esima entry nel file di log;
- $t(X_i)$ è il timestamp della entry X_i ;
- W è una finestra temporale configurabile.

L'algoritmo, dunque, lavora valutando se la differenza tra due timestamp di due entry consecutive è minore di una determinata soglia W , detta **finestra di coalescenza**. Se ciò avviene, allora le due entry fanno parte della stessa tupla. A questo punto, dunque, risulta chiara l'importanza della scelta della suddetta finestra di coalescenza in quanto il suo valore determina i risultati ottenuti dall'algoritmo stesso perché:

- una finestra di coalescenza troppo grande porterebbe ad incorrere in un eccessivo numero di collisioni → si parla di **collisione** quando due eventi relativi a due guasti diversi vengono inseriti nella stessa tupla (è un'operazione ottimistica e pericolosa perché di due guasti ne consideriamo soltanto uno);
- una finestra di coalescenza troppo piccola porterebbe ad incorrere in un eccessivo numero di troncamenti → si parla di **troncamento** quando due eventi appartenenti allo stesso fenomeno (stesso guasto) sono inseriti in tuple differenti (è un'operazione pessimistica perché è come se considero due fallimenti anche se, in realtà, se ne è verificato soltanto uno).

La scelta del valore della finestra di coalescenza rientra nella cosiddetta analisi di sensitività (**sensitivity analysis**). Tramite un apposito script Matlab, dunque, abbiamo graficato l'andamento del numero di tuple rispetto alla dimensione della finestra di coalescenza ottenendo il seguente risultato:

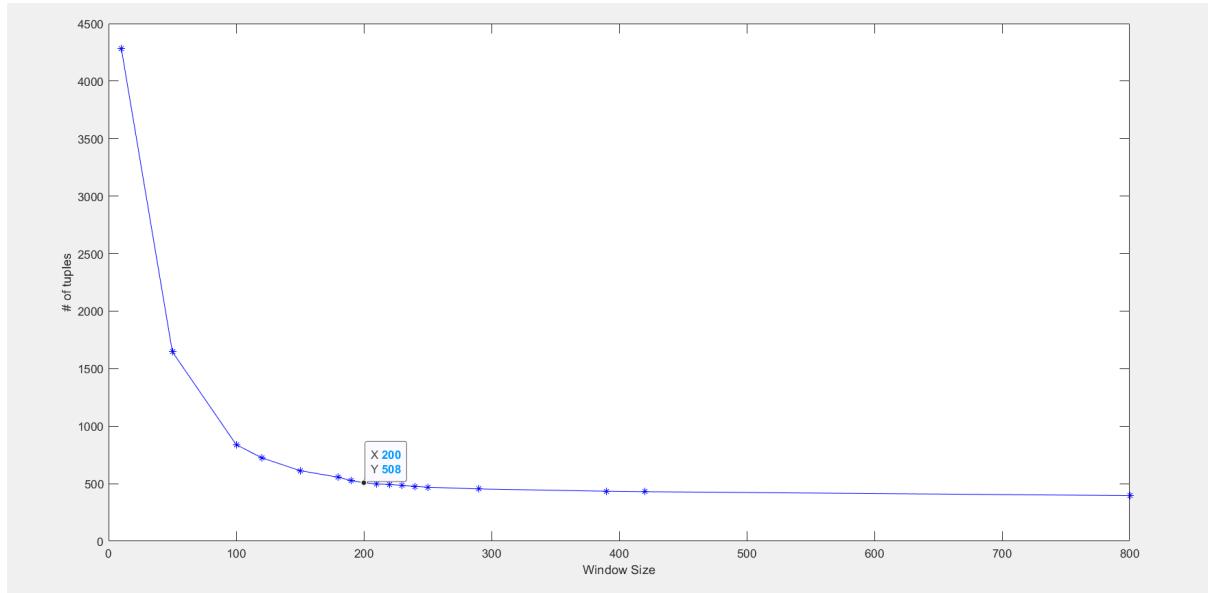


Figura 6.2: Andamento del numero di tuple rispetto alla dimensione della finestra di coalescenza

A questo punto, abbiamo seguito una regola empirica per la scelta del valore della finestra di coalescenza secondo la quale tale valore debba essere quello subito successivo al “gomito” della curva mostrata in Figura 6.2. Il tempo di coalescenza scelto, dunque, è pari a *200 secondi*.

Scelto tale valore della finestra di coalescenza, abbiamo utilizzato un apposito script Shell che ci ha permesso di generare tanti file quanto il numero di tuple ottenute (508) ed ulteriori file utili che elenchiamo brevemente di seguito:

- *lengths.txt* → differenza tra i timestamp della prima e dell’ultima entry della tupla, indica dunque la durata della tupla;
- *interarrivals.txt* → tempo (in secondi) che intercorre tra la prima entry della tupla corrente e l’ultima entry della tupla precedente;
- *startingPoints.txt* → timestamp della prima entry della tupla con relativa durata della stessa.

6.3.2 Data Analysis

Terminata la fase di Data Manipulation, passiamo alla fase di Data Analysis vera e propria. L’obiettivo di tale fase è quello di cercare di estrarre dai dati precedentemente raccolti e manipolati una serie di informazioni relative alla reliability del sistema. In particolare, ricordiamo che:

- la reliability indica la probabilità che il sistema funzioni per un certo tempo t dall'ultimo fallimento, e quindi non fallisca prima di tale tempo t ;
- la unreliability indica la probabilità che il sistema fallisca dopo un certo tempo t dall'ultimo fallimento.

In particolare, coi dati a nostra disposizione è possibile estrarre la distribuzione di probabilità empirica del Time to Failure (TTF) del sistema. Abbiamo, dunque, realizzato un grafico costruendo la CDF del TTF (ossia la unreliability del sistema). Inoltre, abbiamo inserito nel grafico anche la reliability del sistema (servendoci della formula $1 - \text{empirical TTF}$). Il grafico così ottenuto viene riportato di seguito:

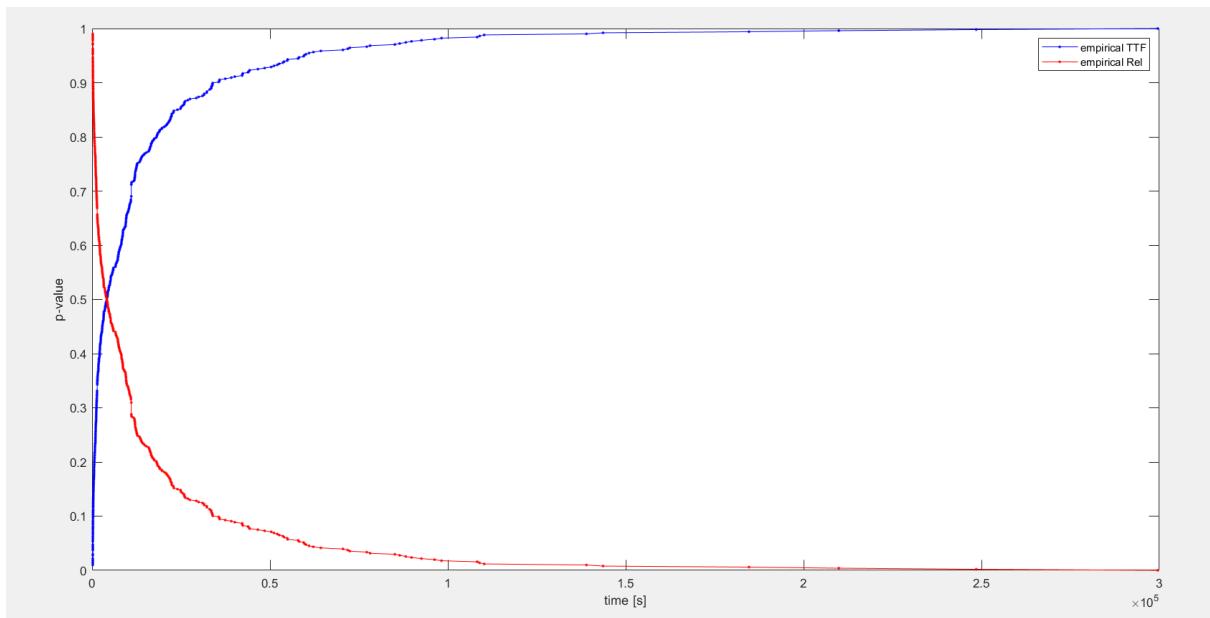


Figura 6.3: Curve di reliability ed unreliability

A questo punto, ottenuta la funzione di reliability empirica, risulta fondamentale confrontare tale distribuzione con una serie di distribuzioni teoriche note in quanto in base alla distribuzione teorica che meglio approssima la nostra distribuzione reale possiamo già dedurre una serie di caratteristiche relative ai fallimenti rappresentati dai dati a nostra disposizione. Tra le distribuzioni teoriche note più importanti ricordiamo:

- distribuzione esponenziale → tipicamente associata a guasti di natura isolata dovuti, ad esempio, ad un unico modulo hardware o software;
- distribuzione iper-esponenziale → tipicamente associata a guasti causati da più moduli distinti ed indipendenti;

- distribuzione di Weibull → tipicamente associata ad errori di accumulazione e fallimenti dovuti a sistemi in esecuzione da molto tempo.

Tramite l'utilizzo dell'apposito tool Matlab *Curve Fitter*, abbiamo effettuato il fitting della funzione di reliability con le 3 distribuzioni precedentemente elencate. Al fine di evidenziare la bontà del fitting di una determinata distribuzione, oltre ad una prima analisi visiva, verranno considerati anche due parametri:

- SSE → per avere una buona approssimazione, la somma quadratica degli errori deve assumere un valore quanto più prossimo a 0;
- R-quadro → indica il legame tra la variabilità dei dati e la correttezza del modello statistico utilizzato e, per questo, deve assumere un valore quanto più prossimo ad 1.

Infine, verrà effettuato anche il test non parametrico di Kolmogorov-Smirnov al fine di avere un'ulteriore conferma della corretta (o non corretta) approssimazione del modello teorico scelto con la nostra distribuzione reale.

Riportiamo, di seguito, i risultati ottenuti.

Fitting esponenziale

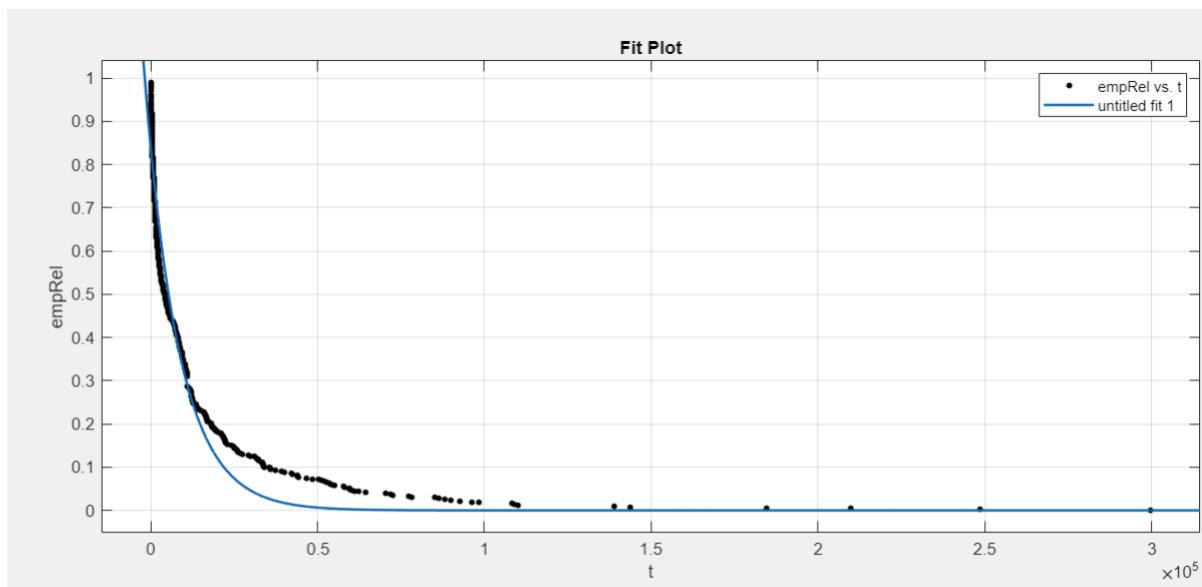


Figura 6.4: Fitting esponenziale

```

General model Exp1:
f(x) = a*exp(b*x)
Coefficients (with 95% confidence bounds):
a = 0.8386 (0.8283, 0.8489)
b = -9.702e-05 (-0.0001007, -9.338e-05)

Goodness of fit:
SSE: 1.747
R-square: 0.9534
Adjusted R-square: 0.9533
RMSE: 0.06136

```

Figura 6.5: Risultati

In questo caso, fin da una prima analisi visiva risulta chiaramente comprensibile che la distribuzione esponenziale non si adatta in maniera soddisfacente alla distribuzione dei dati a nostra disposizione. Tale risultato viene confermato non solo da un valore di SSE troppo alto ma anche dal test non parametrico di Kolmogorov-Smirnov che ha evidenziato un p-value pari a $5.2 \cdot 10^{-5}$. Un valore del p-value così basso, dunque, rigetta l'ipotesi nulla permettendoci di affermare che i campioni **non appartengono** ad una popolazione distribuita esponenzialmente.

Fitting iper-esponenziale

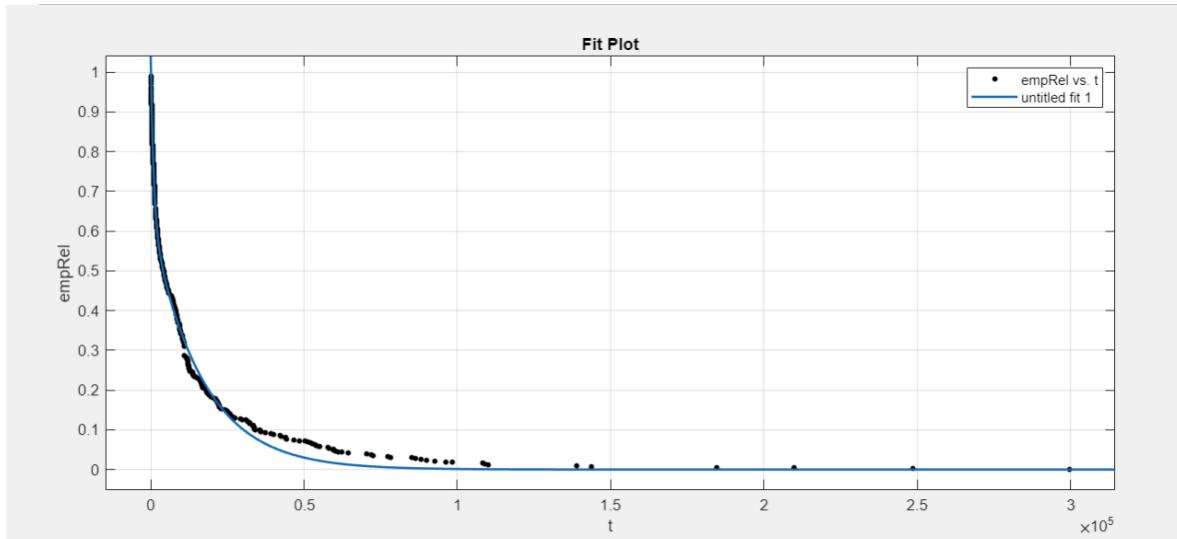


Figura 6.6: Fitting iper-esponenziale

```

General model Exp2:
f(x) = a*exp(b*x) + c*exp(d*x)
Coefficients (with 95% confidence bounds):
a = 0.3922 (0.3824, 0.402)
b = -0.001036 (-0.001099, -0.0009734)
c = 0.6297 (0.6207, 0.6388)
d = -6.077e-05 (-6.226e-05, -5.929e-05)

Goodness of fit:
SSE: 0.1457
R-square: 0.9961
Adjusted R-square: 0.9961
RMSE: 0.01776

```

Figura 6.7: Risultati

Nel caso di funzione iper-esponenziale, notiamo subito visivamente che l'approssimazione risulta essere migliore. Tale risultato viene confermato anche da un valore di R-quadro molto prossimo ad 1 ed un valore di SSE che si avvicina allo 0 molto di più rispetto al caso esponenziale. Inoltre, il test di Kolmogorov-Smirnov ha restituito un p-value pari a 0.816 e, dunque, molto elevato.

Fitting Weibull

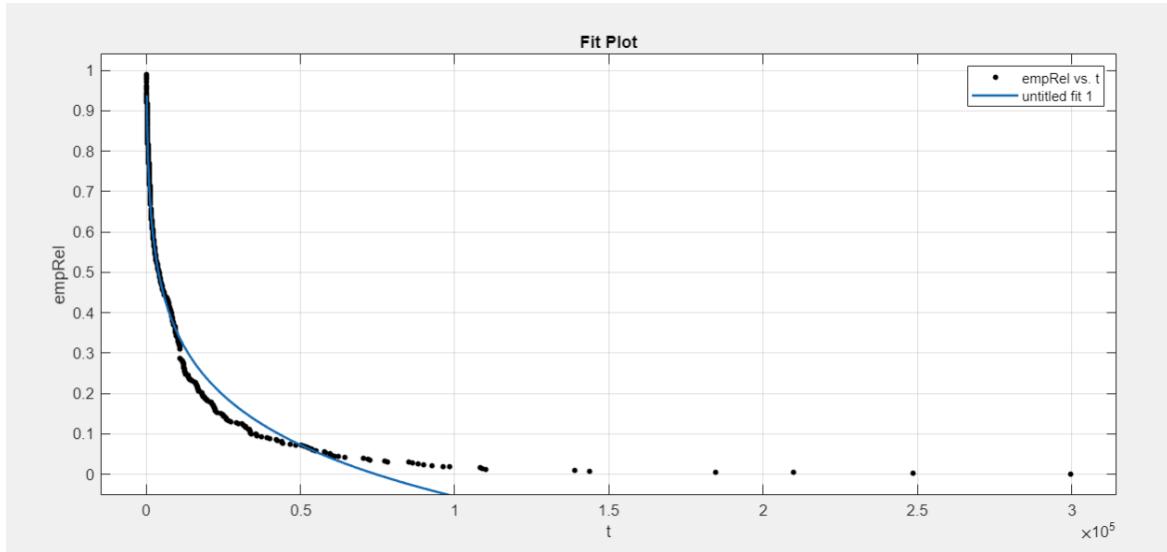


Figura 6.8: Fitting Weibull

```

General model:
f(x) = exp(-a.*x.^b)
Coefficients (with 95% confidence bounds):
a = 5.193e+05 (-4.926e+06, 5.965e+06)
b = 0.05193 (0.03862, 0.06524)
exp = 3.546 (2.76, 4.333)

Goodness of fit:
SSE: 0.6317
R-square: 0.9832
Adjusted R-square: 0.9831
RMSE: 0.03694

```

Figura 6.9: Risultati

Infine, anche nel caso di distribuzione di Weibull otteniamo valori di R-quadro piuttosto alti seppur inferiori a quelli della distribuzione iper-esponenziale. Anche il test di Kolmogorov-Smirnov riporta un p-value inferiore pari a 0.7130.

A valle delle considerazioni effettuate riguardo il fitting delle 3 distribuzioni considerate, possiamo concludere che la migliore tra tali distribuzioni risulta essere quella **iper-esponenziale**. Questo ci permette di effettuare una serie di considerazioni tra le quali troviamo il fatto che una distribuzione di questo tipo evidenzia solitamente la presenza di **errori riconducibili ad uno o più elementi hardware**. Tale ipotesi viene confermata anche dal fatto che la maggior parte delle entry presenti nel file di log appartiene alla categoria “DEV” (come vedremo nell’apposito paragrafo relativo all’analisi dei bottleneck del sistema). Errori appartenenti a tale categoria sono tipicamente legati alle schede PCI ed alle periferiche hardware in generale.

6.4 Blue Gene/L

Blue Gene/L (BGL) è un super-calcolatore sviluppato da *IBM* costituito da 64 *Rack (cabinet)* ognuno dei quali è costituito da due *Midplane* che a loro volta contengono al loro interno 16 *node board* (nodi) ciascuno, per un totale di 32 nodi per ogni rack. Tali nodi sono costituiti da 16 *Compute card* ciascuno e da due *I/O card* addizionali non sempre presenti (soltanto i nodi *N0, N4, N8* ed *NC* contengono I/O card). Ogni *Compute card* presenta una serie di processori (tipicamente 4 processori, per un totale di circa 32000 processori). Vediamo di seguito una figura rappresentante l’architettura del sistema:

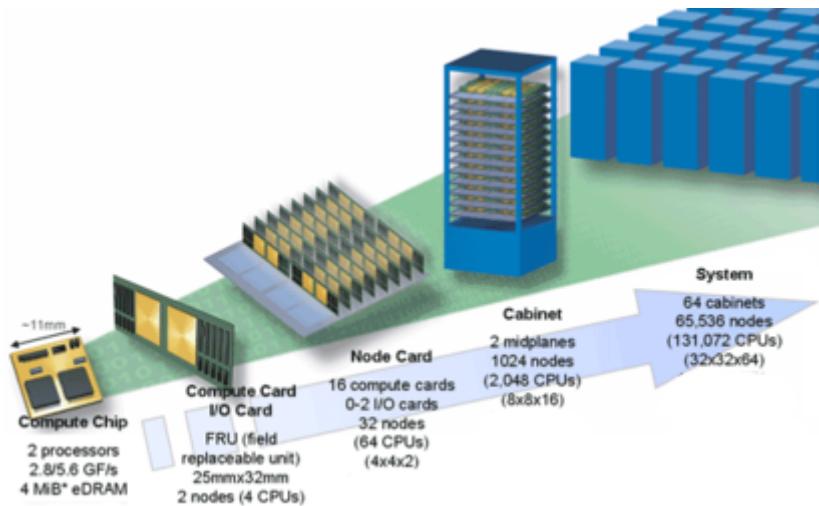


Figura 6.10: Architettura Blue Gene/L

Blue Gene/L presenta un indirizzamento unico per i vari processori, ad esempio con l'indirizzo *R27-M0-NA-C:J11-U11* abbiamo *R27* per il Rack 27, *M0* indica il Midplane 0, *N* il nodo di computazione e *J11* e *U11* sono le unità ed indicano il tipo di card.

Dopo aver presentato il sistema, andiamo a considerare i dati da analizzare. Tali dati sono contenuti in un file di log chiamato *BGLErrorLog.txt* e contiene un totale di 125624 entry. Anche in questo caso, come per Mercury, il formato risulta sostanzialmente lo stesso del Syslog:

- **timestamp** → valore temporale 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(R), midplane(M) e nodo(N) 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 dividere le card in due grandi categorie, card di I/O (J18-Uxx) e card di computation;
- **messaggio testuale** → stringa che contiene il messaggio d'errore.

6.4.1 Data Manipulation

Come fatto nel caso del sistema Mercury, anche la fase di data manipulation per quanto riguarda il sistema Blue Gene consta dell'utilizzo della tecnica di coalescenza temporale.

Tramite un apposito script Matlab, dunque, abbiamo graficato l'andamento del numero di tuple rispetto alla dimensione della finestra di coalescenza ottenendo il seguente risultato:

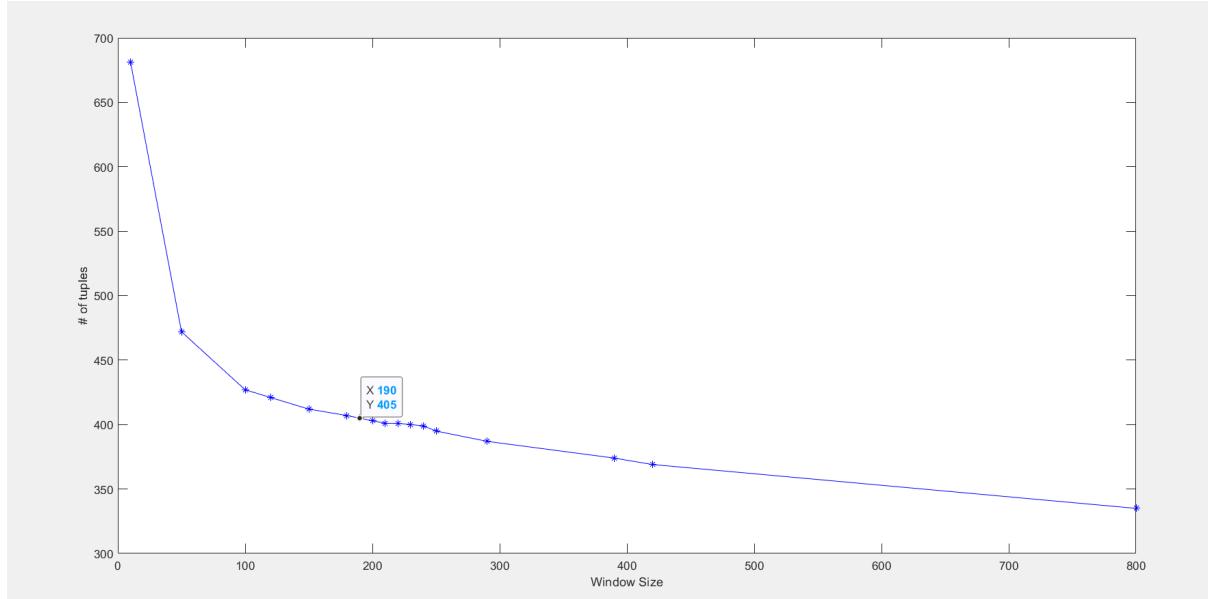


Figura 6.11: Andamento del numero di tuple rispetto alla dimensione della finestra di coalescenza

Seguendo la regola empirica utilizzata anche nel caso del sistema Mercury secondo la quale il valore ottimale della finestra di coalescenza va preso subito dopo il “gomito” della curva riportata in Figura 6.11, abbiamo scelto il valore di *190 secondi*.

Scelto tale valore della finestra di coalescenza, abbiamo utilizzato un apposito script Shell che ci ha permesso di generare tanti file quanto il numero di tuple ottenute (405), oltre ai file *lengths.txt*, *interarrivals.txt*, *startingPoints.txt* spiegati in precedenza.

6.4.2 Data Analysis

Terminata la fase di data manipulation, passiamo alla fase di data analysis vera e propria. L'obiettivo di tale fase è quello di cercare di estrarre dai dati precedentemente raccolti e manipolati una serie di informazioni relative alla reliability del sistema. In particolare, coi dati a nostra disposizione è possibile estrarre la distribuzione di probabilità empirica del Time to Failure (TTF) del sistema. Abbiamo, dunque, realizzato un grafico costruendo la CDF del TTF (ossia la unreliability del sistema). Inoltre, abbiamo inserito nel grafico anche la reliability del sistema (servendoci della formula $1 - \text{empiricalTTF}$). Il grafico così ottenuto viene riportato di seguito:

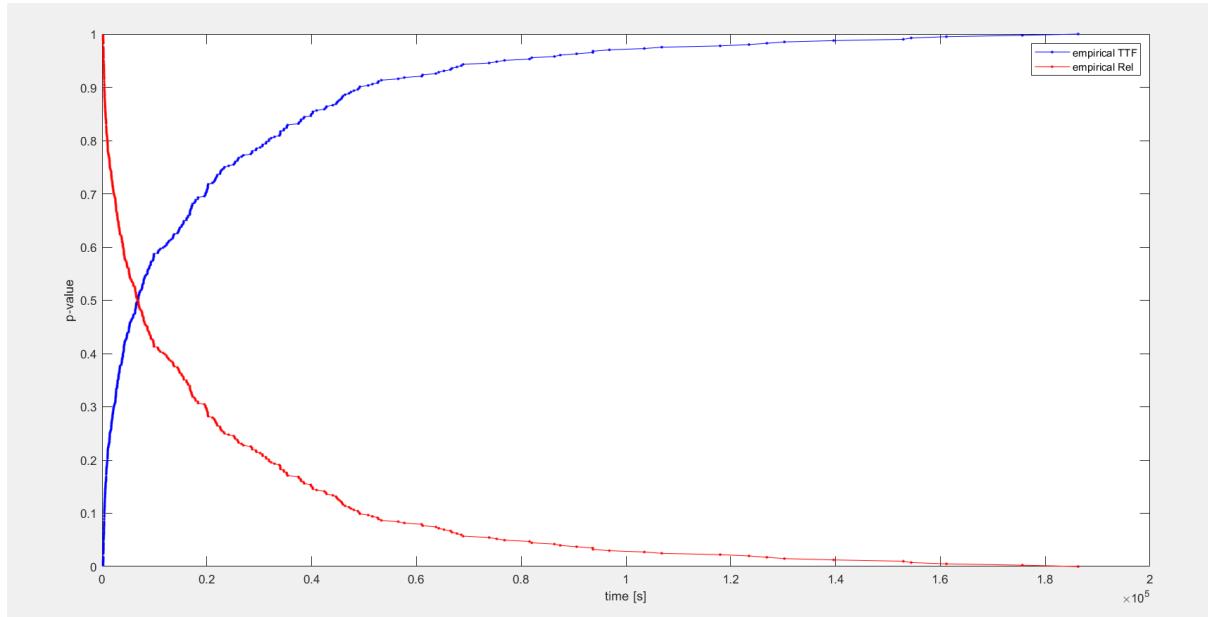


Figura 6.12: Curve di reliability ed unreliability

A questo punto, ottenuta la funzione di reliability empirica, risulta fondamentale confrontare tale distribuzione con una serie di distribuzioni teoriche note in quanto in base alla distribuzione teorica che meglio approssima la nostra distribuzione reale possiamo già dedurre una serie di caratteristiche relative ai fallimenti rappresentati dai dati a nostra disposizione. Tramite l'utilizzo dell'apposito tool Matlab *Curve Fitter*, abbiamo effettuato il fitting della funzione di reliability con le 3 distribuzioni precedentemente elencate. Riportiamo, di seguito, i risultati ottenuti.

Fitting esponenziale

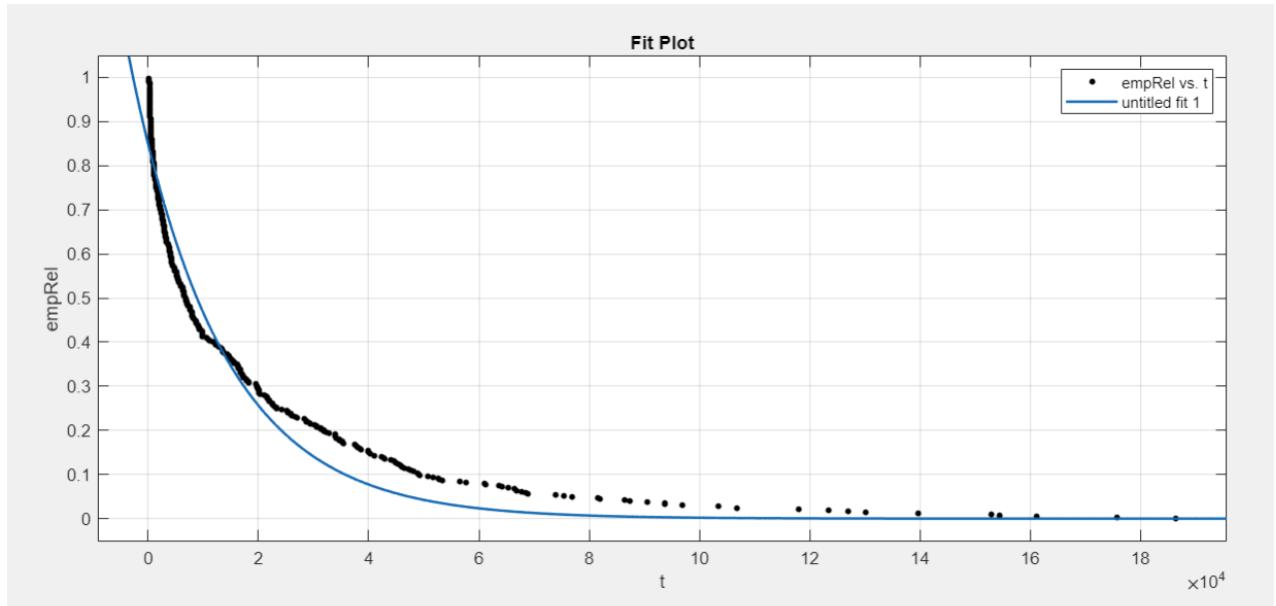


Figura 6.13: Fitting esponenziale

```

General model Exp1:
f(x) = a*exp(b*x)
Coefficients (with 95% confidence bounds):
  a =  0.8494 (0.8383, 0.8605)
  b = -5.986e-05 (-6.221e-05, -5.751e-05)

Goodness of fit:
SSE: 1.436
R-square: 0.9555
Adjusted R-square: 0.9554
RMSE: 0.06038

```

Figura 6.14: Risultati

In questo caso, fin da una prima analisi visiva risulta chiaramente comprensibile che la distribuzione esponenziale non si adatta in maniera soddisfacente alla distribuzione dei dati a nostra disposizione. Tale risultato viene confermato non solo da un valore di SSE troppo alto ma anche dal test non parametrico di Kolmogorov-Smirnov che ha evidenziato un p-value pari a $3.5 \cdot 10^{-4}$. Un valore del p-value così basso, dunque, rigetta l'ipotesi nulla permettendoci di affermare che i campioni **non appartengono** ad una popolazione distribuita esponenzialmente.

Fitting iper-esponenziale

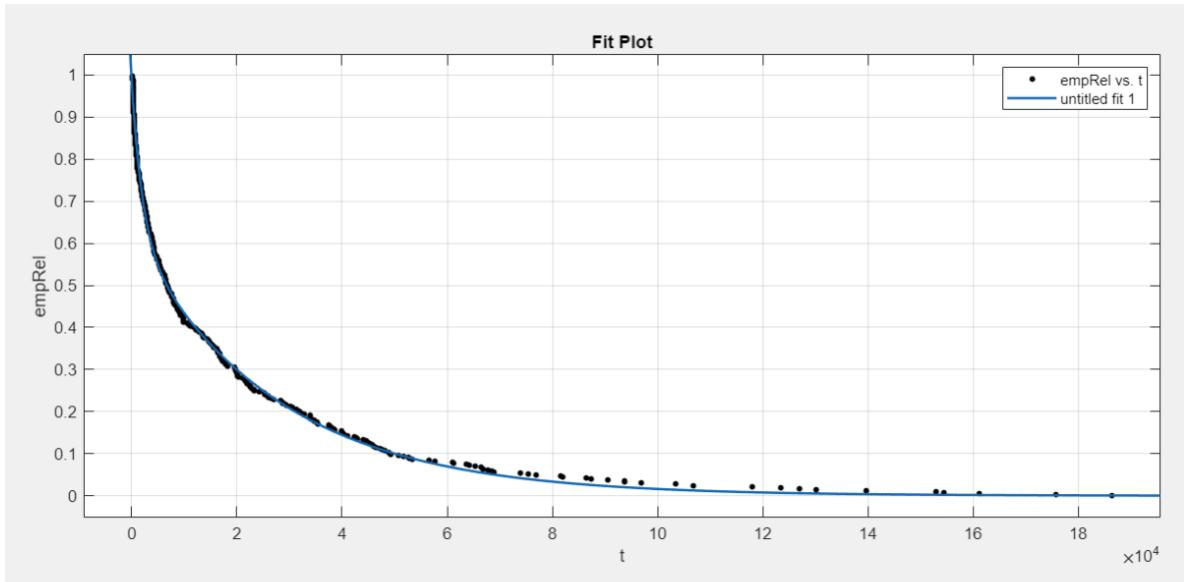


Figura 6.15: Fitting iper-esponenziale

```
General model Exp2:
f(x) = a*exp(b*x) + c*exp(d*x)
Coefficients (with 95% confidence bounds):
a = 0.367 (0.3589, 0.3751)
b = -0.0004956 (-0.0005215, -0.0004698)
c = 0.6215 (0.6132, 0.6299)
d = -3.65e-05 (-3.726e-05, -3.574e-05)

Goodness of fit:
SSE: 0.07311
R-square: 0.9977
Adjusted R-square: 0.9977
RMSE: 0.01366
```

Figura 6.16: Risultati

Nel caso di funzione iper-esponenziale, notiamo subito visivamente che l'approssimazione risulta essere migliore. Tale risultato viene confermato anche da un valore di R-quadro molto prossimo ad 1 ed un valore di SSE che si avvicina allo 0 molto di più rispetto al caso esponenziale. Inoltre, il test di Kolmogorov-Smirnov ha restituito un p-value pari a 0.8971 e, dunque, molto elevato.

Fitting Weibull

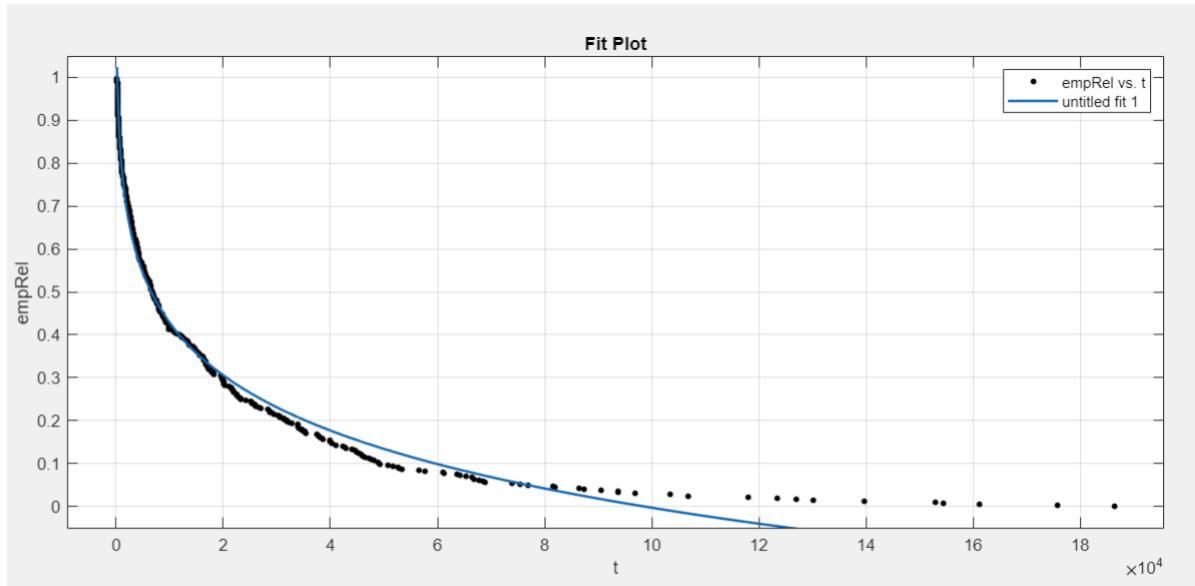


Figura 6.17: Fitting Weibull

General model:
 $f(x) = \exp(-a \cdot x)^b$
 Coefficients (with 95% confidence bounds):
 $a = 46.74$ (-122.1, 215.5)
 $b = 0.0696$ (0.06028, 0.07892)
 $\exp = 2.909$ (2.589, 3.229)

Goodness of fit:
 SSE: 0.2179
 R-square: 0.9932
 Adjusted R-square: 0.9932
 RMSE: 0.02355

Figura 6.18: Risultati

Infine, anche nel caso di distribuzione di Weibull otteniamo valori di R-quadro piuttosto alti seppur inferiori a quelli della distribuzione iper-esponenziale. Il test di Kolmogorov-Smirnov riporta un p-value molto alto pari a 0.9926.

A valle delle considerazioni effettuate riguardo il fitting delle 3 distribuzioni considerate, possiamo concludere che la migliore tra tali distribuzioni risulta essere quella **Weibull**. Tale scelta è dovuta al valore di p-value molto alto che si ottiene. Questo ci permette di effettuare una serie di considerazioni tra le quali troviamo il fatto che una distribuzione di questo tipo evidenzia solitamente la presenza di **errori riconducibili a fenomeni di accumulazione e/o saturazione**.

6.5 Confronto dell'affidabilità di Mercury e Blue Gene

A valle delle considerazioni fatte sulle reliability dei due super-calcolatori presi in esame, riportiamo di seguito un grafico che confronta proprio l'affidabilità di Mercury e Blue Gene.

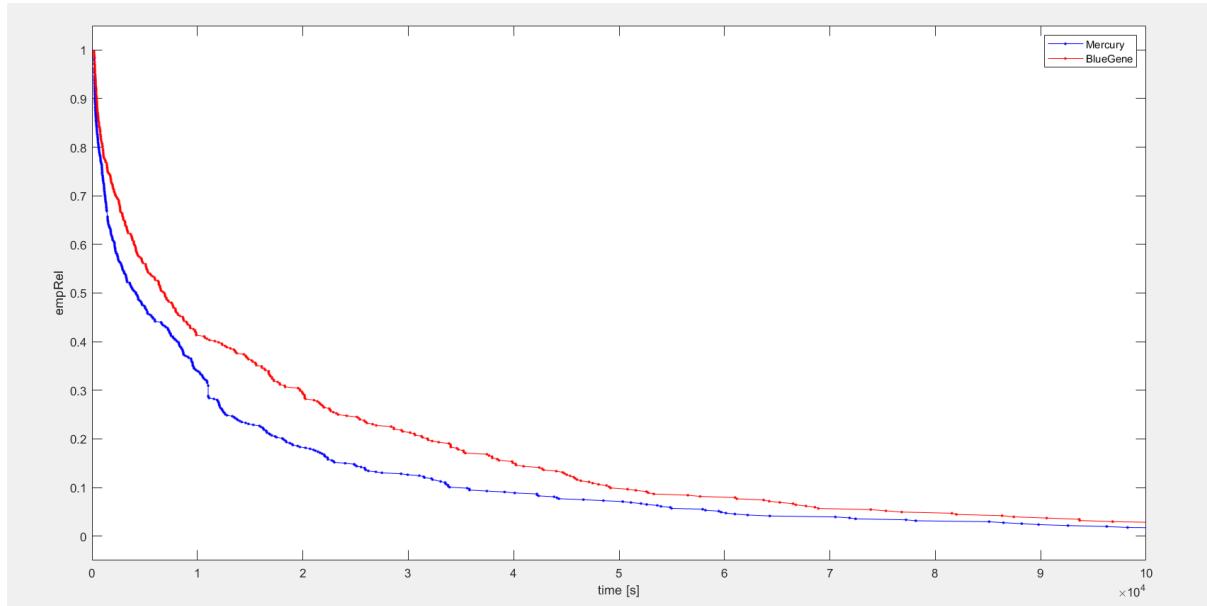


Figura 6.19: Confronto affidabilità tra Mercury e Blue Gene

Già da una prima analisi visiva, è possibile notare che il sistema Blue Gene risulta essere maggiormente affidabile rispetto al sistema Mercury per un qualsiasi mission time. Tale risultato viene confermato anche dal calcolo del Mean Time To Failure (MTTF) dei due sistemi:

- $MTTF_{Mercury} \simeq 1.4 * 10^4 s$
- $MTTF_{BlueGene} \simeq 1.9 * 10^4 s$

6.6 Ulteriori analisi: i bottleneck di Mercury e Blue Gene

Al fine di verificare l'eventuale presenza di bottleneck all'interno dei sistemi, abbiamo deciso di analizzare in maniera più approfondita le tuple ottenute per ogni sistema in modo da evidenziare nodi particolarmente avvezzi a generare fallimenti.

Mercury

Per quanto riguarda il sistema Mercury, abbiamo innanzitutto esplicitato il numero di errori presenti in ogni tupla:

Row ID	I ▼ count
tuple299	9825
tuple298	8590
tuple324	5883
tuple325	5065
tuple114	4049
tuple312	3884
tuple326	3772
tuple305	3583
tuple311	3536
tuple459	2938
tuple308	2412
tuple315	2057
tuple332	1550
tuple317	1148
tuple302	1137
tuple323	1112
tuple327	1083
tuple331	993
tuple318	927
tuple402	875
tuple306	763
tuple304	746
tuple370	712
tuple334	701
tuple15	698
tuple10	680
tuple314	650
tuple319	628
tuple507	562
tuple300	503
tuple414	457
tuple307	389
tuple322	344
tuple313	334
tuple69	330
tuple76	299
tuple502	256
tuple409	238

Figura 6.20: Numero di errori in ogni tupla

Notiamo subito che la maggior parte degli errori sono contenuti tra le tuple 299 e 332. Analizzando maggiormente nel dettaglio tali tuple, è possibile notare che il maggior numero di errori viene causato da un particolare nodo, il *tg-c401*:

Row ID	count
tg-c401	62340
tg-maste	4098
tg-c572	4030
tg-s044	3224
tg-c238	1273
tg-c242	1067
tg-login	823
tg-c648	643
tg-c117	268
tg-c669	267
tg-c550	256
tg-c324	240
tg-s038	230
tg-c894	218
tg-s176	209
tg-c284	180
tg-c451	159
tg-c447	149
tg-c407	97
tg-c781	97
tg-c415	92
tg-c735	84
tg-c685	80
tg-c860	63
tg-c733	62
tg-c850	56
tg-c533	52
tg-c757	50
tg-c416	37
tg-c880	35
tg-c846	33
tg-c891-	32
tg-c027	30
tg-c196	30
tg-s159	28
tg-c399	25
tg-c696	25
tg-c834	24

Figura 6.21: Numero di errori per nodo

Tale nodo, dunque, può essere considerato come vero e proprio **bottleneck del sistema**.

Infine, come anticipato anche in precedenza, è possibile analizzare anche le categorie di appartenenza degli errori più comuni:

Row ID	count
DEV	57248
MEM	12819
I-O	5547
NET	3702
PRO	1504
OTH	34

Figura 6.22: Numero di errori per categoria

Come è possibile notare, la maggior parte degli errori appartiene alla categoria “DEV” e “MEM” e, dunque, sono riconducibili a problemi relativi all’hardware del sistema.

Blue Gene

Anche per il sistema Blue Gene, abbiamo innanzitutto esplicitato il numero di errori presenti in ogni tupla:

Row ID	count
tuple343	12152
tuple184	4096
tuple282	3071
tuple116	2048
tuple142	2048
tuple27	2048
tuple352	2048
tuple43	2048
tuple79	2048
tuple89	2048
tuple98	2048
tuple104	1728
tuple230	1174
tuple266	1128
tuple353	1117
tuple276	1062
tuple19	1035
tuple115	1025
tuple106	1024
tuple107	1024
tuple119	1024
tuple151	1024
tuple152	1024
tuple157	1024
tuple161	1024
tuple174	1024
tuple177	1024
tuple190	1024
tuple2	1024
tuple210	1024
tuple212	1024
tuple222	1024
tuple23	1024
tuple24	1024
tuple26	1024
tuple333	1024
tuple340	1024
tuple341	1024

Figura 6.23: Numero di errori in ogni tupla

Differentemente da quanto constatato per il sistema Mercury, notiamo che per quanto riguarda il sistema Blue Gene la distribuzione delle tuple risulta essere molto più omogenea. In questo caso, dunque, non è possibile rilevare un intervallo temporale interessante da analizzare.

Anche l'analisi dei nodi non evidenzia particolari criticità dovute ad un singolo nodo come è possibile notare dalla seguente tabella:

Row ID	I ▼ count
R71-M0-N4	1716
R12-M0-N0	1563
R63-M0-N2	976
R03-M1-NF	960
R63-M0-N0	791
R36-M1-N0	788
R62-M0-N4	515
R63-M0-NC	460
R63-M0-N8	454
R63-M0-N4	452
R30-M0-N1	394
R55-M1-N3	394
R03-M0-N7	390
R21-M1-N7	390
R60-M1-NC	381
R72-M1-NC	355
R01-M1-N8	326
R10-M0-N8	321
R46-M1-NC	320
R67-M1-NC	318
R46-M1-N0	313
R46-M1-N4	311
R46-M1-N8	305
R62-M1-NA	304
R02-M0-N4	299
R57-M0-NC	292
R57-M1-N0	268
R64-M0-N4	265
R64-M0-N8	262
R62-M0-NC	261
R65-M0-N4	261
R65-M1-N0	261
R61-M0-NC	260
R64-M0-NC	260
R65-M0-N8	260
R60-M0-N0	259
R61-M0-N8	259
R61-M1-NC	259

Figura 6.24: Numero di errori per nodo

Un'ultima analisi che è possibile effettuare al fine di comprendere se qualche sottoparte del sistema possa essere un potenziale bottleneck è quella di separare gli errori dovuti alle card di I/O (etichettate come schede J18) da quelli dovuti alle card di tipo computazionale.

Row ID	I count
J18-U11	50055
J18-U01	49932
J14-U01	2257
J12-U01	1877
J07-U01	1780
J10-U11	1333
J03-U11	1020
J16-U11	973
J06-U11	960
J11-U11	888
J17-U11	824
J09-U01	808
J16-U01	753
J09-U11	746
J08-U01	741
J03-U01	701
J11-U01	700
J05-U11	670
J04-U01	655
J13-U01	647
J15-U01	633
J17-U01	602
J12-U11	596
J04-U11	593
J13-U11	563
J08-U11	560
J10-U01	554
J02-U01	524
J05-U01	456
J15-U11	454
J02-U11	449
J07-U11	445
J14-U11	445
J06-U01	430

Figura 6.25: Numero di errori per categoria

La tabella mostrata sopra evidenzia che la maggior parte delle entry del file di log appartengono ad errori relativi all'I/O. Considerando l'omogeneità presente nelle tuple che non permette di selezionare un particolare nodo "colpevole" di tali fallimenti, risulta ragionevole supporre che **un possibile bottleneck del sistema Blue Gene sia il suo intero sistema di I/O.**