UNIVERSITÀ DEGLI STUDI DI NAPOLI FEDERICO II

Corso di Laurea Magistrale in Ingegneria Informatica

Dipartimento di Ingegneria Elettrica e delle tecnologie dell'informazione

Elaborato finale in

IMPIANTI DI ELABORAZIONE

Professore:

Domenico Cotroneo

Candidati:

Benfenati Domenico M63001165

Carandente Vincenzo M63001229

ANNO ACCADEMICO 2021/2022

Primo Semestre

Indice

1 Benchmark **4**

1.1 Traccia 4

1.2 Descrizione del problema 4

1.3 Raccolta dei dati 5

1.4 Analisi dei sistemi 8

1.4.1 Primo sistema 8

1.4.2 Secondo sistema 8

1.5 Confronto sui dati ottenuti 9

1.6 Approfondimento sistemi senza Virtualizzazione 10

2 Workload Characterization **18**

2.1 Traccia 18

2.2 Workload Characterization 18

2.2.1 Analisi preliminare 18

2.2.2 Preprocessing dei dati 20

2.2.3 Ulteriori considerazioni 22

2.2.4 Principal Component Analysis (PCA) 25

2.2.5 Clustering 27

2.2.6 Calcolo delle devianze 29

2.3 Analisi delle devianze 30

2.3.1 Devianza persa vs Numero di componenti principali 31

2.3.2 Devianza persa vs Dimensione del Workload 32

2.4 Caratterizzazione del workload sintetico 33

3 Web Server **35**

3.1 Traccia 35

3.2 Descrizione del sistema 35

3.3 Capacity Test 36

3.3.1 Approfondimento 38

3.4 Workload Characterization 42
3.4.1 Workload di alto livello 42
3.4.2 Workload di basso livello 44
3.4.3 Caratterizzazione dei workload 45
3.4.4 Applicazione del workload sintetico di alto livello 50
3.4.5 Validazione del workload sintetico di alto livello 54
3.5 Design Of Experiment 55
3.5.1 Analisi dell'importanza 57
3.5.2 Analisi della significatività 58
3.5.3 ANOVA 60

4 RBD **61**
4.1 Esercizio 1 61
4.1.1 Traccia 61
4.1.2 Svolgimento 61
4.2 Esercizio 2 66
4.2.1 Traccia 66
4.2.2 Svolgimento 66
4.3 Esercizio 3 69
4.3.1 Traccia 69
4.3.2 Svolgimento 69
4.4 Esercizio 4 72
4.4.1 Traccia 72
4.4.2 Svolgimento 72
4.5 Esercizio 5 76
4.5.1 Traccia 76
4.5.2 Svolgimento 77

5 Field Failure Data Analysis **81**
5.1 Traccia 81
5.2 Field Feilure Data Analysis 81
5.3 Mercury 82
5.3.1 Descrizione del sistema 82
5.3.2 Fase di Data Manipulation del sistema 83
5.3.3 Analisi Troncamenti e Collisioni 87
5.3.4 Fase di Data Analysis del sistema 89
5.3.5 Fase di Data Manipulation dei componenti 90
5.3.6 Fase di Data Analysis dei componenti 92

5.3.7 Fase di Data Manipulation dei nodi 94
5.3.8 Fase di Data Analysis dei nodi 96
5.3.9 Domande 100
5.4 Blue Gene/L 103
5.4.1 Descrizione del sistema 103
5.4.2 Fase di Data Manipulation del sistema 103
5.4.3 Analisi Troncamenti e collisioni 105
5.4.4 Fase di Data Analysis del sistema 106
5.4.5 Fase di Data Manipulation dei nodi 107
5.4.6 Fase di Data Analysis dei rack 111
5.4.7 Fase di Data Manipulation delle card 113
5.4.8 Fase di Data Analysis delle card 115
5.4.9 Domande 116
5.5 Confronto tra i due sistemi 120

Capitolo 1

Benchmark

1.1 Traccia

Confrontare due sistemi differenti utilizzando il benchmark *N-Body*.

1.2 Descrizione del problema

Per tale analisi si è scelto di utilizzare la simulazione NBody. Tale simulazione considera N corpi liberi di muoversi sotto l'azione della forza di attrazione gravitazionale reciproca. La meccanica classica permette di impostare un sistema di *3N* equazioni differenziali ordinarie accoppiate di second'ordine che si possono ridurre a *3(N-1)* nel sistema del centro di massa. Fissate le condizioni iniziali di ogni corpo, la sua posizione e la sua velocità nello specifico, resta definito un problema di Cauchy che può essere trattato matematicamente. Il problema matematico definito dal benchmark è utile in tale analisi perchè le operazioni eseguite mettono sotto stress non solo le FLops (Floating Point operations), ma anche lo stack e il numero di core del SUT (System Under Test). L'obiettivo dell'analisi è la valutazione delle prestazioni di un certo componente del sistema, detto CUT (Component Under Test), quando esso è sottoposto ad un certo workload, al fine di comparare differenti sistemi. La valutazione che si vuole effettuare è relativa alle prestazioni del processore nell'eseguire operazioni floating point. In particolare il benchmark restituisce un solo parametro di interesse, *Time(ms)* che corrisponde al tempo in microsecondi per costruire la simulazione. Nel caso specifico, il benchmark è stato utilizzato per comparare due sistemi dotati di due processori differenti:

- il primo sistema possiede un processore *AMD Ryzen 5 4600H*, processore a 7 nm caratterizzato da 6 core e 12 threads, con fraquenza massima pari a 4 GHz, e con installati 16 GB di memoria RAM;

- il secondo sistema possiede un processore *Intel i5 8250U*, processore a 14 nm caratterizzato da 4 core e 8 threads, con frequenza massima pari a 3.4 GHz, e con installati 8 GB di memoria RAM.

Altra particolarità da segnalare al fine dell'analisi è che su entrambi i sistemi è stata installato una macchina virtuale Ubuntu tramite *Oracle VirtualBox*, per permettere l'esecuzione dei comandi necessari alla raccolta dei dati sui tempi di esecuzione del benchmark. Su entrambi i sistemi la macchina virtuale è stata dotata di 2 core e di 6 GB di RAM.

1.3 Raccolta dei dati

Per ogni dimensione fissata di corpi sono state raccolte 30 osservazioni su entrambi i sistemi, eseguendo un singolo test e riavviando ogni volta il sistema, ripetendo la raccolta 5 volte e mediando i risultati ottenuti. Tale procedura assicura che le osservazioni siano tra loro **indipendenti e identicamente distribuite** (*i.i.d.*). In questo modo sono verificate le ipotesi del **teorema del limite centrale**, il quale garantisce che al tendere ad infinito della dimensione di un campione, la media campionaria tende ad una distribuzione normale. Di conseguenza, essendo valida l'ipotesi di normalità è possibile applicare test parametrici. Le osservazioni raccolte per ogni dimensione costituiscono il **precampione**, utilizzato per stimare la deviazione standard della popolazione. L'operazione di raccolta dei precampioni ha prodotto i seguenti valori:

AMD Ryzen 5 4600H				Intel Core i5 8250U			
1000	20000	100000	500000	1000	20000	100000	500000
493	7208	39963	181208	758	3469	119543	268820
483	7114	34210	196502	788	3458	93507	264235
493	7538	36389	179519	779	3649	112664	255409
481	7273	34333	172106	749	3344	114435	251097
494	7682	35532	172614	758	3533	92529	236804
494	7676	35791	195477	770	3493	115370	251143
503	7391	37567	178248	781	3473	156133	239911
488	7078	36997	190721	790	3513	120793	251980
499	7577	35919	183907	795	3438	128018	247292
493	7677	36204	184946	783	3581	133482	245900
480	7358	39305	179115	771	3458	122407	247896
494	7056	35751	185358	787	3452	121248	247625
504	7481	38529	178028	750	3522	114541	247406
509	7290	34678	181707	757	3444	126028	250016
479	8501	41650	182047	777	3554	93775	251952
497	7901	40558	188788	780	3440	121412	251573
481	7530	36986	181025	776	3458	97605	250457
492	7525	37067	178876	786	3493	107155	244517
504	7852	37716	182717	772	3439	91557	244974
491	7598	37605	173137	759	3555	91555	259883
482	7337	36906	175671	779	3461	93721	249087
478	7281	36568	190969	806	3444	116935	252534
475	7561	35918	180669	784	3507	104605	251129
497	7596	40685	186716	758	3438	137630	250449
488	7434	34613	179980	784	3382	96200	250176
484	7161	33842	178574	764	3471	134184	262974
481	7501	36075	189413	760	3470	127849	251927
489	7540	37673	190965	781	3507	115844	258421
482	7338	45352	166708	774	3398	103602	243415
509	8046	37234	182955	790	3596	119433	251945

Ottenuta tale informazione è possibile calcolare la **dimensione campionaria**, con un livello di confidenza del 95%. Per ogni dimensione e per ogni sistema, è stata calcolata quindi la dimensione campionaria tramite la formula

$$n = \left(\frac{z_\alpha \cdot \sigma}{E} \right)^2 \tag{1.1}$$

dove:

- z_α=1.96 è l'α-quantile della distribuzione normale, con α pari a 0.95;
- σ è la deviazione standard dei precampioni;
- **E** è definito a priori dall'utente e rappresenta l'errore massimo che si accetta di commettere; per questa analisi si è deciso di fissarlo al 5%.

Una volta determinata la dimensione campionaria opportuna, sono state eventualmente raccolte ulteriori osservazioni, in modo da ottenere un campione per ogni dimensione e per ogni sistema. I dati raccolti hanno evidenziato però una dimensione campionaria uguale o inferiore alla dimensione del precampione già raccolto, come mostrato dalle tabelle seguenti, e si è quindi ritenuto superfluo raccogliere osservazioni aggiuntive.

	First System - AMD Ryzen 5 4600H			
	1000	**20000**	**100000**	**500000**
Means	490,5666667	7470,033333	36887,2	182288,8667
Medians	491,5	7501	36737	181457,5
Std_dev	9,445609574	237,9450072	2018,598701	6914,209241
E-value	24,52833333	373,5016667	1844,36	9114,443333
N-value	0,569686653	1,55912253	4,601725748	2,210735987
Sample-Size	**1**	**2**	**5**	**3**

	Second System - Intel i5 8250U			
	1000	**20000**	**100000**	**500000**
Means	774,8666667	3481,333333	113458,6667	251031,5667
Medians	778	3469,5	114955,5	250777
Std_dev	14,08039315	63,76050736	15794,45547	6786,911794
E-value	38,74333333	174,0666667	5672,933333	12551,57833
N-value	0,507396818	0,51544751	29,77872494	1,12320718
Sample-Size	**1**	**1**	**30**	**2**

Dall'analisi preliminare di tali valori si potrebbe già concludere che entrambi i sistemi rispettano l'ipotesi di normalità: considerata la scarsa variazione tra i due valori, la distribuzione dei precampioni è assimilabile ad una distribuzione normale. Tale ipotesi sarà verificata nei paragrafi successivi attraverso ulteriori test.

Formule usate

- **E-value** $= \overline{x}_{5\%}$ (5% del valore della media campionaria)
- **N-value** $= \left(\frac{z_\alpha \cdot \sigma}{E}\right)^2$
- **Sample-Size** = intero maggiorante del N-value

1.4 Analisi dei sistemi

1.4.1 Primo sistema

A valle della raccolta delle osservazioni costituenti il precampione del primo sistema, per avere conferma dell'ipotesi di normalità, è stato realizzato per ogni dimensione un **QQ-plot** (plot quantile-quantile), utilizzando *JMP*.

La stima di normalità è molto robusta al test visivo: dai test eseguiti sul primo sistema, si evince infatti come tutte le distribuzioni siano approssimabili a distribuzioni normali, come si vede in figura.

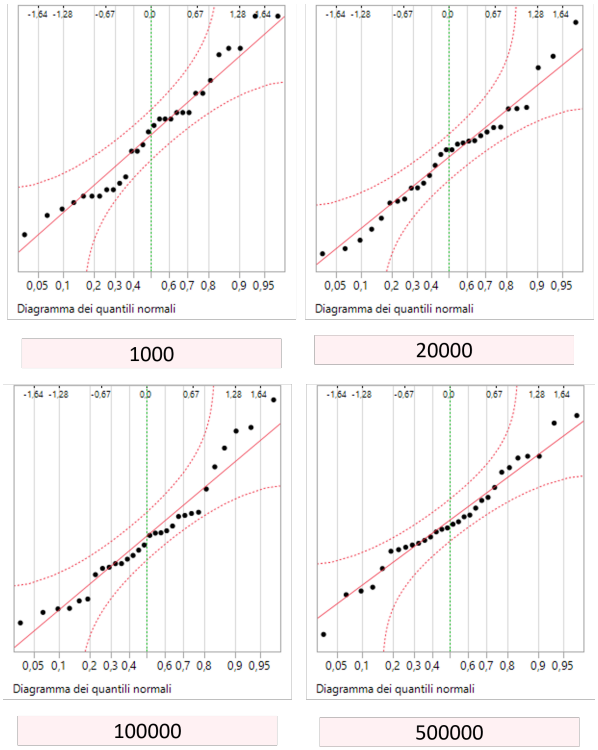


Figura 1.1: Analisi di normalità del sistema con AMD

1.4.2 Secondo sistema

A valle della raccolta delle osservazioni costituenti il precampione del secondo sistema, per avere conferma dell'ipotesi di normalità, è stato realizzato per ogni dimensione un **QQ-plot** (plot quantile-quantile), utilizzando *JMP*.

La stima di normalità è molto robusta al test visivo: dai test eseguiti sul secondo sistema, si evince infatti come tutte le distribuzioni siano approssimabili a distribuzioni normali, come si vede in figura.

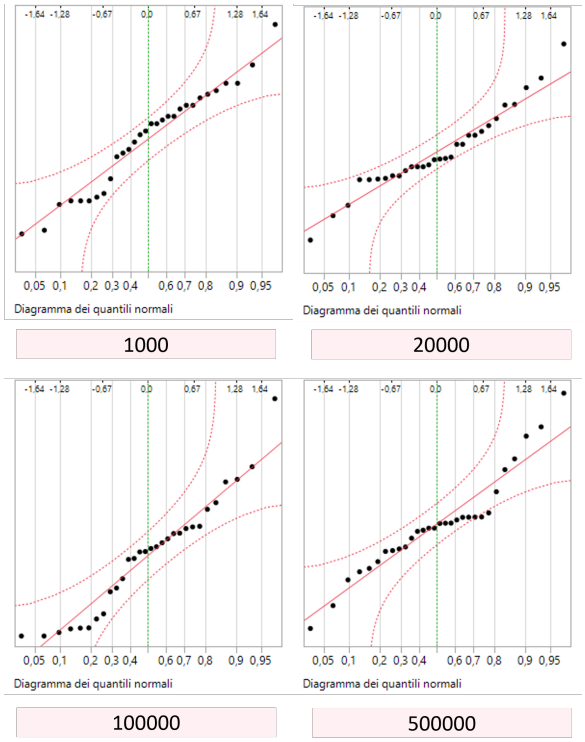


Figura 1.2: Analisi di normalità del sistema con Intel

1.5 Confronto sui dati ottenuti

Al fine di produrre un analisi per verificare che i campioni provenienti dalle varie distribuzioni siano statisticamente diversi, sono stati prodotti dei *campioni differenza*, ottenuti sottraendo, per ogni dimensione, i campioni del primo e del secondo sistema, secondo la formula AMD - INTEL. Tale operazione ha portato alla costruzione della seguente tabella.

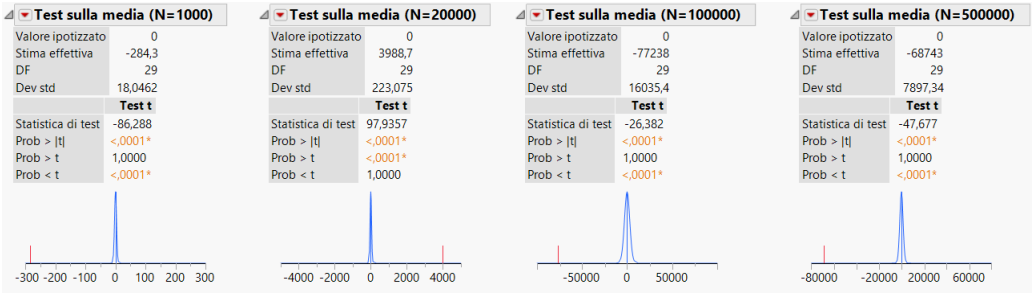
Tramite l'utilizzo di JMP, si è analizzata quindi la significatività statistica dei vari campioni differenza raccolti. Nel dettaglio, l'obiettivo è quello di dimostrare che le differenze di prestazioni dei due sistemi sono statisticamente significative e che non sono dovute ad effetti aleatori. Per fare ciò, si è scelto di eseguire un **T-Test paired** con lo scopo di rigettare l'ipotesi nulla H0: media μ pari a 0. Di fatto si sta eseguendo uno **Zero-Mean Test** sul campione differenza. Di seguito sono riportati i risultati.

Da tali risultati si evince come l'ipotesi nulla sia rigettata per tutte le dimensioni considerate. Tale risultato ci porta a dire che i campioni raccolti dai due sistemi sono STATISTICAMENTE SIGNIFICATIVI, ma non per ogni dimensione trattata, e che quindi i due sistemi sono differenti statisticamente solo in parte. Infatti si nota come per la dimensione 20000, è presente una variazione delle performance a favore del sistema con Intel. In generale però, come è possibile verificare tramite i campioni differenza ottenuti, il sistema con AMD risulta mediamente più performante rispetto al sistema con Intel, perchè sono stati registrati tempi di esecuzione generalmente più bassi, a conferma del fatto che sulla carta il sistema con AMD risulta migliore.

diff_1000	diff_20000	diff_100000	diff_500000
-265	3739	-79580	-87612
-305	3656	-59297	-67733
-286	3889	-76275	-75890
-268	3929	-80102	-78991
-264	4149	-56997	-64190
-276	4183	-79579	-55666
-278	3918	-118566	-61663
-302	3565	-83796	-61259
-296	4139	-92099	-63385
-290	4096	-97278	-60954
-291	3900	-83102	-68781
-293	3604	-85497	-62267
-246	3959	-76012	-69378
-248	3846	-91350	-68309
-298	3947	-52125	-69905
-283	4461	-80854	-62785
-295	4072	-60619	-69432
-294	4032	-70088	-65641
-268	4413	-53841	-62257
-268	4043	-53950	-86746
-297	3876	-56815	-73416
-328	3837	-80367	-61565
-309	4054	-68687	-70460
-261	4158	-96945	-63733
-296	4052	-61587	-70196
-280	3690	-100342	-84400
-279	4031	-91774	-62514
-292	4033	-78171	-67456
-292	3940	-69250	-76707
-281	4450	-82199	-68990

1.6 Approfondimento sistemi senza Virtualizzazione

Terminata l'analisi fatta tra i due sistemi in condizione di virtualizzazione, si è deciso di perfezionare l'analisi andando a verificare i dati anche senza la presenza della macchina virtuale. Infatti, le prestazioni di un ambiente virtualizzato dipendono da come la macchina virtuale risponde all'esecuzione del software guest, e, quindi, da come l'hypervisor emula e riproduce il comportamento dell'hardware fisico. Altro problema in cui è facile incorrere è la contesa della CPU fisica: la contesa sulla CPU è un evento in cui i singoli componenti e macchine della CPU in un sistema hardware virtualizzato attendono troppo tempo per il loro turno durante l'elaborazione. In un tale sistema, le risorse (ad es. CPU, memoria, ecc.) sono distribuite tra macchina virtuale e sistema host. Poiché risorse di elaborazione diverse sono assegnate a macchine diverse, gli scheduler nel sistema ordinano l'input / output e altre attività. L'elaborazione di queste attività viene ritardata



Test sulla media (N=1000)

Valore ipotizzato	0
Stima effettiva	-284,3
DF	29
Dev std	18,0462
	Test t
Statistica di test	-86,288
Prob > |t|	<,0001*
Prob > t	1,0000
Prob < t	<,0001*

Test sulla media (N=20000)

Valore ipotizzato	0
Stima effettiva	3988,7
DF	29
Dev std	223,075
	Test t
Statistica di test	97,9357
Prob > |t|	<,0001*
Prob > t	<,0001*
Prob < t	1,0000

Test sulla media (N=100000)

Valore ipotizzato	0
Stima effettiva	-77238
DF	29
Dev std	16035,4
	Test t
Statistica di test	-26,382
Prob > |t|	<,0001*
Prob > t	1,0000
Prob < t	<,0001*

Test sulla media (N=500000)

Valore ipotizzato	0
Stima effettiva	-68743
DF	29
Dev std	7897,34
	Test t
Statistica di test	-47,677
Prob > |t|	<,0001*
Prob > t	1,0000
Prob < t	<,0001*

Figura 1.3: Test sulla media dei dati differenze

quando le macchine a esse assegnate riscontrano una contesa della CPU. Tale problema risulta molto frequente in sistemi virtualizzati. Tuttavia, esistono diversi modi per analizzare il sistema per scoprire se la contesa della CPU è un problema: una metrica chiamata *ready percentage* (% pronta) mostra per quanto tempo una macchina deve attendere la potenza di elaborazione. Quando questo numero sale troppo in alto, indica contesa della CPU. Per confermare l'evidenza che per entrambi i sistemi la virtualizzazione è un ostacolo, si è deciso di eseguire il benchmark senza l'utilizzo di virtual machine, e confrontare i dati ottenuti con e senza virtual machine. Il dati a confronto per il sistema AMD sono riportati nella tabella seguente.

AMD Ryzen 5 4600H con VM				AMD Ryzen 5 4600H senza VM			
1000	20000	100000	500000	1000	20000	100000	500000
493	7208	39963	181208	212	4045	19938	95184
483	7114	34210	196502	216	4082	19615	95414
493	7538	36389	179519	213	4003	19449	96653
481	7273	34333	172106	219	4034	19724	95596
494	7682	35532	172614	217	4056	19621	95733
494	7676	35791	195477	215	4062	19779	95623
503	7391	37567	178248	215	4032	19182	95833
488	7078	36997	190721	214	4067	19454	96428
499	7577	35919	183907	212	4033	19170	96517
493	7677	36204	184946	214	4088	19631	95585
480	7358	39305	179115	213	4064	19160	96340
494	7056	35751	185358	214	4099	19700	95860
504	7481	38529	178028	210	4041	19608	96108
509	7290	34678	181707	212	4157	19328	95354
479	7501	41650	182047	215	4038	19514	95849
497	7901	40558	188788	212	4105	19669	96817
481	7530	36986	181025	214	4047	19620	96337
492	7525	37067	178876	214	4076	19305	96136
504	7852	37716	182717	212	4137	19570	95777
491	7598	37605	173137	212	4041	19611	95223
482	7337	36906	175671	212	4091	19232	95658
478	7281	36568	190969	211	3995	19638	95658
475	7561	35918	180669	217	4049	19690	95752
497	7596	40685	186716	213	4025	19464	95594
488	7434	34613	179980	214	4069	19577	95706
484	7161	33842	178574	212	3999	19433	96539
481	7501	36075	189413	214	4060	19587	95569
489	7540	37673	190965	217	4025	19314	96242
482	7338	34352	166708	216	4077	19631	95723
509	8046	37234	182955	218	4087	19685	95666

Si è fatto lo stesso con il sistema Intel, ottenendo le seguenti tabelle di confronto.

Intel Core i5 8250U con VM				Intel Core i5 8250U senza VM			
1000	20000	100000	500000	1000	20000	100000	500000
758	3469	119543	268820	345	6514	33275	155443
788	3458	93507	264235	344	6397	34035	152448
779	3649	112664	255409	297	6226	33545	139323
749	3344	114435	251097	342	3984	31465	151827
758	3533	92529	236804	310	6005	31497	150542
770	3493	115370	251143	349	7447	29680	164380
781	3473	156133	239911	355	7237	30867	152856
790	3513	120793	251980	352	7076	30211	153632
795	3438	128018	247292	350	7220	30049	150157
783	3581	133482	245900	347	6560	30462	152894
771	3458	122407	247896	380	5939	29667	153570
787	3452	121248	247625	361	7073	31834	146721
750	3522	114541	247406	358	4568	29977	148592
757	3444	126028	250016	406	7252	30135	146203
777	3554	93775	251952	393	5763	28604	138737
780	3440	121412	251573	347	5927	29798	148312
776	3458	97605	250457	361	6405	29281	146276
786	3493	107155	244517	337	5439	29686	146717
772	3439	91557	244974	389	5712	31258	144437
759	3555	91555	259883	347	7599	31076	157324
779	3461	93721	249087	350	5782	31163	156969
806	3444	116935	252534	347	5634	28030	146714
784	3507	104605	251129	344	6333	30516	149161
758	3438	137630	250449	321	5768	30274	152532
784	3382	96200	250176	357	6627	30365	141905
764	3471	134184	262974	334	7045	31069	144252
760	3470	127849	251927	367	6070	30786	142836
781	3507	115844	258421	349	5609	33063	154318
774	3398	103602	243415	369	6913	31189	148831
790	3596	119433	251945	343	6534	27941	146135

Anche per i dati raccolti senza virtual machine si è deciso di andare a calcolare alcune statistiche per verificarne la normalità. Tali statistiche sono riportate nelle seguenti tabelle.

	AMD Ryzen 5 4600H no Virtual Machine			
	1000	**20000**	**100000**	**500000**
Means	213,9666667	4059,466667	19529,96667	95882,46667
Medians	214	4058	19597,5	95742,5
Std_dev	2,189053225	37,18521521	192,995709	429,8471046
E-value	10,69833333	202,9733333	976,4983333	4794,123333
N-value	0,175092287	0,140361905	0,16335744	0,033619886
Sample-Size	**1**	**1**	**1**	**1**

	Intel i5 8250U no Virtual Machine			
	1000	**20000**	**100000**	**500000**
Means	351,7	6288,6	30693,26667	149468,1
Medians	349	6365	30489	148996
Std_dev	22,18752	823,852121	1462,160351	5606,188
E-value	17,585	314,43	1534,663333	7473,407
N-value	6,115684	26,3731934	3,487192444	2,161774
Sample-Size	**7**	**27**	**4**	**3**

Dai dati ottenuti è visibile un elevato miglioramento delle performance del benchmark tra esecuzione con VM e senza VM. Caso particolare resta sempre la dimensione 20000, perchè eseguita senza virtualizzazione il risultato della CPU AMD risulta decisamente migliore rispetto a quello ottenuto su Intel senza considerare la VM.

Per verificare se la macchina virtuale sia un parametro che influenza maggiormente le performance di un sistema rispetto ad un altro, si è scelto di confrontare le differenze tra i dati con e senza virtual machine di entrambi i sistemi, per verificare che tali differenze siano più o meno significative su un sistema rispetto all'altro.

Le differenze sono state calcolate tramite la formula Dati con VM - Dati senza VM, ottenendo i dati in tabella.

AMD (VM-NoVM)				Intel (VM-NoVM)			
1000	20000	100000	500000	1000	20000	100000	500000
281	3163	20025	86024	413	-3045	86268	113377
267	3032	14595	101088	444	-2939	59472	111787
280	3535	16940	82866	482	-2577	79119	116086
262	3239	14609	76510	407	-640	82970	99270
277	3626	15911	76881	448	-2472	61032	86262
279	3614	16012	99854	421	-3954	85690	86763
288	3359	18385	82415	426	-3764	125266	87055
274	3011	17543	94293	438	-3563	90582	98348
287	3544	16749	87390	445	-3782	97969	97135
279	3589	16573	89361	436	-2979	103020	93006
267	3294	20145	82775	391	-2481	92740	94326
280	2957	16051	89498	426	-3621	89414	100904
294	3440	18921	81920	392	-1046	84564	98814
297	3133	15350	86353	351	-3808	95893	103813
264	3463	22136	86198	384	-2209	65171	113215
285	3796	20889	91971	433	-2487	91614	103261
267	3483	17366	84688	415	-2947	68324	104181
278	3449	17762	82740	449	-1946	77469	97800
292	3715	18146	86940	383	-2273	60299	100537
279	3557	17994	77914	412	-4044	60479	102559
270	3246	17674	80013	429	-2321	62558	92118
267	3286	16930	95311	459	-2190	88905	105820
258	3512	16228	84917	440	-2826	74089	101968
284	3571	21221	91122	437	-2330	107356	97917
274	3365	15036	84274	427	-3245	65835	108271
272	3162	14409	82035	430	-3574	103115	118722
267	3441	16488	93844	393	-2600	97063	109091
272	3515	18359	94723	432	-2102	82781	104103
266	3261	14721	70985	405	-3515	72413	94584
291	3959	17549	87289	447	-2938	91492	105810

Per tali dati è evidentemente rispettata l'ipotesi di normalità, visto che i dati d'origine provenivano da distribuzioni normali.

Una prima analisi che è possibile fare dando uno sguardo alle differenze ottenute è che, essendo esse influenzate unicamente dal fattore Macchina Virtuale, visto che i dati sono stati raccolti nelle medesime condizioni, e sulla stessa macchina, se si presenta un evidenza di sbilanciamento verso uno dei due sistemi è possibile concludere che l'influenza della macchina virtuale sia stata maggiore su quel sistema piuttosto che sull'altro. In particolare, come si vede dai dati stessi, sembra che le differenze ottenute siano maggiori sul sistema Intel, e ciò fa pensare che il sistema Intel sia quello che ha maggiormente beneficiato dell'esecuzione del benchmark senza virtualizzazione.

Tale evidenza può essere confermata dall'analisi sulle differenze tra le differenze appena ottenute: si è quindi scelto di procedere allo studio della significatività di tali differenze effettuando un t-test

accoppiato, come fatto per il primo confronto tra i due sistemi.

AMD (VM-NoVM) - Intel (VM-NoVM)			
diff_1000	diff_20000	diff_100000	diff_500000
-132	6208	-66243	-27353
-177	5971	-44877	-10699
-202	6112	-62179	-33220
-145	3879	-68361	-22760
-171	6098	-45121	-9381
-142	7568	-69678	13091
-138	7123	-106881	-4640
-164	6574	-73039	-4055
-158	7326	-81220	-9745
-157	6568	-86447	-3645
-124	5775	-72595	-11551
-146	6578	-73363	-11406
-98	4486	-65643	-16894
-54	6941	-80543	-17460
-120	5672	-43035	-27017
-148	6283	-70725	-11290
-148	6430	-50958	-19493
-171	5395	-59707	-15060
-91	5988	-42153	-13597
-133	7601	-42485	-24645
-159	5567	-44884	-12105
-192	5476	-71975	-10509
-182	6338	-57861	-17051
-153	5901	-86135	-6795
-153	6610	-50799	-23997
-158	6736	-88706	-36687
-126	6041	-80575	-15247
-160	5617	-64422	-9380
-139	6776	-57692	-23599
-156	6897	-73943	-18521

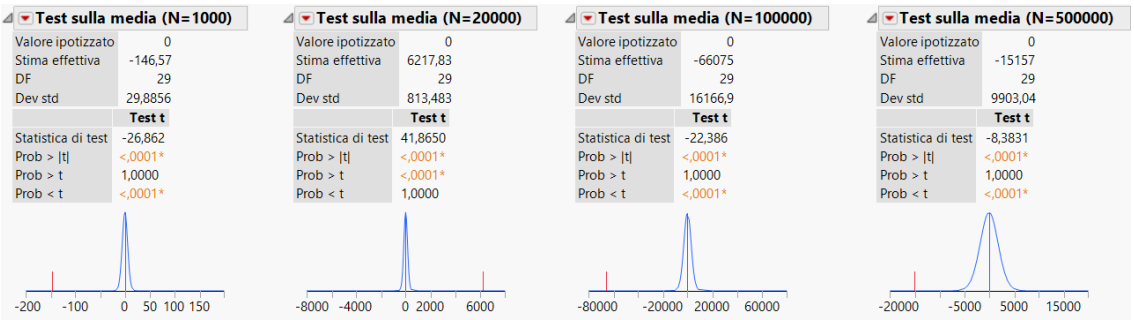


Figura 1.4: T-Test sulle differenze tra VM-NoVM dei sistemi

É quindi visibile che il miglioramento maggiore c'è stato sul sistema con Intel, piuttosto che sul

sistema AMD, perchè per la totalità delle dimensioni si rigetta l'ipotesi nulla e in particolare si ottiene che il sistema Intel ha differenze temporali tra con e senza VM superiori rispetto a quelle del sistema con AMD.

Capitolo 2

Workload Characterization

2.1 Traccia

Effettua la caratterizzazione del workload si un sistema bancario, analizzando il dataset ed effettuando operazioni usando le tecniche di *Principal Component Analysis* e di *Clustering*. Successivamente studiare la devianza in funzione della dimensione del workload.

2.2 Workload Characterization

Il processo di caratterizzazione di un workload ha lo scopo di generare un workload sintetico per il caso di test, basandosi su un workload reale. In altre parole, si estrae un modello sintetico del workload reale, tale che sia in grado di rappresentarne la variabilità. Il workload reale è composto da 24 componenti (colonne del dataset) e da 3000 osservazioni (righe del dataset), per un totale di 72000 valori da analizzare.

2.2.1 Analisi preliminare

E' buona norma, prima di applicare le tecniche indicate, evidenziare all'interno del dataset quali componenti non risultano importanti per la costruzione del modello sintetico dei dati;

Comprensione dei dati

Prima di guardare i dati da un punto di vista analitico, si è pensato di andare a verificare che cosa indica ogni distribuzione. Per fare ciò è necessario ricercare quali sono i valori da analizzare e cosa essi indicano: il dataset ha una sua descrizione al seguente link. Da tale documento si evince la seguente descrizione per la distribuzioni in analisi:

- ***VmPeak***: picco della memoria virtuale;

- ***VmSize***: dimensione della memoria virtuale;
- ***VmHWM***: picco della porzione di memoria virtuale occupata da un processo ("high water mark");
- ***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***: valore di memoria libera, derivante dalla somma di due quantità;
- ***Buffers***: archiviazione temporanea per blocchi di dischi grezzi che non dovrebbero diventare tremendamente grandi (20 MB circa);
- ***Cached***: quantità di cache per la lettura di file da disco;
- ***Active***: memoria che è stata usata più recentemente e di solito 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;
- ***Writeback***: dati attivamente riscritti in memoria;
- ***AnonPages***: Pagine non supportate da file mappate in tabelle di pagina nello spazio utente;
- ***Mapped***: file mappati in memoria, come ad esempio 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. La memoria impegnata è una somma di tutta la memoria che è stata allocata dai processi, anche se essi non l'hanno ancora "usata";
- ***NumOfAllocFH***: descrizione non disponibile;
- ***proc-fd***: descrizione non disponibile;
- ***avgThroughput***: throughput medio applicato sul sistema;
- ***avgElapsed***: tempo medio di risposta delle richieste;
- ***avgLatency***: latenza media misurata;

- ***Errors***: numero di errori incorsi.

Da tale analisi, è possibile quindi effettuare una riduzione preliminare del dataset, andando a verificare se alcune colonne possono risultare ridondanti o superflue. Infatti, come evidenziato dalla loro descrizione, i valori di ***VmPeak*** e di ***VmHWM*** sono semplicemente dei valori di picco della memoria usata e della memoria del processo, ed essendo entrambi legati al valore di ***VmRSS*** risulta inutile includerli nell'analisi. Per tutti gli altri dati, non è possibile trarre alcuna conclusione preliminare, ed è quindi opportuno andare a considerare tutte le distribuzioni restanti, nonostante essi dovessero mostrare un alto valore di correlazione che sia però minore dell'unità. A valle di tale attività, il dataset conta 22 componenti e 3000 istanze.

2.2.2 Preprocessing dei dati

Dopo la fase di comprensione dei dati, è buona norma andare a considerare le componenti restanti per verificarne l'effettiva importanza all'interno dell'analisi, andando ad escludere dati a bassa variabilità e ad alta correlazione. Eímportante ricordare che le colonne che si sceglie di non considerare nelle fasi preliminari dell'analisi, vengono escluse solo dal processo di analisi stesso, ma al fine di caratterizzare il workload sintetico tali distribuzioni devono essere ugualmente conteggiate.

Dati costanti

All'interno di un vasto dataset, è buona norma andare ad escludere distribuzioni a variabilità nulla, in quanto non offriranno alcun contributo alla varianza dei dati totali. Per il dataset trattato, infatti, in figura è possibile notare come le colonne **Active**, **AnonPages**, **avgLatency** e **Errors** risultano costanti, e quindi è opportuno scartarle.

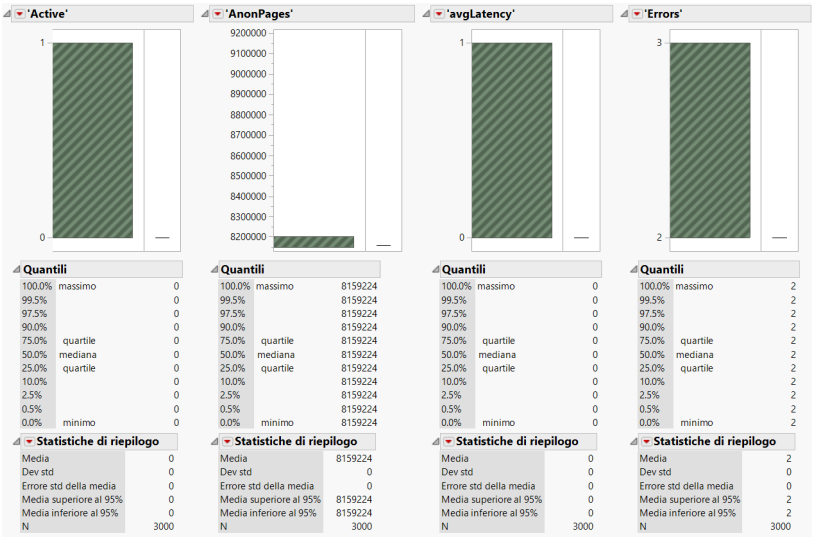


Figura 2.1: Analisi delle colonne costanti

Guardando quindi i dati per tali distribuzioni, infatti, avremo delle colonne che presentano sempre lo stesso valore, come è possibile notare nell'estratto del dataset in tabella.

	Active	AnonPages	avgLatency	Errors
1	0	8159224	0	2
2	0	8159224	0	2
3	0	8159224	0	2
4	0	8159224	0	2
5	0	8159224	0	2

A valle di questa ulteriore riduzione, si ottiene un dataset che comprende 18 componenti.

Dati correlati

Per identificare la correlazione tra i dati, quello che si fa è calcolare la matrice di correlazione tra i dati stessi, e se si evidenzia un elemento della matrice a valore pari a **1** in corrispondenza della coppia di variabili, è possibile concludere che quelle variabili sono correlate, e quindi è opportuno andare a considerare solo una delle due.
Tramite il tool JMP, è possibile quindi tracciare tale tabella, che assumerà i seguenti valori:

Correlazioni

	'VmPeak'	'VmSize'	'VmHWM'	'VmRSS'	'VmPTE'	'Threads'	'MemFree'	'Buffers'	'Cached'	'Inactive'	'Dirty'	'Writeback'	'Mapped'	'Slab'	'PageTables'	'Committed_AS'	'NumOfAllocFH'	'proc-fd'	'avgThroughput'	'avgElapsed'
'VmPeak'	100%	99%	98%	94%	99%	60%	-49%	70%	35%	76%	30%	-49%	-6%	-1%	88%	5,1%	42%	51%	96%	-48%
'VmSize'	99%	100%	98%	97%	99%	59%	-52%	69%	38%	76%	33%	-52%	-6%	-1%	90%	5,1%	44%	52%	97%	-45%
'VmHWM'	98%	98%	100%	97%	98%	56%	-40%	64%	26%	68%	21%	-40%	-6%	-1%	92%	5,9%	32%	53%	94%	-46%
'VmRSS'	94%	97%	97%	100%	95%	55%	-39%	58%	25%	64%	20%	-39%	-5%	-1%	94%	5,5%	30%	53%	91%	-42%
'VmPTE'	99%	99%	98%	95%	100%	61%	-44%	67%	30%	72%	24%	-44%	-5%	-1%	89%	6,0%	37%	53%	96%	-47%
'Threads'	60%	59%	56%	55%	61%	100%	-22%	47%	10%	44%	6,8%	-22%	-2%	-0%	53%	9,1%	20%	36%	61%	-34%
'MemFree'	-49%	-52%	-40%	-39%	-44%	-22%	100%	-76%	-98%	-92%	-97%	100%	5,3%	0,4%	-34%	0,1%	-99%	-12%	-52%	14%
'Buffers'	70%	69%	64%	58%	67%	47%	-76%	100%	61%	90%	62%	-76%	-12%	-4%	52%	5,0%	69%	25%	69%	-33%
'Cached'	35%	38%	26%	25%	30%	10%	-98%	61%	100%	83%	99%	-98%	-4%	-0%	22%	-4%	99%	3,3%	38%	-6%
'Inactive'	76%	76%	68%	64%	72%	44%	-92%	90%	83%	100%	80%	-92%	-1%	2,7%	62%	12%	88%	33%	78%	-30%
'Dirty'	30%	33%	21%	20%	24%	6,8%	-97%	62%	99%	80%	100%	-97%	-8%	-2%	16%	-8%	98%	-2%	33%	-4%
'Writeback'	-49%	-52%	-40%	-39%	-44%	-22%	100%	-76%	-98%	-92%	-97%	100%	5,3%	0,4%	-34%	0,1%	-99%	-12%	-52%	14%
'Mapped'	-6%	-6%	-6%	-5%	-5%	-2%	5,3%	-12%	-4%	-1%	-8%	5,3%	100%	45%	21%	82%	-2%	51%	9,8%	9,7%
'Slab'	-1%	-1%	-1%	-1%	-1%	-0%	0,4%	-4%	-0%	2,7%	-2%	0,4%	45%	100%	12%	45%	-0%	21%	6,8%	4,4%
'PageTables'	88%	90%	92%	94%	89%	53%	-34%	52%	22%	62%	16%	-34%	21%	12%	100%	38%	28%	70%	92%	-37%
'Committed_AS'	5,1%	5,1%	5,9%	5,5%	6,0%	9,1%	0,1%	5,0%	-4%	12%	-8%	0,1%	82%	45%	38%	100%	1,8%	62%	25%	4,6%
'NumOfAllocFH'	42%	44%	32%	30%	37%	20%	-99%	69%	99%	88%	98%	-99%	-2%	-0%	28%	1,8%	100%	11%	46%	-11%
'proc-fd'	51%	52%	53%	53%	53%	36%	-12%	25%	3,3%	33%	-2%	-12%	51%	21%	70%	62%	11%	100%	62%	-21%
'avgThroughput'	96%	97%	94%	91%	96%	61%	-52%	69%	38%	78%	33%	-52%	9,8%	6,8%	92%	25%	46%	62%	100%	-43%
'avgElapsed'	-48%	-45%	-46%	-42%	-47%	-34%	14%	-33%	-6%	-30%	-4%	14%	9,7%	4,4%	-37%	4,6%	-11%	-21%	-43%	100%

Le correlazioni sono stimate per metodo A livello di riga.

Figura 2.2: Heatmap delle correlazioni percentuali

Da quello che si può notare dalle due figure, tra le due componenti *WriteBack* e *MemFree* c'è una perfetta correzione, il che indica una dipendenza tra le due variabili. Questo è dato dal fatto che nella matrice di correlazione si trova un 1 situato al di fuori della diagonale principale della matrice. Si può quindi escludere una delle due componenti, ad esempio MemFree, dal dataset. Tale esclusione è possibile anche confermarla tramite un calcolo della differenza tra le due colonne; avendo una correlazione pari a uno ci si aspetta di avere una differenza identicamente nulla per le due distribuzione, come infatti avviene.

Eliminando tale correlazione, il dataset consta di 17 componenti.

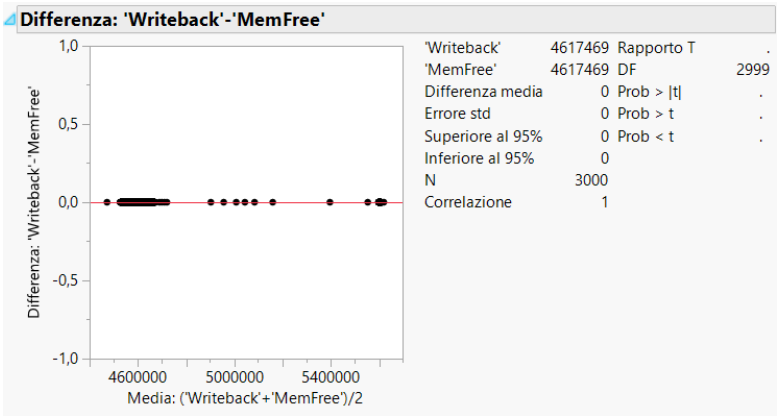


Figura 2.3: Differenza delle distribuzioni di Writeback e MemFree

Standardizzazione dei dati

Per fare si che l'analisi alle componenti principali (PCA) non risulti influenzata dal valore assoluto dai dati, è necessario eseguire una normalizzazione dei dati stessi. In particolare, è possibile utilizzare la normalizzazione **z-score**, che consiste nel normalizzare in media e varianza tutti i dati su cui stiamo lavorando, andando ad applicare la formula seguente:

$$z = \frac{x - \overline{x}}{\sigma_x} \tag{2.1}$$

La serie di dati z avrà media nulla e varianza unitaria.

2.2.3 Ulteriori considerazioni

Valori di Slab

La colonna Slab presenta unicamente due valori non nulli, che sono quindi considerabili outlier. Tale colonna potrebbe essere non considerata ai fini dell'analisi, se si verifica che i due valori non nulli non sono effettivamente dei valori significativi. In questo caso, si deve verificare che ai soli due valori non nulli della componente Slab, corrispondano valori che dimostrano la tesi che tali valori sono effettivamente outliers e tale distribuzione può essere non considerata; questo è verificabile andando a effettuare un analisi di tali istanze e dei valori del dataset nell'intorno di tali istanze. Considerato che l'outlier della distribuzione Slab si ha all'istanza 90 del dataset, si è deciso di considerare un intorno di 10 istanze, quindi considerando le istanze da 80 a 100.

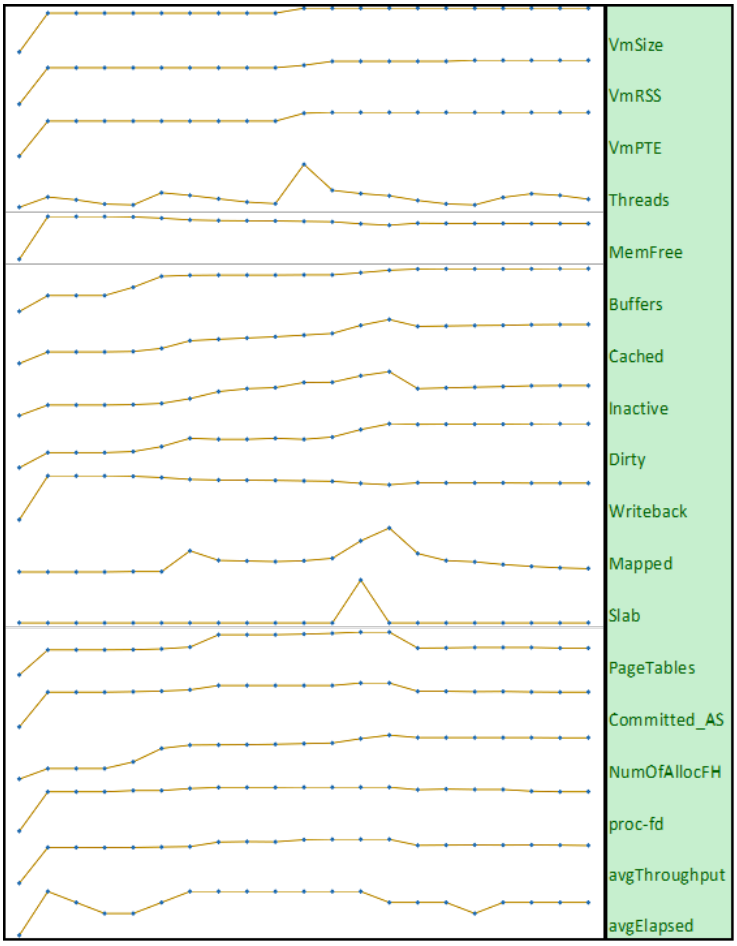


Figura 2.4: Andamento dei valori a cavallo del picco di Slab

- Intorno dell'istanza 80: Mapped e CommittedAS sono bassissimi e quindi, si presume che si sta usando davvero poco la memoria, in particolare ci sono solo poche applicazioni aperte (evidenziato dal valore di CommittedAS). Anche lo Slab è nullo, quindi il Kernel non sta lavorando.

- Intorno dell'istanza 84: il valore di Cached comincia a salire coerentemente con l'aumento del valore di Mapped e CommittedAS (vengono mappate applicazioni, drive ecc.).

- Intorno dell'istanza 90: picco di Slab e quindi attivazione del Kernel;

- Istanza 91: terminato il picco di Slab abbiamo dei notevoli picchi di Cached, Mapped e CommittedAS. Se fosse un webserver, si potrebbe ipotizzare che in questo momento qualche pagina dinamica abbia richiesto delle esecuzioni da fare.

- Intorno dell'istanza 100: il Sistema cambia completamente il suo comportamento, si notano alti livelli di Cache, mentre Mapped e CommittedAS sono calati nuovamente, qindi la memoria è stata deallocata (applicazioni terminate).

Prime righe del dataset

Si osserva poi che le prime righe del dataset, in particolare le prime 80, hanno dei valori che sono differenti dai valori che compaiono nell'intero dataset; questo potrebbe rappresentare uno stato di boot del sistema. Tale informazione è confermata dal fatto che, se si analizzano i valori di memoria libera del sistema per le prime righe del dataset si hanno dei valori decisamente più alti rispetto al resto dei valori assunti dal sistema per tale distribuzione. La memoria libera è quindi indice del fatto che il sistema è in fase di avviamento, e tale condizione potrebbe non essere considerevole ai fini dell'analisi. Ulteriore conferma si ha se si analizza la distribuzione relativa al throughput medio: infatti in corrispondenza delle prime righe del dataset, il throughput risulta molto basso rispetto agli altri valori e ciò conferma ancora una volta che si tratta di istanze relative ad una condizione particolare del sistema, probabilmente il suo avviamento.
Se si avessero dei limiti a livello di costi per l'analisi sarebbe consigliabile andare ad eliminare tali righe del dataset, ma per tale analisi si è deciso di tenere in conto anche queste istanze, per avere un workload sintetico che rappresenta meglio il dataset completo.

Controllo dei dati

Per fare un analisi visiva sulla qualità sui dati può essere importante capire se ci sono righe del dataset che non sono significative a sufficienza per far parte della mia analisi. Questo perchè una troppa concentrazione di dati che sono outlier in ogni distribuzione potrebbero essere indice di una troppa variabilità dei dati e andare a falsificare l'analisi.
Uno strumento che ci permette di farci un'idea visiva sui dati è il **box-plot**, che per i dati sotto test è rappresentato in figura.

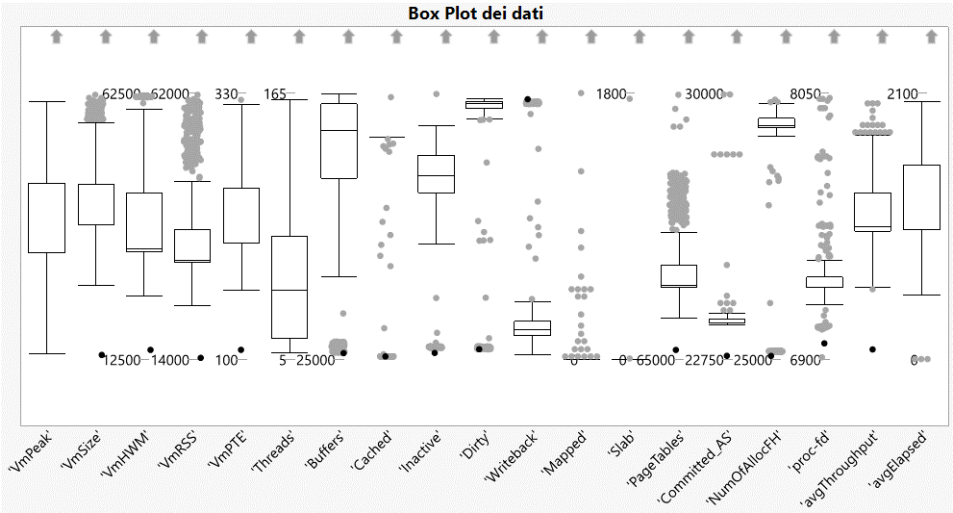


Figura 2.5: Box plot dei dati

2.2.4 Principal Component Analysis (PCA)

La prima tecnica che si applica ai dati così rielaborati è la **Principal Component Analysis** (PCA), che permette di trovare le componenti principali di un dataset per poi traslarle in uno spazio di stato differente. Le componenti principali di un set di dati sono le direzioni ortogonali della base ortonormale, nella quale le dimensioni dei dati sono linearmente indipendenti.

Inoltre, le componenti principali di un insieme di punti vengono ordinate per quantità di varianza massima spiegata dalla componente stessa: la prima componente principale è la prima direzione che spiega la massima varianza, la seconda componente è la seconda direzione che massimizza la varianza, ecc.

La PCA è una procedura preliminare che serve ad effettuare quello che viene detto **dimensionality reduction**: si proietta ogni punto del dataset in uno spazio costituito solo da alcune delle prime componenti principali, in modo da ottenere un dataset caratterizzato da una minore dimensionalità, conservando quanta più varianza possibile.

Tramite JMP, la PCA è possibile eseguirla andando a fare un analisi multivariata alle componenti principali, ottenendo quindi tali autovettori e autovalori.

Autovettori

	1 principale	2 principale	3 principale	4 principale	5 principale	6 principale	7 principale	8 principale	9 principale	10 principale	11 principale	12 principale	13 principale	14 principale	15 principale	16 principale	17 principale
VmSize	0,31895	0,14260	-0,16019	0,06391	0,13848	-0,11202	0,04308	0,10076	0,09381	0,16142	-0,21003	0,19709	-0,15660	0,52658	-0,62035	-0,00249	-0,00038
VmRSS	0,29067	0,20049	-0,18152	0,12685	0,18683	-0,19395	0,09076	0,13433	-0,02247	-0,49840	0,02417	-0,07449	0,16286	0,42501	0,51503	0,00358	0,00137
VmPTE	0,30640	0,17825	-0,18115	0,03996	0,12402	-0,07882	-0,02302	0,09863	0,15666	0,42740	-0,38020	0,36506	-0,14315	-0,35692	0,41852	0,00094	0,00040
Threads	0,19852	0,17035	-0,14901	-0,15167	-0,04225	0,87552	0,31421	0,04934	-0,05979	-0,09950	-0,01070	0,00047	-0,05685	-0,00398	0,00619	0,00140	0,00007
Buffers	0,29318	-0,10853	-0,03898	-0,09098	-0,04397	0,20326	-0,75440	-0,13246	0,29330	-0,23486	0,18024	0,18964	-0,00720	-0,02319	-0,02980	0,08332	0,21098
Cached	0,23510	-0,36079	0,16632	0,01228	-0,01677	-0,06894	0,25180	0,02012	-0,07984	0,00910	-0,07432	-0,16585	-0,27414	0,00320	0,05882	0,28592	0,71975
Inactive	0,33028	-0,14467	0,06085	-0,02208	-0,00865	0,04369	-0,19888	0,00776	0,02991	0,07764	-0,41718	-0,68598	0,11582	-0,06286	-0,03890	0,18532	-0,34561
Dirty	0,22139	-0,38734	0,15741	0,00613	-0,02553	-0,05039	0,16337	-0,01867	-0,03214	-0,15052	0,23247	0,28403	-0,41685	-0,00470	0,06436	0,30525	-0,56249
Writeback	-0,27502	0,31096	-0,12956	0,00457	0,02377	0,01076	-0,03920	0,01044	0,01407	0,05282	0,01340	0,05660	0,14745	0,02306	-0,02555	0,88264	0,01341
Mapped	0,01131	0,22417	0,53073	0,02368	-0,22473	-0,00171	0,11973	0,50929	0,57681	-0,07904	0,03955	-0,02108	0,02515	-0,03213	-0,02314	-0,00005	-0,00005
Slab	0,01346	0,13273	0,38467	-0,58649	0,67601	-0,03392	0,03538	-0,17233	0,02291	-0,01119	0,01011	0,00612	0,01623	-0,01030	-0,00458	-0,00006	-0,00004
PageTables	0,28359	0,27308	-0,00685	0,15660	0,09605	-0,17289	0,04110	0,14134	-0,31578	-0,39875	0,05489	0,02637	0,04581	-0,59502	-0,37347	0,01290	0,03776
Committed_AS	0,05614	0,25957	0,50592	0,04704	-0,22727	0,08975	-0,30858	0,08178	-0,61233	0,11907	-0,13598	0,13805	-0,10009	0,23212	0,13823	0,00091	0,00056
NumOfAllocFH	0,25689	-0,32762	0,16348	-0,00186	-0,06688	0,03021	0,17092	-0,02611	-0,07200	0,12194	-0,01481	0,33248	0,79145	0,01960	-0,05031	0,06737	0,00065
proc-fd	0,17622	0,32673	0,22069	0,21561	-0,22747	-0,05965	0,20430	-0,78701	0,21794	-0,03859	-0,02634	-0,02256	-0,01895	-0,00195	0,00508	0,00036	0,00003
avgThroughput	0,32200	0,17479	-0,04981	0,06028	0,05855	-0,03974	-0,00857	0,07285	-0,06072	0,49835	0,72128	-0,27043	0,03643	0,01302	0,04851	0,00131	-0,00010
avgElapsed	-0,14460	-0,12084	0,21991	0,72447	0,54519	0,28514	-0,08261	-0,00809	0,06150	0,04460	-0,00620	-0,00037	0,00472	-0,00119	-0,00043	0,00001	-0,00001

Figura 2.6: Autovettori ottenuti dalla PCA

Autovalori

Numero	Autovalore	Percentuale	20 40 60 80	Percentuale cumulativa
1	8,2204	48,355		48,355
2	3,4881	20,518		68,874
3	2,3527	13,839		82,713
4	0,7753	4,560		87,273
5	0,6855	4,032		91,306
6	0,6107	3,592		94,898
7	0,3382	1,989		96,887
8	0,2621	1,542		98,429
9	0,1580	0,929		99,358
10	0,0809	0,476		99,834
11	0,0124	0,073		99,907
12	0,0079	0,046		99,953
13	0,0058	0,034		99,987
14	0,0014	0,008		99,995
15	0,0008	0,004		100,000
16	0,0000	0,000		100,000
17	0,0000	0,000		100,000

Figura 2.7: Autovalori ottenuti dalla PCA

Facciamo delle considerazioni sugli autovalori trovati: infatti, come si vede dall'analisi, ci sono degli autovalori che sembra non spieghino alcuna varianza, come ad esempio si vede tra gli autovalori 15, 16 e 17. Ciò però non è vero, dal momento che, incrociando i dati tra autovalori e autovettori, se l'autovalore 17 ad esempio non spiegasse alcuna varianza, avrei che l'autovettore ad esso corrispondesse sia identicamente nullo, ma ciò non accade.
Altra conferma che è possibile fare è andando ad aumentare le cifre decimali analizzate da JMP. Infatti, come si vede dalla figura in basso, gli autovalori 15, 16 e 17 spiegano comunque un minimo di varianza cumulativa in più rispetto agli autovalori da 1 a 14, ma l'approssimazione fatta da JMP sulla tabella riduce la risoluzione a sole tre cifre decimali, non permettendo agevolmente di distinguere tale differenza.

Autovalori

Numero	Autovalore	Percentuale	20 40 60 80	Percentuale cumulativa
1	8,2204	48,355		48,355498
2	3,4881	20,518		68,873903
3	2,3527	13,839		82,71304
4	0,7753	4,560		87,273477
5	0,6855	4,032		91,305822
6	0,6107	3,592		94,898058
7	0,3382	1,989		96,8873
8	0,2621	1,542		98,428995
9	0,1580	0,929		99,35821
10	0,0809	0,476		99,83424
11	0,0124	0,073		99,906947
12	0,0079	0,046		99,953249
13	0,0058	0,034		99,987408
14	0,0014	0,008		99,995494
15	0,0008	0,004		99,999982
16	0,0000	0,000		99,999999
17	0,0000	0,000		100

Figura 2.8: Visualizzazione completa degli autovalori

Fatta l'analisi sugli autovalori, dai risultati capiamo quindi che se ipotizzassimo di prendere le prima 5 componenti, avremmo una perdita di varianza accettabile, dal momento che si andrebbe a conservare più del 91% della variabilità dei dati, mentre se si considerano le prime 6 componenti si raggiungerebbe una varianza totale di circa il 95%. É quindi cruciale trovare il giusto numero di componenti principali per conservare il massimo della varianza, e quindi come prima scelta si andrebbero a considerare 6 componenti a discapito di 5.
Tale considerazione è però fatta non tenendo conto del fatto che il lavoro che si è voluto effettuare sui dati non ha alcun tipo di vincolo su costi o su altri parametri che potrebbero influenzare la scelta del numero di componenti principali da considerare. Infatti, scegliere un numero di componenti principali troppo alto, porterebbe si una maggior conservazione della varianza dei dati, ma potrebbe consequenzialmente essere indice di costi elevati nel caso in cui si dovessero effettuare altri tipi di analisi. Questi aspetti saranno approfonditi nei successivi capitoli.

2.2.5 Clustering

Eseguita la PCA, e scelto un numero di componenti principali da conservare pari a 6, quello che si vuole fare è ridurre anche le istanze del dataset. Una procedura per fare ciò è effettuare un **clustering** dei dati. Quello utilizzato in questa trattazione è un clustering di tipo *gerarchico* che, applicando un approccio agglomerativo detto **metodo di Ward**, costruisce i cluster aggregando istanze secondo una metrica. Tale tecnica di clusterizzazione di per sé non introduce ulteriore perdita di varianza perché raggruppa le istanze minimizzando la possibile devianza nei cluster. L'obiettivo è quello di identificare classi che rappresentino comportamenti funzionali del sistema differenti, riducendo allo stesso tempo la dimensione dell'insieme dei dati. Lo scopo è quello di minimizzare la variabilità interna ad ogni cluster (devianza **intracluster**) e massimizzare quella esterna (devianza **intercluster**). Grazie al tool JMP è possibile eseguire facilmente il clustering sulle componenti trovate, per ottenere in output un **dendrogramma** dal quale è possibile estrarre alcune informazioni circa i dati nei cluster. Tale dendrogramma è visibile in figura, con riferimento a 6 componenti principali conservate.

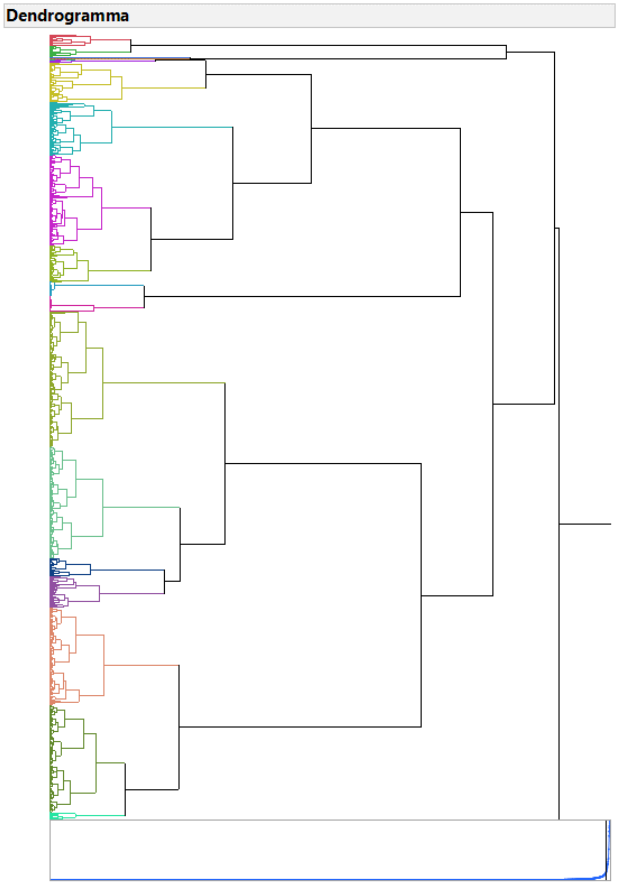


Figura 2.9: Dendrogramma riferito a 6 componenti principali

Il tool JMP suggerisce di scegliere un numero di cluster pari a 20, ma tale valore è troppo elevato, quindi si decide di analizzare il numero ottimale di cluster andando a verificare la distanza tra i vari cluster. Tali distanze, attraverso il tool JMP è possibile reperirle attraverso la finestra di cronologia di clusterizzazione.

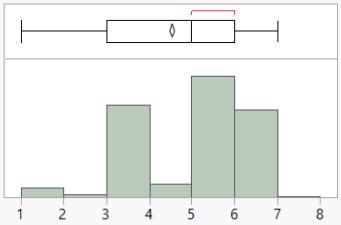
Cronologia di clusterizzazione

Numero di cluster	Distanza	Leader	Subordinato
1	54,53517424	1	90
2	53,95500809	1	92
3	48,85579295	1	84
4	47,44331983	92	96
5	43,91346385	92	2885
6	39,74335517	96	109
7	27,90785505	92	1873
8	19,53889309	1873	1920
9	18,77599091	96	121
10	16,65216714	92	2663
11	14,95998943	84	91
12	13,90127455	121	193
13	13,77032455	109	110
14	12,23787640	193	1863
15	11,26088840	92	2844
16	10,80291687	1920	1932
17	10,02522087	2885	2939
18	8,60799999	1	7
19	7,99804969	110	113
20	7,73398197	2663	2795
21	6,55117743	1873	1929
22	6,53542263	2663	2742

Figura 2.10: Cronologia di clusterizzazione

Da tali cluster e osservando le distanze, il numero ottimale da scegliere è quello che da una maggiore differenza tra l'aggiunta e la rimozione di un cluster. Infatti, come si vede dai dati a disposizione, nel caso di 7 cluster si avrebbe un aumento di distanza maggiore nel caso di aggiunta di un cluster, e una diminuzione sostanziale di distanza con la rimozione di uno di essi, andando a considerare tutte le possibili triple di cluster. La scelta ottima ricade quindi su un numero di cluster pari a 7. In definitiva, si avranno delle etichette per ogni istanza del workload reale che definiranno il cluster di appartenenza per tali istanze. Per andare a verificare la distribuzione delle istanze all'interno dei cluster, quello che è possibile fare è graficare l'istogramma di tale distribuzione, che appare come segue.

Cluster



Quantili

100.0%	massimo	7
99.5%		6
97.5%		6
90.0%		6
75.0%	quartile	6
50.0%	mediana	5
25.0%	quartile	3
10.0%		3
2.5%		1
0.5%		1
0.0%	minimo	1

Statistiche di riepilogo

Media	4,5466667
Dev std	1,3065443
Errore std della media	0,0238541
Media superiore al 95%	4,5934388
Media inferiore al 95%	4,4998946
N	3000

Figura 2.11: Distribuzione delle istanze nei cluster

Da tale analisi si vede come la distribuzione delle istanze sia sbilanciata, dal momento che alcuni cluster sono decisamente più ricchi di altri; ciò porta a dire che sarà cruciale la scelta delle istanze per la caratterizzazione del workload sintetico per alcuni cluster.

2.2.6 Calcolo delle devianze

Dopo aver fissato numero di cluster e componenti principali, si è deciso di andare ad analizzare la devianza dei dati che si sono analizzati e che si sta cercando di conservare, visto che rispetto alla varianza è un indice statistico di maggiore interesse.

Devianza pre-PCA

Per fare ciò si è dapprima calcolata la devianza totale dei dati del workload reale, utilizzando la formula

$$D_{tot} = \sum_{i=1}^{N} ||x_i - \overline{x}||^2 = 51000.00000000039 \tag{2.2}$$

dove x_i rappresenta l'i-esima istanza del dataset, $\overline{x}$ la media delle istanze, ed N il numero di istanze.

Devianza post-PCA

A valle dell'operazione di PCA, la devianza dei dati d'origine sarà sicuramente cambiata, dal momento che la PCA attua un cambiamento di stato. Tale devianza, se si sceglie, come fatto, di conservare circa il 94% di varianza dei dati salvando 6 componenti principali, sarà pari a

$$D_{post-PCA} = \sum_{i=1}^{N} ||x_{PCA_i} - \overline{x}_{PCA}||^2 = 48398.00980496028 \tag{2.3}$$

dove invece x_{PCA_i} rappresenta l'i-esima istanza del dataset nello spazio trasformato, $\overline{x}_{PCA}$ la media delle istanze nello spazio trasformato dalla PCA, ed N la dimensione del dataset. É facile accorgersi dunque che il rapporto tra la devianza pre-PCA e la stessa quantità calcolata dopo la PCA sia pari alla varianza che si è scelto di preservare a monte, scegliendo un numero di componenti pari a 6. Infatti

$$\frac{D_{post-PCA}}{D_{tot}} = 0.9489805844109787 \simeq 95\% \tag{2.4}$$

Devianza post-clustering

Oltre a ciò, anche le operazioni di clustering hanno modificato la devianza dei dati. Per i cluster però vanno considerati due contributi di devianza: quella *intra-cluster* e quella *inter-cluster*, che identificano rispettivamente la devianza tra i dati di uno stesso cluster e tra i dati appartenenti a cluster differenti. La devianza totale dei dati dopo la clusterizzazione è calcolabile attraverso la

somma di tali contributi. Si procede quindi ad identificare tali valori attraverso le seguenti formule:

$$D_{intra} = \sum_{k=1}^{K} \sum_{i=1}^{N_k} ||x_i - \overline{x}_k|| = 4665.669591338057 \tag{2.5}$$

$$D_{inter} = \sum_{k=1}^{K} N_k ||\overline{x}_k - \overline{x}|| = 43732.340213622236 \tag{2.6}$$

dove:

- K è il numero di cluster;
- N_k è il numero di istanze che appartengono al cluster k;
- x_i è l'istanza i-esima del dataset;
- $\overline{x}_k$ è l'istanza centroide del cluster k;
- $\overline{x}$ è la media di tutte le istanze del dataset.

La somma di tali contributi, come ci si aspettava, sarà pari alla devianza preservata dopo l'analisi PCA.

Perdita di devianza

Dopo tali calcoli, ci si chiede a quanto ammonta la devianza persa durante le operazioni di PCA e clustering attuate. Tale risultato è calcolabile attraverso due metodi equivalenti:

$$\%D_{loss} = (1 - \%D_{post-PCA}) + \%D_{intra} * \%D_{post-PCA} = 0.1425031330662373 \tag{2.7}$$

$$\%D_{loss} = 1 - \%D_{inter} * \%D_{post-PCA} = 0.1425031330662373 \tag{2.8}$$

In conclusione si ha che, a valle delle operazioni di clustering e di analisi delle componenti principali, si ottiene un workload che esprime circa il 14% di devianza in meno del workload reale.

2.3 Analisi delle devianze

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

2.3.1 Devianza persa vs Numero di componenti principali

Per valutare la devianza persa e la variazione di quest'ultima all'aumentare del numero di componenti principali, basta ricalcolare le devianze per diversi numeri di componenti principali e di cluster. Per tale analisi si è scelto di andare a considerare un intorno di 6 componenti principali, in particolare da 3 a 7, e di eseguire i suddetti calcoli fino ad un massimo di 20 cluster, valore suggerito dallo strumento JMP come valore ideale di cluster. Tale analisi ha prodotto il seguente grafico.

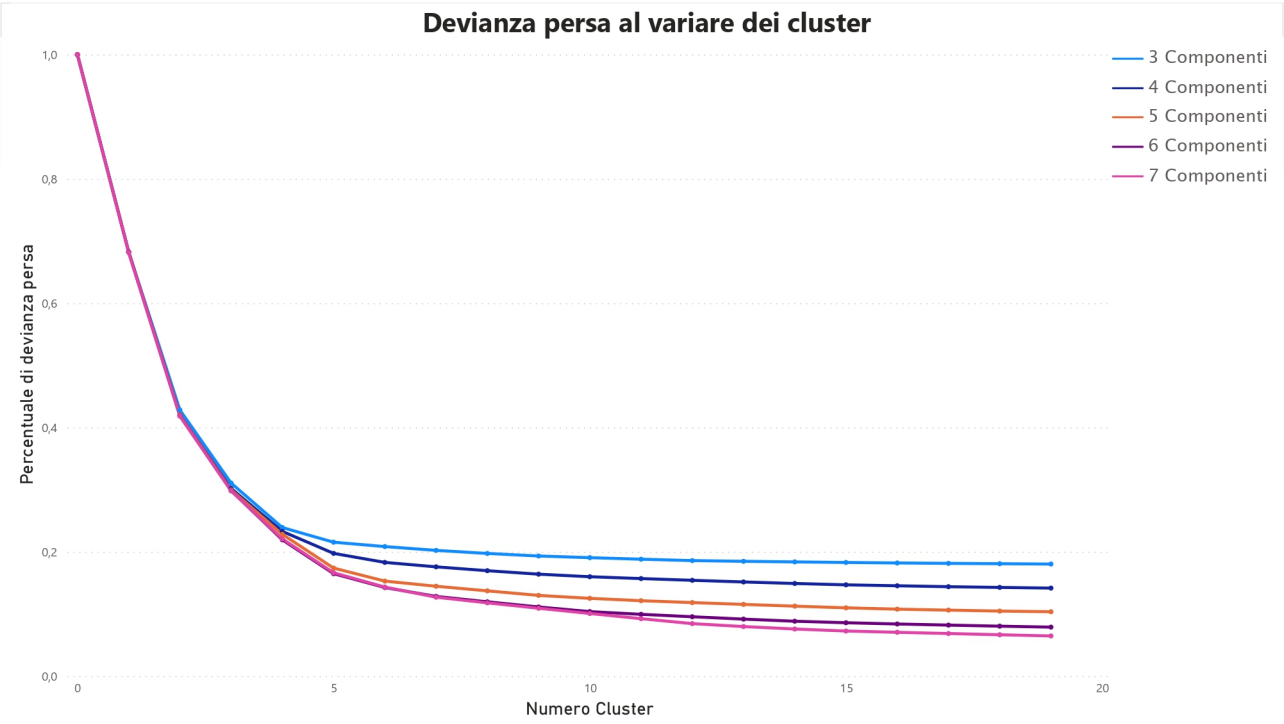


Figura 2.12: Devianza persa al variare del numero di cluster

Da tale analisi si nota che l'aumento sia di numero di cluster che si numero di componenti principali porta ad una minore percentuale di devianza persa. Un valore ottimale di cluster è quindi in corrispondenza del gomito di tali curve, e come è possibile notare, il gomito si ha come visto intorno al cluster 7. Il valore 7 è un valore limite di cluster: si nota che se si considerasse un valore di cluster precedente a 7 non si avrebbe alcuna differenza tra i valori considerati di componenti principali, mentre oltre ad un valore di cluster pari a 6 la devianza persa non decresce ulteriormente e il suo valore dipenderebbe solo dal numero di componenti principali considerate, visto l'andamento lineare che si ottiene dal grafico. Scelti 7 cluster, visto che non comporta una modifica sostanziale in termini di devianza, si ha un valore ottimo di componenti pari a 6, visto che 7 componenti darebbero la stessa devianza persa ma comporterebbero costi maggiori in termini di analisi.

2.3.2 Devianza persa vs Dimensione del Workload

Un'ulteriore osservazione che è possibile fare è riguardo la dimensione del WL sintetico che deriva dall'analisi. Per dimensione di workload sintetico, per questa analisi, si intende il rapporto tra numero di valori del workload reale e numero di valori del workload sintetico. Tale rapporto assume valore massimo pari ad 1 quando il workload sintetico ha la stessa dimensione del workload reale, e valore minimo quanto il workload sintetico ha dimensione unitaria, ovvero nel caso in cui si consideri un unico cluster. Idealmente, si vorrebbe che sia la devianza persa che la dimensione del workload sintetico siano quanto più bassa possibili. Tuttavia, la devianza decresce con l'aumentare del numero di clusters (e di componenti principali) mentre la dimensione del workload sintetico aumenta con l'aumentare degli stessi parametri. Per questo motivo, è possibile considerare il prodotto tra devianza persa e dimensione del workload sintetico ed osservare quali sono i valori di cluster e componenti principali che minimizzano tale quantità. Graficando i vari contributi si è ottenuto in grafico seguente.

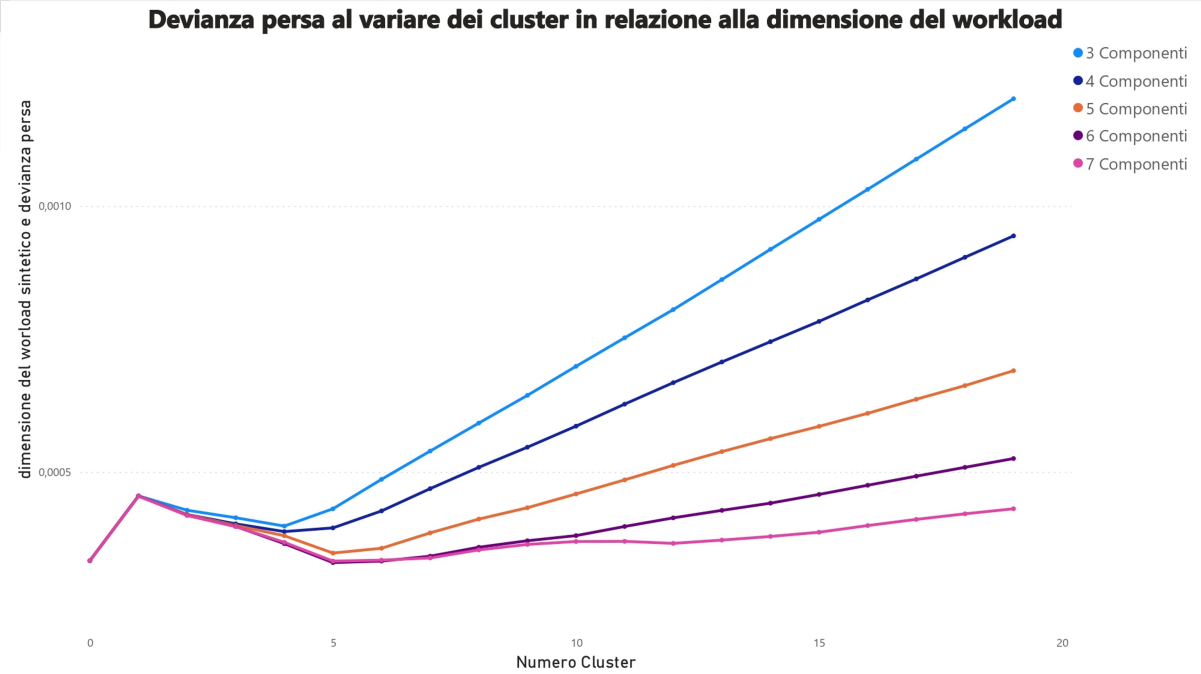


Figura 2.13: Dimensione del workload al variare del numero di cluster

Tale risultato sta a significare che una scelta elevata di cluster porterebbe ad un dimensionamento elevato del workload sintetico, con una conseguente scarsa riduzione di dimensionalità, che è l'obiettivo di tale analisi. La scelta fatta di considerare un numero di cluster pari a 7 risulta valida, in quanto si ottiene il minimo della dimensionalità del workload, senza però avere una perdita eccessiva di devianza.

2.4 Caratterizzazione del workload sintetico

A valle di tutte le operazioni di analisi del dataset, è ora necessario identificare e caratterizzare il workload sintetico attraverso delle istanze caratterizzanti del workload reale. Tale caratterizzazione avviene o scegliendo da ogni cluster un elemento randomico per andare a caratterizzarne le istanze, o, come avverrà in questa analisi, andare ad identificare il centroide reale del cluster, cioè l'istanza che di trova più vicino al centroide virtuale calcolato durante la fase di definizione delle devianze. Nello spazio trasformato dalle operazioni di PCA, l'analisi dei 7 cluster scelti ha portato all'individuazione di 7 centroidi virtuali, che sono riportati in tabella.

cluster	componente 1	componente 2	componente 3	componente 4	componente 5	componente 6
1	-5.224777	2.626848	-1.271459	-0.602047	0.588452	0.822892
2	-1.960935	0.353338	-0.374679	0.342625	-0.191774	-0.540921
3	-0.891292	14.903990	18.274293	-5.846637	-7.875777	-0.793595
4	11.467000	7.082632	-2.823275	0.225687	0.256265	-0.228367
5	1.256075	-1.007140	0.555972	-0.586979	0.384371	-0.048770
6	-6.064716	25.356980	49.559103	24.901215	25.376207	1.134095
7	0.944036	-0.839495	0.169095	0.577594	-0.465357	0.348614

Tabella 2.1: Tabella dei centroidi virtuali dei cluster

Una riflessione che si può fare è per il cluster 7 ad esempio, contenendo esso un unico elemento, non ha senso effettuare un analisi dei centroidi, ma l'istanza da considerare è banalmente l'unica contenuta nel cluster. Per gli altri cluster tale analisi risulta necessaria. Si è quindi proceduto a calcolare la distanza euclidea di tutte le istanze dai centroidi dei cluster, per poi identificare l'istanza che risultasse più vicina. Il calcolo, eseguito tramite Excel, ha prodotto il seguente workload sintetico:

Cluster	1	2	3	4	5	6	7
VmPeak	153484	153484	194112	215352	172392	172388	170344
VmSize	151436	151436	188448	206136	170340	170340	170340
VmHWM	26612	26612	43804	61784	32868	32156	31144
VmRSS	26468	26472	38872	53204	30848	31912	31140
VmPTE	160	160	268	312	200	200	200
Threads	55	14	107	9	41	9	52
MemFree	5397504	5008324	4551620	4539220	4593744	4667952	4655664
Buffers	81168	83852	202380	204712	172688	100104	89480
Cached	435176	767660	1118268	1118500	1117616	1117036	1090572
Active	0	0	0	0	0	0	0
Inactive	167336	380596	499448	521348	440024	407636	541220
Dirty	430616	597076	912412	907380	933472	893716	773332
Writeback	5397504	5008324	4551620	4539220	4593744	4667952	4655664
AnonPages	8159224	8159224	8159224	8159224	8159224	8159224	8159224
Mapped	4764	108968	144	36	160	44	324400
Slab	0	0	0	0	0	0	1760
PageTables	81704	126120	91216	105544	83192	84212	134500
Committed_AS	24376	28324	23848	23848	23844	23740	29956
NumOfAllocFH	81332	92116	113712	110156	110520	108968	106788
proc-fd	7424	7984	7276	7320	7208	7208	8028
avgThroughput	302560	344944	337444	345724	314568	317332	365264
avgElapsed	1530	2040	1020	1530	1530	1020	2040
avgLatency	0	0	0	0	0	0	0
Errors	2	2	2	2	2	2	2

Tabella 2.2: Workload sintetico caratterizzato

Capitolo 3

Web Server

3.1 Traccia

Effettuare l'analisi prestazionale di un Web Server attraverso le seguenti fasi:

- *Capacity Test*, per valutare il throughput e i tempi di risposta al variare del carico;
- *Workload Characterization* dei parametri di alto e basso livello, verificando l'effettiva caratterizzazione del workload di basso livello;
- *Design of Experiment* per studiare l'influenza dei vari parametri sui tempi di risposta

3.2 Descrizione del sistema

La prima fase del processo di performance analysis consiste nel definire il sistema da analizzare. Il System Under Test (SUT) è un server Linux eseguito su una macchina virtuale Oracle VM Virtual Box, le cui caratteristiche sono riportate in tabella.

	Specifica Server
CPU	AMD Ryzen 5 4600H
Core	2
RAM	6 GB
SO	Ubuntu 20.04 64-bit

In particolare si vuole studiare il comportamento di un Web Server *Apache 2.4*, che rappresenta il Component Under Test (CUT). Tale web server espone delle risorse ad un generico client, il quale vi accede tramite richieste HTTP. Nello specifico, si è scelto di utilizzare delle risorse differenziate, sull'ordine dei MB, eterogenee per tipologia di risorsa, generate tramite un sito di creazione di contenuti a dimensione fissata. Le risorse sono riportate nella tabella seguente.

Risorsa	Tipologia	Dimensione
4k.jpg	Multimedia	4 KB
2m.txt	Testuale	2 MB
3m.txt	Testuale	6 MB
4m.txt	Testuale	4 MB
5m.txt	Testuale	5 MB
6m.txt	Testuale	6 MB
7m.jpg	Multimedia	7 MB
8m.txt	Testuale	8 MB
9m.txt	Testuale	9 MB
10m.txt	Testuale	10 MB

Le richieste HTTP sono state generate da una macchina client avente le caratteristiche descritte in tabella, utilizzando l'applicativo *Apache JMeter*.

	Specifica Client
CPU	AMD Ryzen 5 4600H
Core	6
RAM	16 GB
SO	Windows 10 Home 64-bit

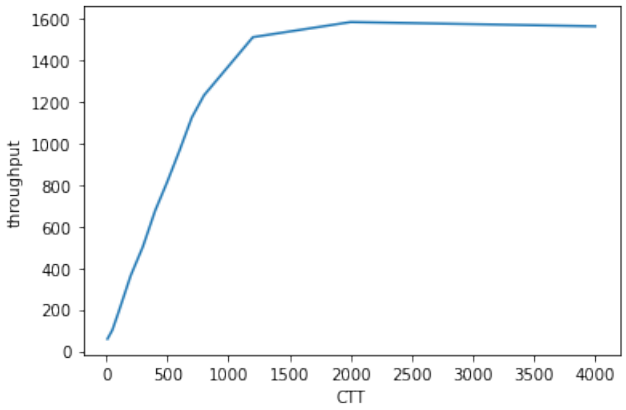
3.3 Capacity Test

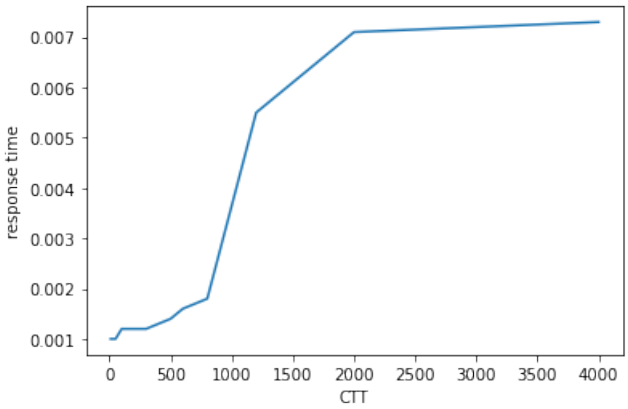
La fase di Capacity Test è volta a studiare il comportamento del sistema al crescere del tasso di richieste al minuto fatte dal client verso il server. Per tale analisi si è deciso di effettuare il test sfruttando pagine di dimensioni miste. In questo modo è possibile simulare il comportamento del sistema nel caso medio di utilizzo.
Per tale test si è scelto di considerare le pagine web descritte in precedenza, facendo diversi test della durata di 1 minuto ciascuno, effettuando 3 ripetizioni per ogni valore fissato di throughput. Si è scelto di considerare i seguenti valori di CTT: 200, 500, 600, 800, 1500, 2000 e 4000. In più si è deciso di analizzare un valore di CTT estremo, pari a 100000, per controllare la caratteristiche di funzionamento del sistema anche in casi irragionevoli. Lo scopo del Capacity Test e quello di definire alcuni parametri che caratterizzano il Server:

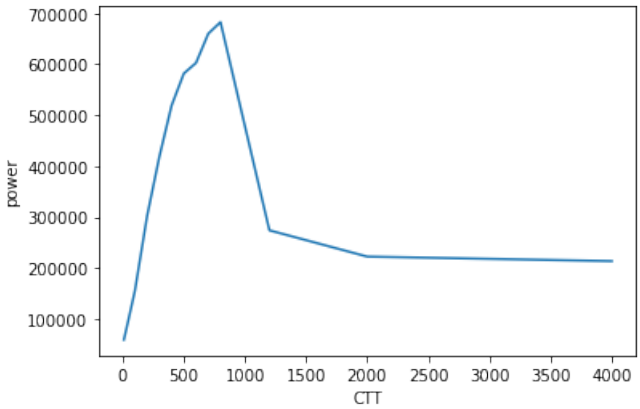
- **Knee Capacity**: punto in cui si registra il massimo throughput con un tempo di risposta ancora accettabile;
- **Usable Capacity**: punto oltre il quale il tempo di risposta aumenta rapidamente in funzione del carico ma il guadagno del throughput è basso.

Tra un esperimento e un altro viene resettato il server al fine di evitare fenomeni di accumulo e caching che possano falsare i risultati. Il numero di richieste al minuto utilizzato per i test

è variabile rispetto alla grandezza della pagina considerata e specificato in ogni test, in modo da raggiungere la saturazione della macchina. A valle dell'esecuzione di tali test, sono emersi i seguenti grafici:







Da tali grafici si identificano i punti di Knee Capacity in corrispondenza del picco della potenza, pari a 800 req/min, mentre la Usable Capacity è individuata in corrispondenza del valore di 1500 req/min.

3.3.1 Approfondimento

Per analizzare in modo più approfondito i tempi di risposta, sono stati graficati i valori di elapsed time rispetto al timestamp delle misurazioni per ogni CTT utilizzato. In questi grafici si è notato un fenomeno interessante. In particolare, con un numero di utenti pari a 30, i CTT inferiori a 1500 pare non seguano un andamento definito, ma che, ricevuta la richiesta, il server provi a soddisfarla il prima possibile, come si può verificare dal tale grafico.

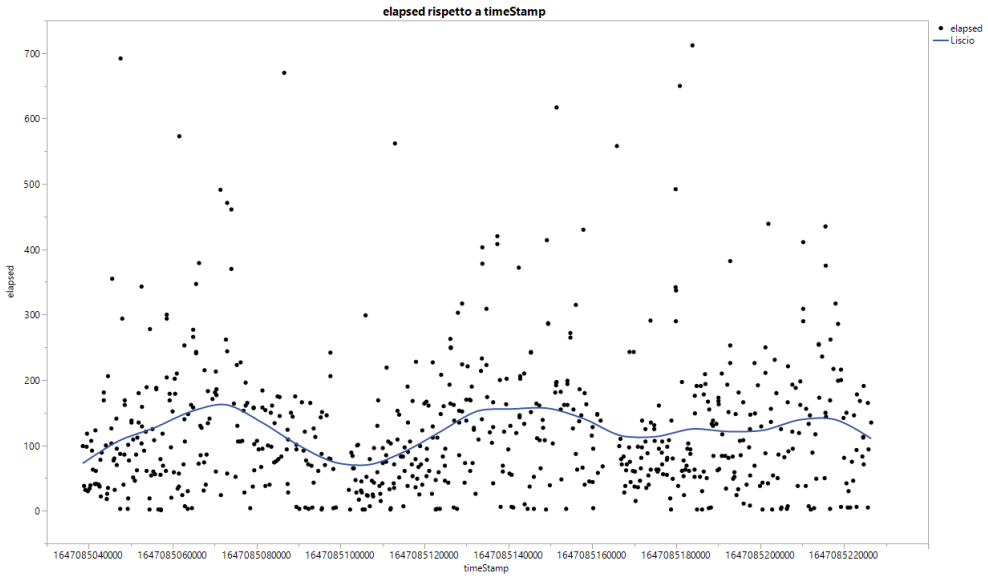


Figura 3.1: Grafico dei tempi a CTT pari a 200 Req/min

A valori di CTT bassi non è possibile distinguere delle fasce evidenti dovute alla dimensione di tale risorsa, questo perchè le richieste vengono servite quasi allo stesso tempo. In generale, il non verificarsi di tale fenomeno è dovuto alla poca variabilità che si ha tra le risorse sul server utilizzato per l'analisi. É comunque possibile confermare che le richieste che hanno un alto valore di elapsed corrispondono a risorse di dimensione elevate.

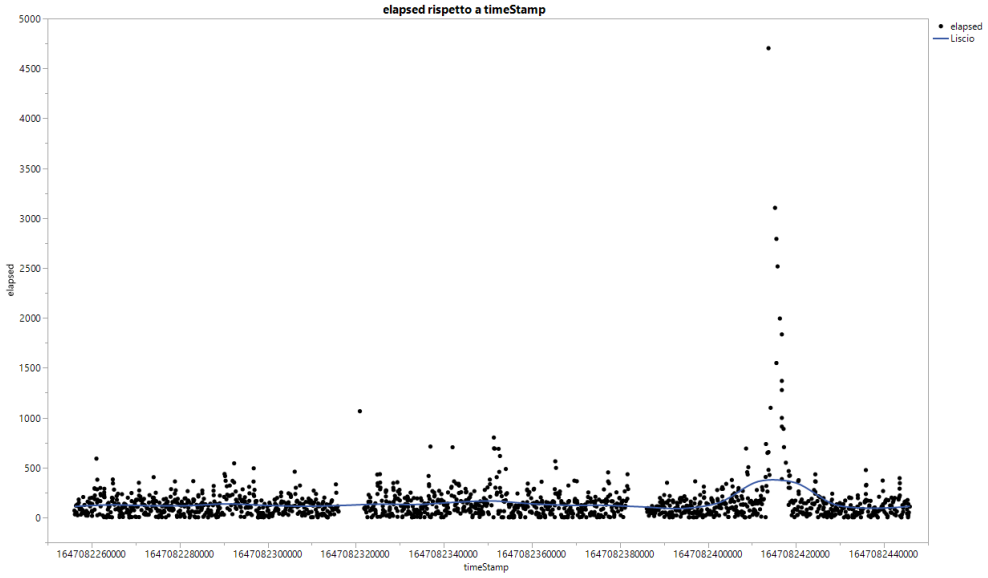


Figura 3.2: Grafico dei tempi a CTT pari a 500 Req/min

All'aumentare del valore di throughput si verifica una diluizione dei tempi di servizio, in particolare i tempi si alzano in maniera importante, e linearmente con il CTT.

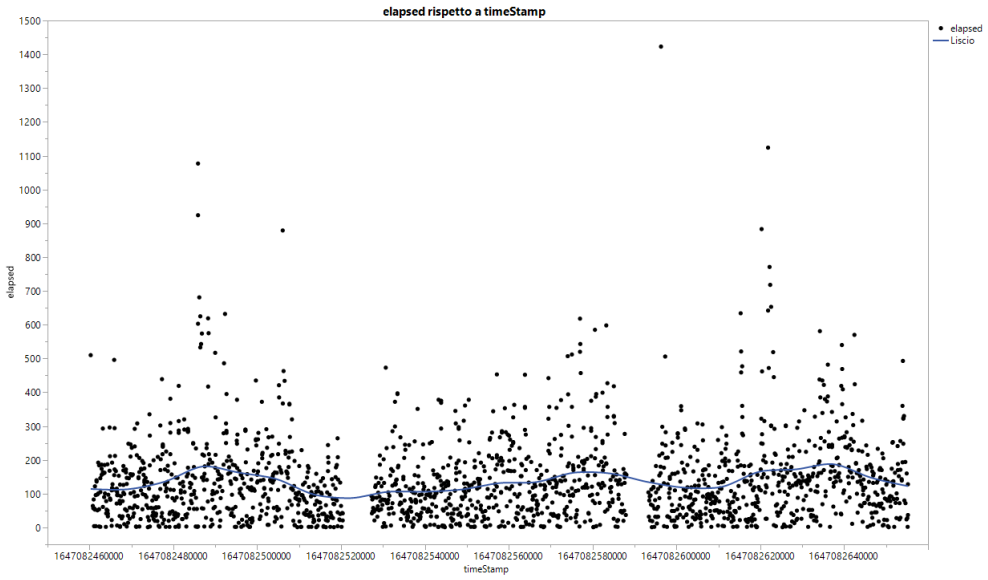


Figura 3.3: Grafico dei tempi a CTT pari a 600 Req/min

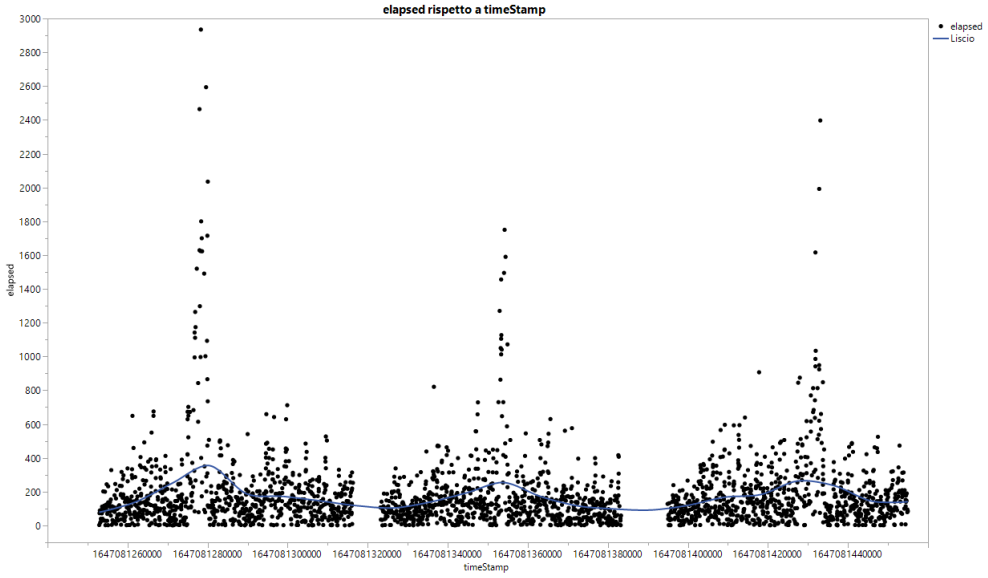


Figura 3.4: Grafico dei tempi a CTT pari a 800 Req/min

Superato il valore di Knee Capacity, si può notare un cambio di andamento, e in particolare si nota come ci sia un eccessivo accumulo di richieste a tempi di risposta bassi per timestamp bassi, per poi aumentare al crescere dell'istante temporale. Questo andamento è probabilmente dovuto a fenomeni di accumulo che avvengono o lato server o lato client.

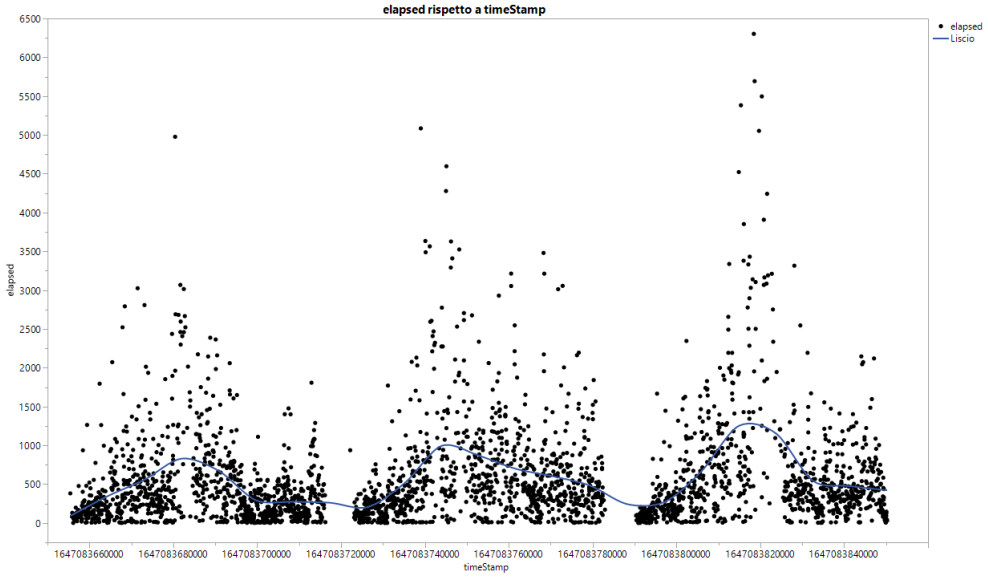


Figura 3.5: Grafico dei tempi a CTT pari a 1500 Req/min

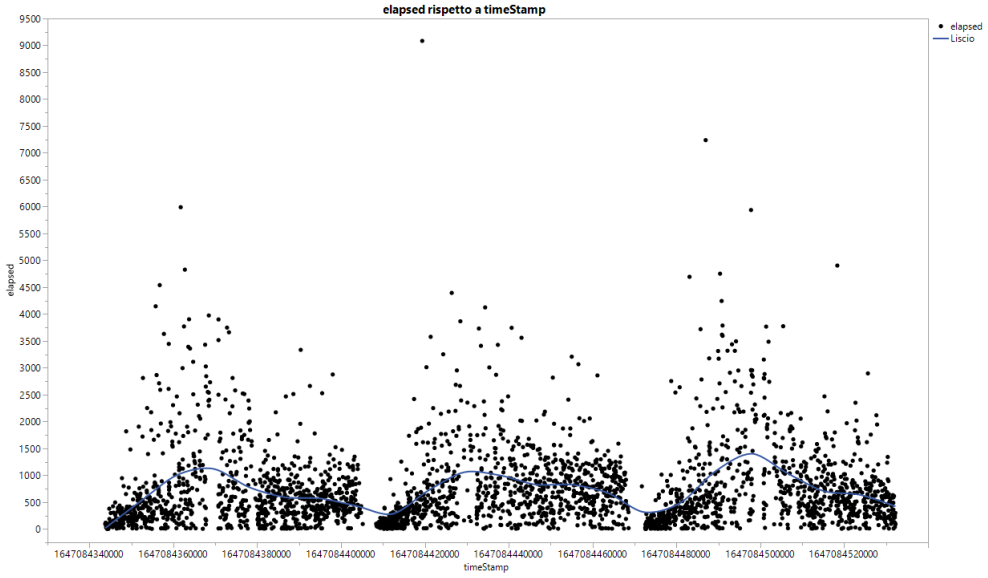


Figura 3.6: Grafico dei tempi a CTT pari a 2000 Req/min

Infine, si è deciso di analizzare l'andamento ad un CTT estremo, pari a 100000, per verificare se ci siano problemi da segnalare, eseguendo un unica ripetizione per tale valore. A valle di tale analisi ci si accorge che i tempi di risposta rispetto a CTT più bassi decrescono. Questa crescita è probabilmente dovuta al fatto che il client si accorge di non essere in grado di servire così tante richieste e riduce il carico. Probabilmente però tale fenomeno potrebbe essere dovuto anche a procedure di scheduling adottate lato server per ottimizzare le richieste da servire.

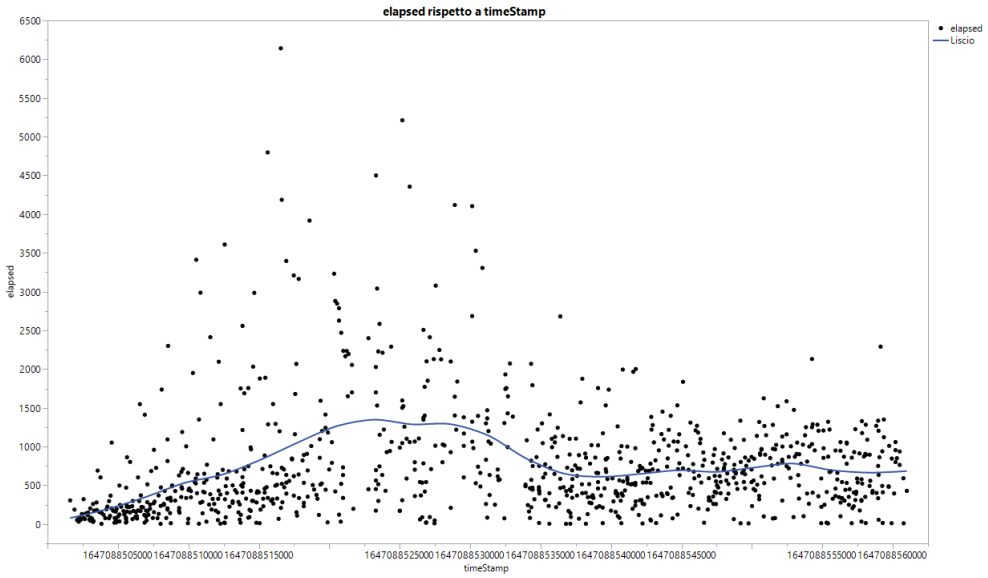


Figura 3.7: Grafico dei tempi a CTT pari a 100000 Req/min

3.4 Workload Characterization

Il processo di caratterizzazione del workload è volto a generare un workload sintetico partendo da uno reale. Sono state eseguite quindi le seguenti fasi:

- Creazione del workload reale:

  di alto livello (indicato con HL High Level)

  di basso livello (indicato con LL Low Level)

- Creazione del workload sintetico:

  caratterizzazione del workload HL (indicato con HL_c)

  caratterizzazione del workload LL (indicato con LL_c)

- Applicazione del workload di alto livello caratterizzato:

  raccolta dei parametri di basso livello (indicato con LL')

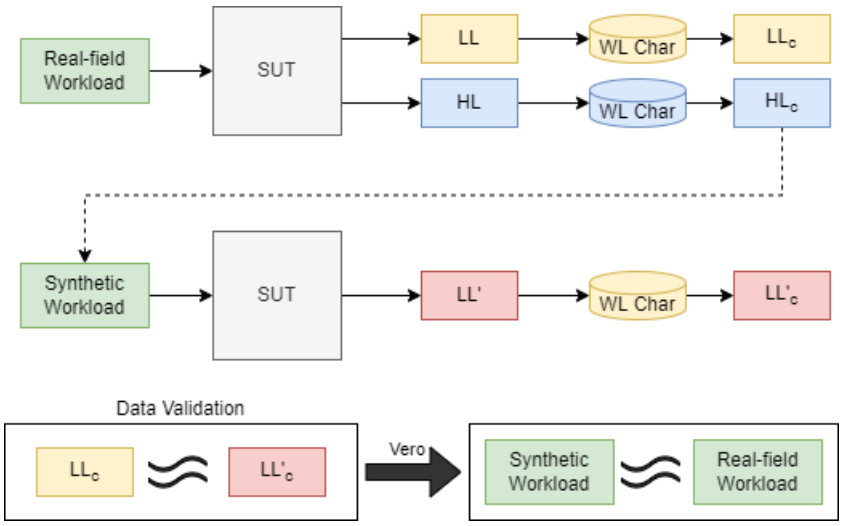
- Creazione del workload reale di basso livello LL'

  caratterizzazione del workload LL' (indicato con LL'_c)

- Verifica della bontà della caratterizzazione del workload di alto livello:

  tramite test di ipotesi dei parametri di basso livello caratterizzati LL_c e LL'_c

Le operazioni sono di seguito indicate in maniera schematica.



3.4.1 Workload di alto livello

Al fine di simulare l'imposizione di un carico reale al sistema si è cercato di diversificare quanto più possibile le richieste effettuate dal client. Per fare ciò, tramite JMeter si è deciso di creare 5

Thread Group, ognuno contenente 30 utenti, ognuno dei quali tramite un **Random Controller**, effettua delle **HTTP Request** randomiche alle risorse presenti sul server. I cinque Thread Group si differenziano sulla base del numero di richieste al minuto effettuate dal singolo utente, imposto tramite un **Constant Throughput Timer**. La configurazione dei singoli Thread Group è riportata in tabella.

Gruppo	Utenti	Throughput
Thread Group 1	30	100 rq/min
Thread Group 2	30	500 rq/min
Thread Group 3	30	800 rq/min
Thread Group 4	30	1200 rq/min
Thread Group 5	30	1500 rq/min

Definita la configurazione delle richieste lato client, è stato avviato il test, per la durata di 5 minuti, facendo lavorare i thread group in maniera sequenziale per un minuto ciascuno.

Raccolta dei parametri

Tramite JMeter, all'atto dell'esecuzione del test si è scelto di raccogliere i parametri di alto livello sfruttando il componente **Simple Data Writer**, che crea un file di output contenente dei parametri specificati dall'utente. Si è scelto di usare tale componente rispetto ad altri che potevano fornire informazioni maggiori, perchè è poco oneroso in termini di memoria lato client. Si è scelto la seguente configurazione di parametri da salvare:

- *timestamp*: istante temporale UNIX a cui la richiesta viene effettuata;
- *label*: etichetta testuale della richiesta effettuata;
- *elapsed*: tempo trascorso tra l'invio della richiesta del client e la ricezione dell'ultimo pacchetto della risposta del server;
- *ThreadName*: nome del Thread Group a cui appartiene il thread che ha effettuato la richiesta;
- *Success*: esito della richiesta;
- *bytes*: bytes ricevuti dal client;
- *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;

- *Connect*: tempo necessario per instaurare la connessione con il server.

Tramite la seguente raccolta, si è ottenuto un workload reale con 2560 istante e 12 componenti. Per completezza se ne riporta un estratto.

timeStamp	**elapsed**	**label**	**threadName**	...	**success**	**URL**	**Latency**	**Connect**
1647106144906	5	HTTP Request1	Thread group 1-1	...	true	http://192.168.56.101/4k.jpg	4	1
1647106145037	14	HTTP Request1	Thread group 1-5	...	true	http://192.168.56.101/4k.jpg	12	5
1647106145068	155	HTTP Request2	Thread group 1-6	...	true	http://192.168.56.101/2m.txt	12	3
1647106144969	483	HTTP Request4	Thread group 1-3	...	true	http://192.168.56.101/4m.txt	5	2
1647106144937	615	HTTP Request10	Thread group 1-2	...	true	http://192.168.56.101/10m.txt	5	3

Tabella 3.1: Workload reale

3.4.2 Workload di basso livello

Raccolta dei parametri

Per raccogliere i dati di basso livello ci si è avvalsi del comando UNIX

```
vmstat 1 -n -w > WL_LowLevel.csv
```

per la raccolta delle statistiche della macchina virtuale relative differenti aree della macchina. In particolare il dataset di basso livello si compone dei seguenti valori:

- **Processi**: "*r*" indica i processi in esecuzione, mentre "*b*" il numero di processi bloccati;
- **Memoria**: "*swpd*" indica il totale della memoria virtuale usata, "*free*" il totale di quella libera, mentre "*buff*" il totale usato come buffer;
- **Swap**: "*si*" rappresenta il totale della memoria su cui il disco ha effettuato degli swap, mentre "*so*" rappresenta la memoria su cui è stato effettuato uno swap nel disco;
- **I-O**: "*bi*" rappresenta la quantità di blocchi inviati ad un dispositivo, mentre "*bo*" quelli ricevuti.
- **Sistema**: "*in*" indica il numero di interrupt al secondo, mentre "*cs*" il numero di contest-switch effettuati al secondo;
- **CPU**: "*us*" è la percentuale di utilizzo CPU da parte di processi in userspace, "*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, e infine "*st*" indica il tempo di CPU occupato da un altra macchina virtuale.

Contemporaneamente all'invio delle richieste definite per la raccolta del workload reale di alto livello, quindi, si è provveduto a generare il workload di basso livello, di cui ne è riportato un estratto.

Proc		Memory				Swap		I-O		System		CPU				
r	**b**	**swpd**	**free**	**buff**	**cache**	**si**	**so**	**bi**	**bo**	**in**	**cs**	**us**	**sy**	**id**	**wa**	**st**
0	0	0	3763000	46832	1378704	0	0	467	76	140	289	2	5	93	0	0
0	0	0	3762992	46832	1378704	0	0	0	0	298	487	1	1	98	0	0
1	0	0	3749472	46836	1387800	0	0	9220	0	549	492	0	29	67	4	0
1	0	0	3732248	46840	1398528	0	0	10576	0	1275	1011	1	46	49	5	0
2	0	0	3722232	46840	1409040	0	0	10496	48	1447	1136	2	63	29	7	0
1	0	0	3714328	46840	1417704	0	0	8640	0	1165	984	1	49	47	3	0
3	0	0	3670252	47684	1434916	0	0	0	12	1671	691	0	61	10	29	0
5	0	0	3667852	47588	1434796	0	0	0	0	1727	557	1	50	50	0	0
1	0	0	3714328	46840	1417704	0	0	8640	0	1165	984	1	49	47	3	0
2	0	0	3686132	47372	1434596	0	0	0	800	1001	428	0	38	62	0	0
0	0	0	3762992	46832	1378704	0	0	0	0	298	487	1	1	98	0	0
0	0	0	3706344	46856	1434120	0	0	0	16	116	166	0	0	100	0	0

3.4.3 Caratterizzazione dei workload

Per effettuare la caratterizzazione dei workload, sia di alto che di basso livello, sono state effettuate le seguenti fasi:

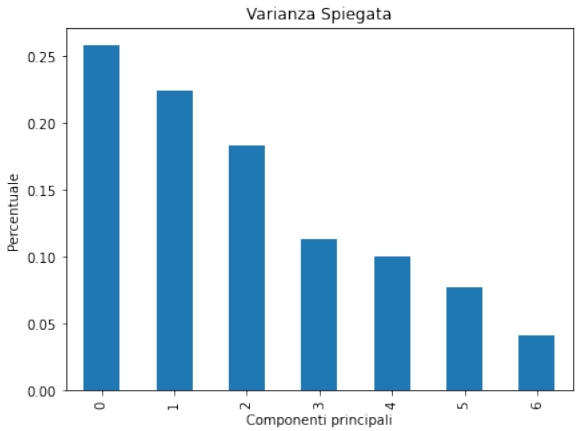
- Preprocessing: eliminazione di eventuali features nominali, costanti o perfettamente correlate;
- Trade-off: individuazione del miglior numero di componenti principali, in funzione del numero di cluster creati;
- PCA e Clustering;
- Creazione del workload sintetico: selezione dell'istanza da ognuno dei cluster.

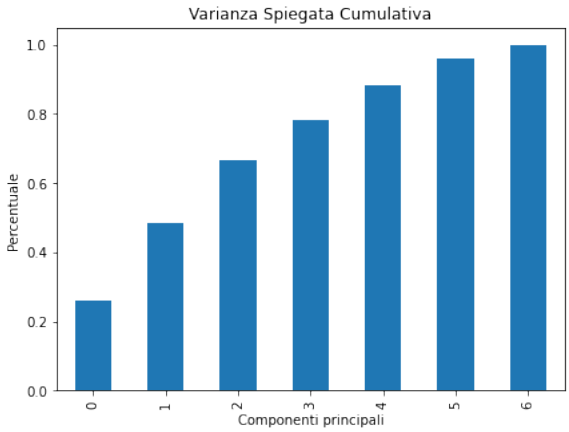
Workload di alto livello

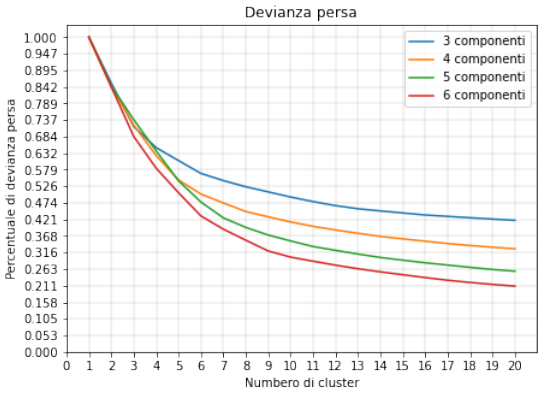
Secondo le operazioni precedentemente descritte, è stata applicata la PCA al dataset costituito dai parametri di alto livello . Successivamente, è stato applicato il metodo di Ward per clusterizzare il dataset utilizzando le prime 6 componenti principali, selezionando 6 cluster. In figura è possibile osservare l'andamento della devianza persa in funzione del numero di cluster, al variare delle componenti principali. Con i parametri scelti (6 componenti principali e 6 cluster) viene preservato circa il 96% di varianza, ammettendo una perdita di devianza di circa l'11.2% dopo le operazioni di PCA e Clustering.

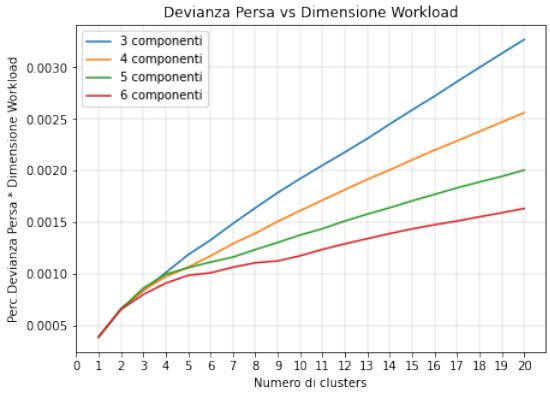
A valle dell'analisi effettuata, si è scelto di considerare 6 componenti principali e 6 cluster, avendo una dimensionalità del workload sintetico ridotta.

L'estrazione delle istanze caratterizzanti non è stata casuale, ma si è ragionato sulle composizioni dei cluster. In particolare, per ogni cluster è stata individuata la tipologia di risorsa più frequente. Successivamente è stata estratta la richiesta associata alla risorsa più frequente della









dev_tot	dev_postPCA	dev_intra_cluster	dev_inter_cluster	dev_tot_persa
17913.0	0.958985	0.073982	0.824619	0.111963

Tabella 3.2: Analisi delle devianze

data tipologia. Individuata la richiesta si è tenuto conto anche del Thread Group a cui appartiene il thread che l'ha generata. In tabella è possibile vedere il conteggio delle istanze raggruppate per risorsa e per cluster.

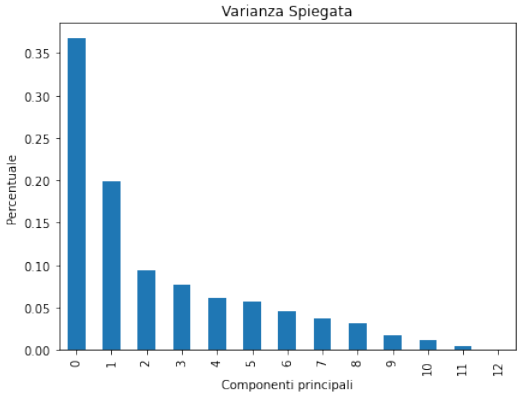
Cluster	URL	Conteggio
1	http://192.168.56.101/2m.txt	1
	http://192.168.56.101/3m.txt	2
	http://192.168.56.101/4k.jpg	1
	http://192.168.56.101/4m.txt	8
	http://192.168.56.101/6m.txt	232
	http://192.168.56.101/8m.txt	222
	http://192.168.56.101/9m.txt	224
2	http://192.168.56.101/2m.txt	264
	http://192.168.56.101/3m.txt	271
	http://192.168.56.101/4k.jpg	209
	http://192.168.56.101/4m.txt	205
	http://192.168.56.101/5mb.txt	1
	http://192.168.56.101/6m.txt	2
3	http://192.168.56.101/10m.txt	5
	http://192.168.56.101/2m.txt	18
	http://192.168.56.101/3m.txt	17
	http://192.168.56.101/4k.jpg	11
	http://192.168.56.101/4m.txt	29
	http://192.168.56.101/5mb.txt	16
	http://192.168.56.101/6m.txt	23
	http://192.168.56.101/7mb.jpg	21
	http://192.168.56.101/8m.txt	25
	http://192.168.56.101/9m.txt	24
4	http://192.168.56.101/2m.txt	2
	http://192.168.56.101/3m.txt	1
	http://192.168.56.101/4k.jpg	2
	http://192.168.56.101/5mb.txt	1
	http://192.168.56.101/6m.txt	1
	http://192.168.56.101/8m.txt	3
5	http://192.168.56.101/2m.txt	1
	http://192.168.56.101/3m.txt	2
	http://192.168.56.101/4k.jpg	2
	http://192.168.56.101/4m.txt	1
	http://192.168.56.101/5mb.txt	1
	http://192.168.56.101/6m.txt	1
	http://192.168.56.101/7mb.jpg	4
	http://192.168.56.101/8m.txt	4
6	http://192.168.56.101/10m.txt	229
	http://192.168.56.101/3m.txt	1
	http://192.168.56.101/5mb.txt	213
	http://192.168.56.101/6m.txt	1
	http://192.168.56.101/7mb.jpg	256

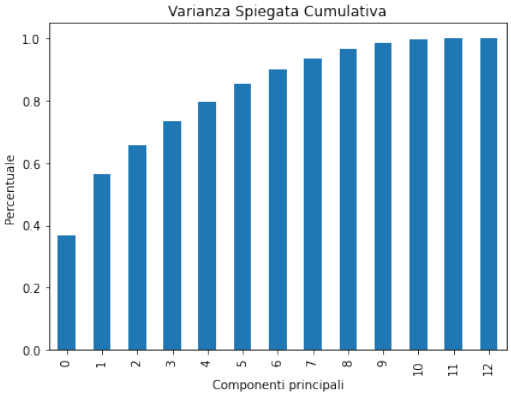
Si ottiene quindi il workload HL caratterizzato, HL_c, di cui se ne riporta una versione con i nomi delle distribuzioni abbreviati per questioni di visibilità all'interno del documento.

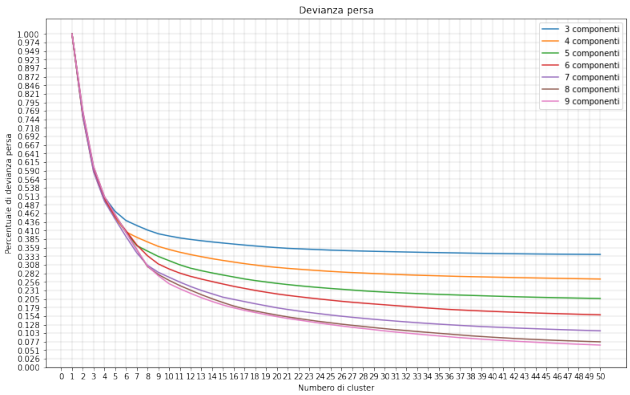
timeStamp	elapsed	label	threadName	success	bytes	sentBts	allThds	URL	Latency	Cnct	Clst
1647106178100	235	Request9	Thd grp 1-25	VERO	9437500	125	14	/9m.txt	7	3	1
1647106212165	35	Request1	Thd grp 2-17	VERO	5369	125	24	/4k.jpg	35	0	2
1647106163289	4678	Request6	Thd grp 1-26	VERO	6291772	125	18	/6m.txt	18	8	3
1647106386982	1772	Request2	Thd grp 5-27	VERO	2097468	125	30	/2m.txt	71	35	4
1647106270123	3516	Request5	Thd grp 3-3	VERO	5243195	126	19	/5mb.txt	488	0	5
1647106227329	663	Request10	Thd grp 2-11	VERO	10486076	126	18	/10m.txt	12	0	6

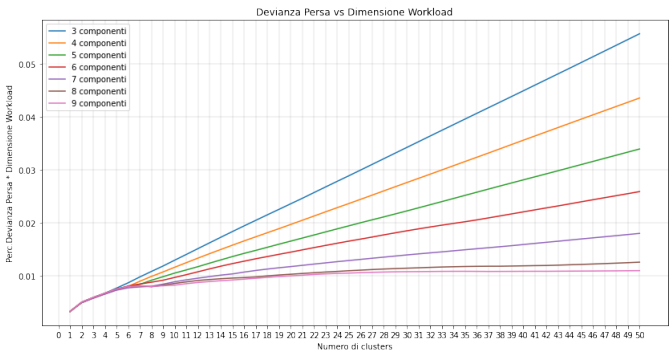
Workload di basso livello

Come come è stato fatto per il workload di alto livello, si ripetono le operazioni per il workload di basso livello, LL, per ottenere il workload caratterizzato LL_c. É stata quindi computata la PCA e successivamente, è stato applicato il metodo di Ward per clusterizzare il dataset costituito dalle prime 8 componenti principali, selezionando 35 cluster, avendo preservata una varianza pari al 95% del dataset originale. In figura è possibile osservare l'andamento della devianza persa in funzione del numero di cluster, al variare delle componenti principali. Con i parametri scelti (8 componenti principali e 35 cluster) viene preservato circa il 90% di devianza.









A valle dell'analisi effettuata, in tabella sono indicate le devianze analizzate durante le fasi di PCA e Clustering.

dev_tot	dev_postPCA	dev_intra_cluster	dev_inter_cluster	dev_tot_persa
3952.0	0.975588	0.062289	0.956505	0.102285

Si ottiene quindi il workload caratterizzato LL_c andando a selezionare in maniera casuale le istanze all'interno dei cluster.

r	b	swpd	free	buff	cache	si	so	bi	bo	in	cs	us	sy	id	wa	st	Cluster
5	0	0	3687904	47500	1434700	0	0	0	0	915	422	0	30	70	0	0	1
2	0	0	3698224	46856	1434128	0	0	0	0	1822	753	0	55	45	0	0	2
0	0	0	3685080	47524	1434728	0	0	0	0	774	520	0	22	78	0	0	3
1	0	0	3695788	47452	1434644	0	0	0	24	1305	553	0	41	58	0	0	4
1	0	0	3671692	47668	1434896	0	0	0	56	1040	447	0	37	63	0	0	5
0	0	0	3702824	46872	1434132	0	0	0	0	209	209	0	0	100	0	0	6
2	0	0	3682280	47548	1434748	0	0	0	0	1841	526	0	52	48	0	0	7
5	0	0	3667852	47588	1434796	0	0	0	0	1727	557	1	50	50	0	0	8
4	0	0	3670152	47652	1434880	0	0	0	0	1677	512	0	47	53	0	0	9
0	0	0	3701500	46848	1434124	0	0	128	0	727	520	0	20	80	0	0	10
2	1	0	3707352	47460	1434652	0	0	0	12	1379	460	0	33	66	0	0	11
2	0	0	3698840	46872	1434132	0	0	0	96	752	475	0	21	78	0	0	12
1	0	0	3694544	46888	1434136	0	0	0	0	1214	907	1	37	62	0	0	13
2	0	0	3674448	47492	1434696	0	0	0	0	1216	403	0	38	62	0	0	14
4	0	0	3684080	47564	1434768	0	0	0	0	1966	556	0	56	44	0	0	15
2	0	0	3670884	47668	1434888	0	0	0	12	1748	764	0	66	9	26	0	16
2	2	0	3681372	47440	1434628	0	0	0	8	1021	551	0	26	73	0	0	17
4	2	0	3672856	47568	1434772	0	0	0	8	1482	657	0	57	28	16	0	18
2	0	0	3722232	46840	1409040	0	0	10496	48	1447	1136	2	63	29	7	0	19
7	0	0	3688400	47484	1434684	0	0	0	0	1318	524	0	48	52	0	0	20
4	1	0	3681360	47556	1434756	0	0	0	12	1123	465	0	43	51	6	0	21
1	0	0	3684920	47412	1434620	0	0	0	0	916	364	0	25	75	0	0	22
5	1	0	3669220	47676	1434912	0	0	0	0	1756	762	0	65	3	31	0	23
1	0	0	3691708	47332	1434652	0	0	420	136	1699	1554	0	57	40	2	0	24
4	0	0	3709712	46848	1423596	0	0	6016	64	902	597	0	42	58	0	0	25
5	0	0	3686600	47348	1434576	0	0	0	0	1338	459	0	43	57	0	0	26
0	0	0	3695032	47420	1434624	0	0	0	0	472	237	0	12	88	0	0	27
1	0	0	3749472	46836	1387800	0	0	9220	0	549	492	0	29	67	4	0	28
1	0	0	3714328	46840	1417704	0	0	8640	0	1165	984	1	49	47	3	0	29
1	0	0	3673920	47644	1434872	0	0	0	0	836	384	1	30	69	0	0	30
7	0	0	3676520	47476	1434668	0	0	0	48	1193	440	1	40	59	0	0	31
4	0	0	3677432	47572	1434776	0	0	0	56	1207	618	1	47	40	13	0	32
5	0	0	3699288	47348	1434660	0	0	0	192	1013	633	0	34	66	0	0	33
4	2	0	3706380	46848	1428272	0	0	4612	0	1077	660	0	45	52	3	0	34
1	0	0	3732248	46840	1398528	0	0	10576	0	1275	1011	1	46	49	5	0	35

3.4.4 Applicazione del workload sintetico di alto livello

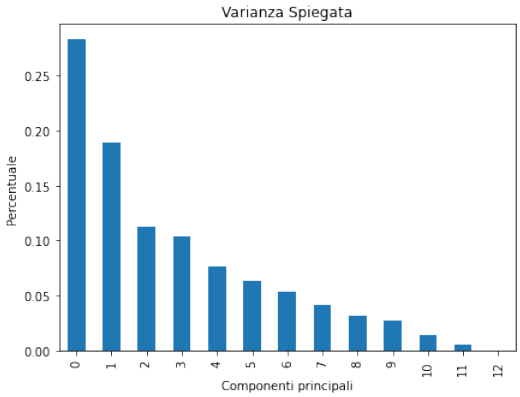
Dopo aver eseguito le operazioni di PCA e Clustering sul workload reale di alto livello, è necessario andare ad applicare il workload HL_c al web server, e raccoglierne nuovamente i parametri di basso livello, per effettuare un confronto statistico e validare la bontà del workload sintetico. Per farlo si è creato un nuovo **Test Plan** con le richieste HTTP contenute all'interno di HL_c da sottomettere al server. Tale workload si compone delle seguenti richieste.

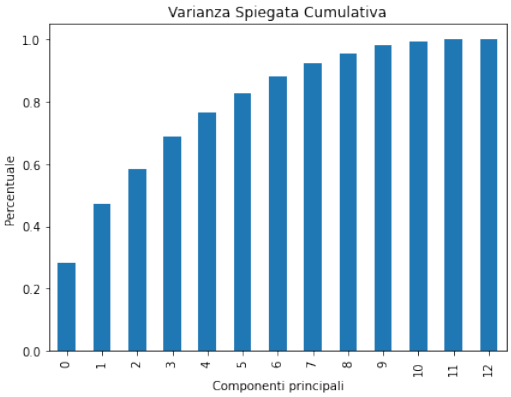
threadName	label	URL	Cluster
Thread group 1-25	HTTP Request9	/9m.txt	1
Thread group 2-17	HTTP Request1	/4k.jpg	2
Thread group 1-26	HTTP Request6	/6m.txt	3
Thread group 5-27	HTTP Request2	/2m.txt	4
Thread group 3-3	HTTP Request5	/5mb.txt	5
Thread group 2-11	HTTP Request10	/10m.txt	6

Quindi si è configurato il client in modo da generare le richieste sopracitate in Thread Group differenti, inviando richieste per circa 5 minuti. Contemporaneamente sono stati raccolti nuovamente i parametri di basso livello, indicato con LL'. Tale raccolta ha permesso di costruire un nuovo workload di basso livello, di cui se ne riporta un estratto.

Proc		**Memory**				**Swap**		**I-O**		**System**		**CPU**				
r	b	swpd	free	**buff**	**cache**	**si**	**so**	**bi**	**bo**	**in**	**cs**	**us**	**sy**	**id**	**wa**	**st**
2	0	0	3736176	46876	1394524	0	0	1162	343	249	501	4	9	86	1	0
0	0	0	3736176	46876	1394524	0	0	0	0	192	294	0	0	100	0	0
2	0	0	3722192	46876	1394524	0	0	0	0	1104	759	0	49	51	0	0
2	0	0	3720188	46876	1395808	0	0	1280	0	1508	735	0	54	45	0	0
1	0	0	3719416	46876	1397960	0	0	2176	0	706	518	0	42	58	0	0
3	0	0	3717952	46876	1400540	0	0	2560	0	1099	1674	3	40	56	1	0

Come fatto per il workload LL, si applicano le operazioni di PCA e Clustering allo stesso modo di come eseguito in precedenza, facendo attenzione a mantenere la stessa configurazione usata per LL, cioè conservando lo stesso numero di componenti principali e considerando lo stesso numero di cluster. Così facendo si preserva lo stesso quantitativo di varianza preservato nel workload LL_c: infatti, scegliendo 8 componenti si ha una varianza spiegata pari al 95% circa. In figura gli istogrammi relativi alle componenti principali del workload LL'.





Studiando quindi l'andamento della devianza durante le fasi del processo, si ottiene che a valle delle operazioni di PCA e Clustering si accetta una perdita di devianza pari all' 85% circa.

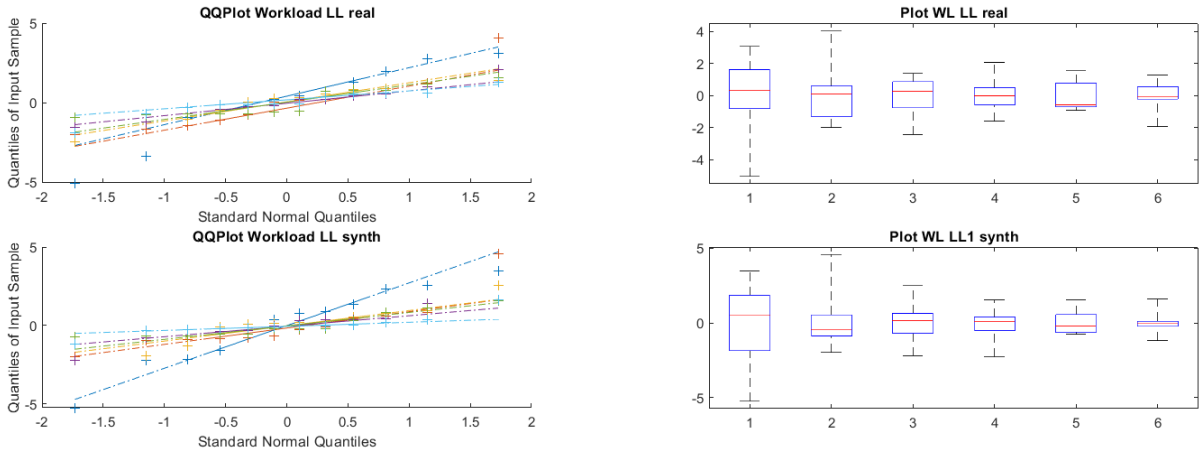
dev_tot	dev_postPCA	dev_intra_cluster	dev_inter_cluster	dev_tot_persa
3913.0	0.935302	0.093824	0.906176	0.152452

Come avvenuto per il workload LL_c, si è scelto di caratterizzare il workload LL' andando ad estrarre casualmente delle istanze dai cluster, ottenendo il workload LL'_c esplicitato in basso.

r	b	swpd	free	buff	cache	si	so	bi	bo	in	cs	us	sy	id	wa	st	Cluster
4	0	0	3689497	47252	1434643	0	0	0	0	1399	606	0	56	44	0	0	1
3	0	0	3699113	46924	1434330	0	0	0	0	1525	599	1	56	43	0	0	2
2	0	0	3720188	46876	1395808	0	0	1280	0	1508	735	0	54	45	0	0	3
2	0	0	3701388	46884	1434182	0	0	5504	0	741	481	0	42	58	1	0	4
5	0	0	3688757	47364	1434609	0	0	0	0	1406	552	1	66	33	0	0	5
0	0	0	3689699	47444	1434565	0	0	0	0	1140	468	0	43	56	0	0	6
7	1	0	3693361	47212	1434255	0	0	0	52	1515	483	0	65	19	16	0	7
2	0	0	3689818	47140	1434150	0	0	0	0	1481	528	1	59	40	0	0	8
1	0	0	3692549	47444	1434338	0	0	0	12	1605	502	0	66	27	7	0	9
4	0	0	3692148	47420	1434219	0	0	0	236	1548	633	0	60	39	1	0	10
2	0	0	3695638	47004	1434422	0	0	0	0	1268	461	0	52	48	0	0	11
4	2	0	3698988	47176	1434553	0	0	0	8	1734	723	0	60	29	11	0	12
2	0	0	3698933	47212	1434177	0	0	0	0	1233	513	0	53	34	13	0	13
1	0	0	3686857	47108	1434348	0	0	0	12	1269	626	0	50	50	0	0	14
2	0	0	3685917	46980	1434478	0	0	0	36	1539	495	0	62	30	7	0	15
6	0	0	3687560	47076	1434380	0	0	0	0	1340	463	0	58	40	2	0	16
1	0	0	3693144	47204	1434478	0	0	0	0	802	392	0	44	56	0	0	17
0	0	0	3736176	46876	1394524	0	0	0	0	192	294	0	0	100	0	0	18
4	0	0	3691066	47260	1434688	0	0	0	0	1718	483	0	67	20	12	0	19
2	0	0	3713048	46876	1404460	0	0	3968	0	1514	2649	8	51	41	1	0	20
4	1	0	3685711	46932	1434664	0	0	0	0	1067	417	0	51	45	4	0	21
3	0	0	3690230	47132	1434253	0	0	0	0	1033	390	0	50	50	0	0	22
6	0	0	3696649	47068	1434302	0	0	0	0	1779	1022	2	69	29	0	0	23
5	1	0	3692118	47260	1434530	0	0	0	12	1541	469	0	67	7	26	0	24
3	0	0	3689863	46884	1434221	0	0	0	0	647	475	0	19	81	0	0	25
3	0	0	3690227	47436	1434176	0	0	0	760	1471	456	0	71	28	0	0	26
1	0	0	3693219	46884	1434614	0	0	0	248	788	701	0	27	73	0	0	27
4	0	0	3694209	47020	1434494	0	0	0	160	1245	464	0	57	43	1	0	28
1	0	0	3693951	47388	1434636	0	0	0	0	1286	502	0	55	45	0	0	29
3	0	0	3717952	46876	1400540	0	0	2560	0	1099	1674	3	40	56	1	0	30
5	0	0	3693240	47356	1434585	0	0	37	272	1343	994	2	52	43	3	0	31
2	0	0	3736176	46876	1394524	0	0	1162	343	249	501	4	9	86	1	0	32
1	0	0	3719416	46876	1397960	0	0	2176	0	706	518	0	42	58	0	0	33
4	0	0	3687074	46948	1434238	0	0	0	20	1641	459	0	68	31	0	0	34
9	0	0	3698203	47164	1434420	0	0	0	0	1504	494	0	67	32	0	0	35

3.4.5 Validazione del workload sintetico di alto livello

Per validare il workload sintetico di alto livello (HL_c) ottenuto a partire da quello reale (HL), sono stati confrontati i due workload sintetici di basso livello, uno (LL_c) relativo all'applicazione del workload reale di alto livello, l'altro (LL'_c) relativo all'applicazione di quello sintetico di alto livello. Il confronto di LL_c e LL'_c è stato effettuato mediante un **test di ipotesi** sulla differenza delle medie, sfruttando uno script *Matlab*. Per prima cosa, per entrambi i campioni è stata verificata la normalità della distribuzione dei valori di ogni componente principale, tracciando un **QQ-plot** ed effettuando il **test parametrico di Shapiro-Wilk**.



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

	r	**b**	**swpd**	**free**	**buff**	**cache**	**si**	**so**	**bi**	**bo**	**in**	**cs**	**us**	**sy**	**id**	**wa**	**st**
Rank test	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

A valle di tale analisi è possibile quindi concludere che non esistono differenze significative tra i due workload sintetici di basso livello, per cui la caratterizzazione del workload di alto livello può essere ritenuta corretta.

3.5 Design Of Experiment

Scopo della fase di **DOE** (*Design Of Experiment*) è quello di riuscire ad ottenere un design tale da massimizzare le informazioni ottenute, minimizzando il numero di esperimenti necessari. Una corretta analisi degli esperimenti aiuta anche a separare gli effetti dei vari fattori che potrebbero influenzare le prestazioni, determinando se un fattore è significativo oppure se le differenze osservate dipendono semplicemente da variazioni casuali dovute ad errori di misura e/o parametri non controllabili. Inoltre, il design of experiments è fondamentale anche per ridurre il numero di esperimenti, quindi il costo, necessario al testing del sistema. Per fare ciò è stata utilizzata la funzionalità **Piano Fattoriale Completo**, resa disponibile dal tool JMP nel quale sono stati utilizzati: come risposta di interesse l'**Average Response Time** (tempo medio di risposta, in particolare si è usato l'Elapsed Time medio), come fattori la **dimensione della risorsa** (categorico a 3 livelli, SMALL, MEDIUM e LARGE) ed il **Throughput** (categorico a 3 livelli). Il Throughput è stato calcolato in funzione della percentuale dell'usable capacity, ed è in particolare definito su 3 livelli:

- **Livello 1**: 25% della usable capacity
- **Livello 2**: 50% della usable capacity
- **Livello 3**: 75% della usable capacity

Si vuole quindi determinare l'influenza di tali fattori sulla risposta di interesse. Per ogni combinazione sono state eseguite 5 ripetizioni, per un totale di $3 \cdot 3 \cdot 5 = 45$ esperimenti, della durata di un minuto ciascuno. Il piano fattoriale completo da origine alla seguente tabella.

Risorsa	Throughput	AvgElapsed
SMALL	25%	6
SMALL	25%	3
SMALL	25%	3
SMALL	25%	3
SMALL	25%	2
SMALL	50%	2
SMALL	50%	3
SMALL	50%	3
SMALL	50%	4
SMALL	50%	2
SMALL	75%	2
SMALL	75%	3
SMALL	75%	3
SMALL	75%	4

Risorsa	Throughput	AvgElapsed
SMALL	75%	2
MEDIUM	25%	357
MEDIUM	25%	962
MEDIUM	25%	941
MEDIUM	25%	444
MEDIUM	25%	154
MEDIUM	50%	718
MEDIUM	50%	247
MEDIUM	50%	465
MEDIUM	50%	421
MEDIUM	50%	164
MEDIUM	75%	309
MEDIUM	75%	379
MEDIUM	75%	906
MEDIUM	75%	465
MEDIUM	75%	396
LARGE	25%	699
LARGE	25%	685
LARGE	25%	594
LARGE	25%	607
LARGE	25%	945
LARGE	50%	816
LARGE	50%	323
LARGE	50%	656
LARGE	50%	625
LARGE	50%	726
LARGE	75%	624
LARGE	75%	542
LARGE	75%	626
LARGE	75%	633
LARGE	75%	446

Ciò che interessa questo studio è verificare l'importanza e la significatività dei fattori: un fattore si dice importante se esprime una buona percentuale di variazione rispetto alla variazione totale del sistema, mentre è detto significativo se rappresenta il contributo che spiega un fattore rispetto a quello spiegato dall'errore. La differenza tra i due è che l'importanza non è un concetto statistico

mentre, di contro, la significatività non è un parametro soggettivo.

3.5.1 Analisi dell'importanza

Per verificare l'importanza di un fattore all'interno del piano, è necessario andare a calcolare alcuni parametri. Chiamati "A" e "B" i fattori di risorsa e request rate, si calcolano i parametri:

- SSA (*sum of square factor A*): percentuale di variazione spiegata dalla dimensione della risorsa;
- SSB (*sum of square factor B*): percentuale di variazione spiegata dal Throughput;
- SSAB (*sum of square interaction A-B*): percentuale di variazione spiegata dall'interazione tra dimensione della pagina e Throughput;
- SSE (*sum of square error E*): percentuale di variazione spiegata dall'errore.

Tramite il tool JMP, si è proceduto allo sviluppo del test degli effetti e all'analisi della varianza, per ottenere poi l'importanza dei fattori tramite le operazioni. Dato l'alto p-value ottenuto per Throughput e per l'interazione dei due fattori considerati, si può concludere già che l'effetto di questi due fattori è trascurabile, e dato il basso valore di p-value per il fattore Dimensione, esso invece è il fattore che influenza maggiormente il modello. Tali considerazioni trovano conferma nei calcoli eseguiti in basso.

Riepilogo della stima

R-quadro	0,742427
R-quadro corretto	0,685189
Scarto quadratico medio	181,2656
Media della risposta	376
Osservazioni (o somme pesate)	45

Analisi della varianza

Origine	DF	Somma dei quadrati	Media quadratica	Rapporto F
Modello	8	3409470,4	426184	12,9708
Errore	36	1182859,6	32857	Prob > F
C. totale	44	4592330,0		<,0001*

Stime dei parametri

Test degli effetti

Origine	Nparm	DF	Somma dei quadrati	Rapporto F	Prob > F
Dimensione	2	2	3294534,5	50,1341	<,0001*
Throughput	2	2	59430,0	0,9044	0,4138
Dimensione*Throughput	4	4	55505,9	0,4223	0,7915

$$\frac{SSA}{SST} = \frac{3294534.5}{4592330.0} \simeq 0.7174 = 71.74\%$$
$$\frac{SSB}{SST} = \frac{59430.0}{4592330.0} \simeq 0.013 = 1.3\%$$
$$\frac{SSAB}{SST} = \frac{55505.9}{4592330.0} \simeq 0.012 = 1.2\%$$
$$\frac{SSE}{SST} = \frac{1182859.6}{4592330.0} \simeq 0.2576 = 25.76\%$$

Come è evidente dai calcoli, il fattore più importante è la dimensione della pagina, mentre l'importanza di Throughput e interazione di secondo livello risulta trascurabile.

In particolare, si osserva il valore dell'R-quadro, che è la percentuale di variabilità della variabile dipendente spiegata dal tuo modello; tale valore risulta superiore al 70%, il che indica un'alta confidenza del modello rispetto ai dati, in relazione all'analisi che si sta svolgendo su tali dati. La conferma si ottiene dal test analitico di Fisher, che riporta un p-value tale da rigettare l'ipotesi di non significatività del modello al 95%. In figura si riportano i grafici dei modelli predetti.

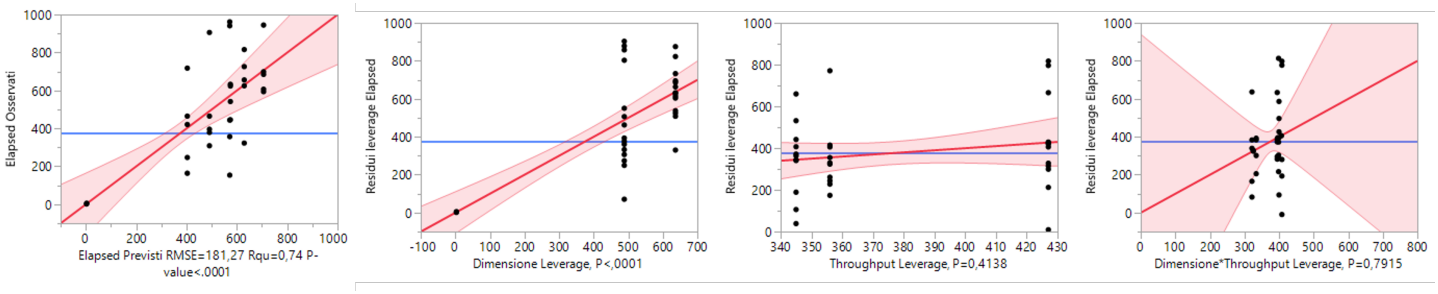


Figura 3.11: Da sinistra a destra: modello completo, Dimensione, Throughput e interazione Dimensione-Throughput

3.5.2 Analisi della significatività

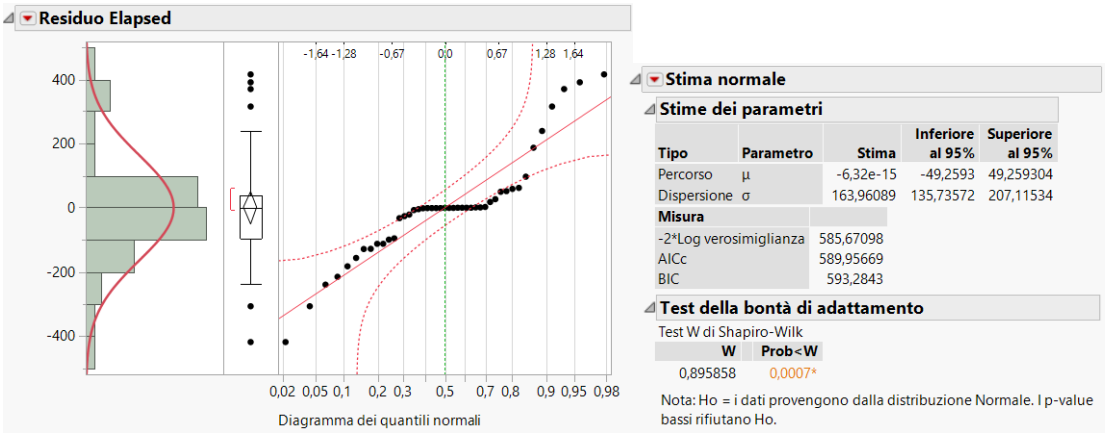
L'ultimo aspetto da valutare è la significatività dei fattori, la quale rappresenta la percentuale di variabilità spiegata da un fattore rispetto a quella spiegata dall'errore. Per studiare la significatività dei fattori considerati si applica un metodo detto **ANOVA** (*ANalysis Of VAriance*). Tuttavia, il metodo preciso da applicare dipende da alcune caratteristiche della distribuzione degli errori, come indicato in tabella.

Normalità	**Omoschedasticità**	**ANOVA**
Si	Si	*F-test*
No	Si	*Kruksal-Wallis test*
Si	No	*Welch's test*
No	No	*Kruskal-Wallis o Friedman test*

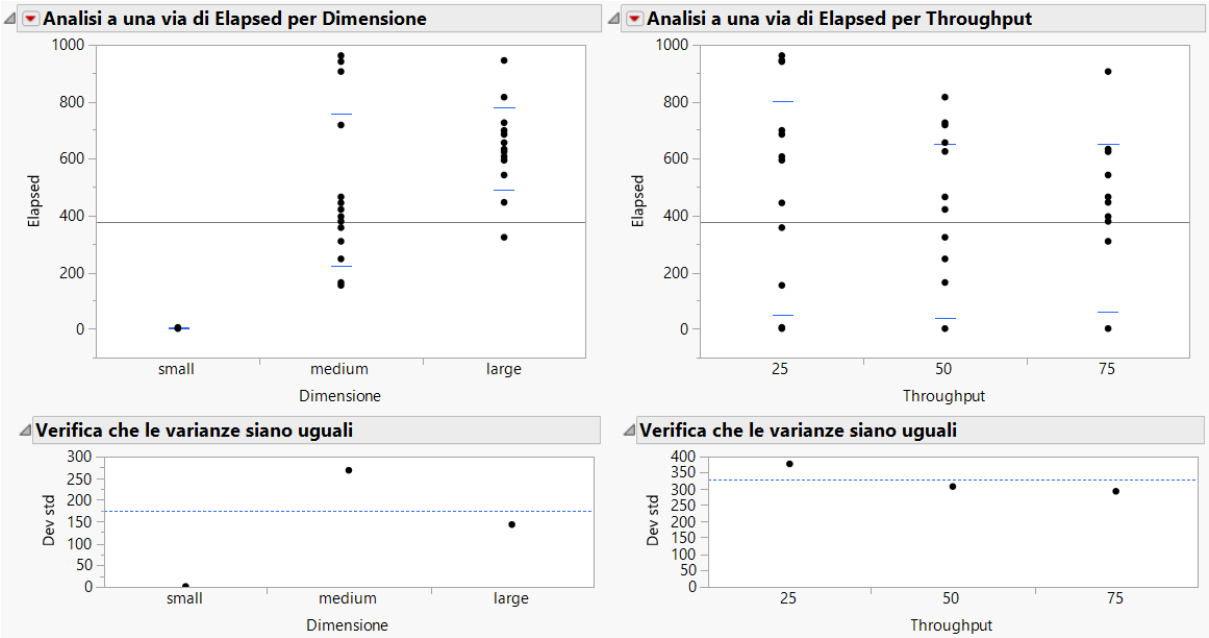
É quindi necessario verificare le due proprietà sui dati. In particolare si parla di **normalità dei residui** se gli errori sono approssimabili ad una distribuzione normale, mentre si parla di

omoschedasticità se la deviazione standard degli errori è costante.

Per verificare la normalità degli errori è possibile effettuare sia un test visuale, QQplot, che un test analitico, test di Shapiro-Wilk, entrambi riportati in figura.



I risultati di entrambi i test rigettano l'ipotesi che gli errori provengano da una popolazione con distribuzione normale. Non è quindi possibile confermare la normalità dei residui, e ci si deve affidare necessariamente al test *Kruksal-Wallis* per l'ANOVA. Nonostante sia superfluo, si è deciso di verificare ugualmente l'omoschedasticità sfruttando JMP. Per verificare l'omoschedasticità degli errori è possibile effettuare un test visuale, tracciandovi i punti a seconda dei livelli dei fattori. Come mostrato in figura, la dispersione dei punti dello scatter plot non è costante, per cui è possibile rifiutare l'ipotesi di omoschedasticità degli errori nel caso della dimensione, mentre è possibile accettarla nel caso del fattore Throughput.



Si è poi scelto di confermare l'ipotesi tramite lo strumento JMP, che mette a disposizione un set di test analitici, ognuno dei quali accetta l'ipotesi nulla per il fattore di dimensione della pagina,

mentre la rigetta per il fattore Throughput. In figura i risultati ottenuti.

Livello	Conteggio	Dev std	Diff assoluta media rispetto alla media	Diff assoluta media rispetto alla mediana
small	15	1,0690	0,6667	0,6667
medium	15	268,4809	209,7156	193,0000
large	15	143,5847	94,4267	93,2667

Test	Rapporto F	Num DF	Den DF	Prob > F
O'Brien[.5]	6,8961	2	42	0,0026*
Brown-Forsythe	8,5893	2	42	0,0007*
Levene	13,6959	2	42	<,0001*
Bartlett	66,2354	2	.	<,0001*

Livello	Conteggio	Dev std	Diff assoluta media rispetto alla media	Diff assoluta media rispetto alla mediana
25	15	377,6330	328,1333	327,0000
50	15	308,1957	268,2667	266,8000
75	15	293,5767	241,7333	236,0000

Test	Rapporto F	Num DF	Den DF	Prob > F
O'Brien[.5]	1,6537	2	42	0,2036
Brown-Forsythe	1,2716	2	42	0,2910
Levene	1,2831	2	42	0,2878
Bartlett	0,5022	2	.	0,6052

Figura 3.13: Test di omoschedasticità per Dimensione (sinistra) e Throughput (destra)

A valle di tale analisi, è possibile confermare che il fattore Dimensione risulta **eteroschedastico**, mentre il fattore Throughput risulta **omoschedastico**.

3.5.3 ANOVA

A valle delle considerazioni fatte, si è applicato il test *Kruksal-Wallis*, detto anche **test della somma dei ranghi**, per verificare quale fattore sia statisticamente significativo. In figura è possibile notare come il test riporti che l'unico fattore significativo a livello statistico è la **Dimensione**, a discapito del Throughput per il quale invece si accetta l'ipotesi nulla, e non è quindi confermabile la significatività statistica.

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

Livello	Conteggio	Somma degli score	Score atteso	Media degli score	(Media-Media0)/Std0
SMALL	15	120,000	345,000	8,0000	-5,419
MEDIUM	15	409,000	345,000	27,2667	1,533
LARGE	15	506,000	345,000	33,7333	3,874

Test a una variabile, approssimazione chi-quadrato

Chi-quadrato	DF	Prob>ChiQu
31,3270	2	<,0001*

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

Livello	Conteggio	Somma degli score	Score atteso	Media degli score	(Media-Media0)/Std0
25%	15	375,000	345,000	25,0000	0,712
50%	15	333,000	345,000	22,2000	-0,278
75%	15	327,000	345,000	21,8000	-0,422

Test a una variabile, approssimazione chi-quadrato

Chi-quadrato	DF	Prob>ChiQu
0,5314	2	0,7667

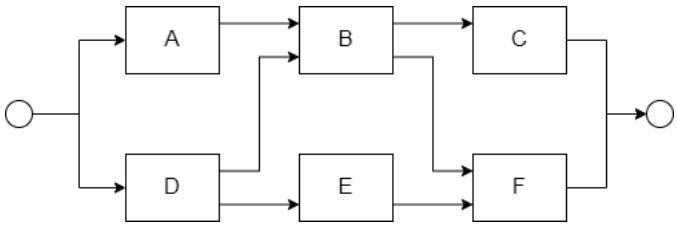
Capitolo 4

RBD

4.1 Esercizio 1

4.1.1 Traccia

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



4.1.2 Svolgimento

Metodo dell'upper-bound

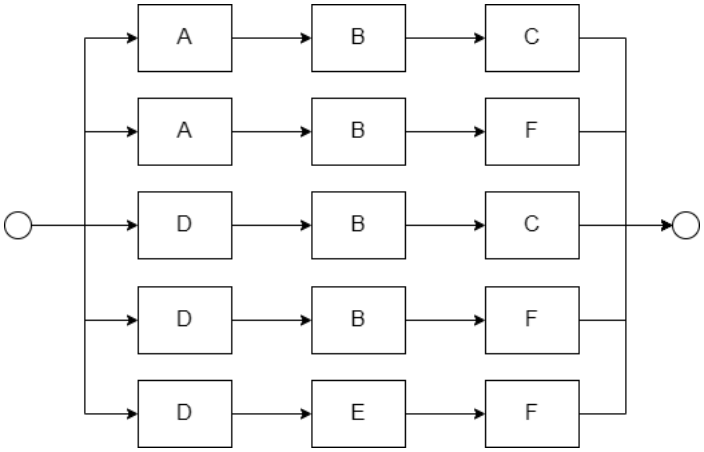
Innanzitutto, è possibile osservare che il sistema non presenta una struttura serie-parallelo. In questi casi si può calcolare rapidamente un limite superiore della reliability del sistema, applicando il **teorema dell'upper bound**. Tale teorema afferma che la reliability di un sistema è minore o uguale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

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

dove $R_{succ-path_i}$ è la reliability dell'i-esimo success path del sistema. Un **success path** è un qualsiasi percorso che conduce dall'input all'output lungo il diagramma. Tale formula fornisce solo un Upper-bound della reliability del sistema, ma se si identifica l'indipendenza tra i success path,

vale la relazione di uguaglianza.

Il sistema è quindi identificabile come un parallelo di success path applicando il teorema dell'upper bound, e si ottiene un diagramma equivalente come il seguente in figura.



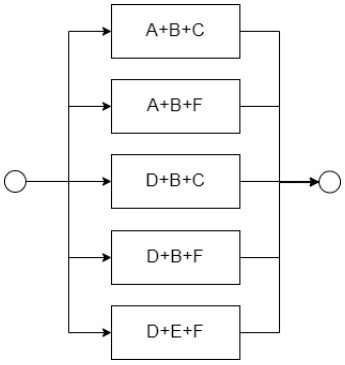
Dal momento che il nuovo sistema ha una struttura serie-parallelo, possiamo calcolarne facilmente la reliability applicando le formule per la risoluzione dei sistemi serie e parallelo. In particolare, per i sistemi costituiti da soli componenti in serie è possibile calcolare la reliability del sistema a partire dalla reliability dei singoli componenti tramite la relazione

$$R_{ser}(t) = \prod_{i=1}^{n} R_i(t) \tag{4.2}$$

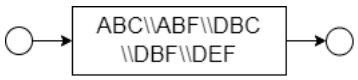
dove n è il numero di componenti in serie e $R_i(t)$ è la reliability dell'i-esimo sistema della serie. Per i componenti in parallelo invece vale la relazione

$$R_{par}(t) = 1 - \prod_{i=1}^{n}(1 - R_i(t)) \tag{4.3}$$

dove n sono i componenti in parallelo e $R_i(t)$ è la reliability del componente i del parallelo. Nel caso specifico, si ha un parallelo di 5 componenti, dove ogni componente è la serie di 3 specifici componenti.



(a) Risoluzione dei sistemi serie



(b) Risoluzione del parallelo

L'upper bound della reliability del sistema è quindi calcolabile come segue:

$$R_{sys} \leq 1 - (1 - R_A R_B R_C)(1 - R_A R_B R_F)(1 - R_D R_B R_C)(1 - R_D R_B R_F)(1 - R_D R_E R_F) \quad (4.4)$$

Dalle ipotesi si ha che ogni componente fallisce randomicamente e indipendentemente, con un tasso di fallimento pari a λ, quindi, per ogni componente si ha $R_i = e^{\lambda t}$. Sostituendo tale informazione nella formula dell'upper bound si ottiene

$$R_{sys}(t) \leq 1 - (1 - e^{3\lambda t})^5 \quad (4.5)$$

Metodo del conditioning

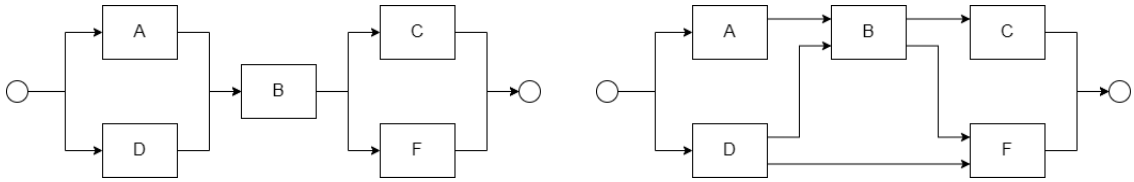
Per calcolare in maniera più affidabile la reliability di un sistema che non può essere ricondotto ad un serie-parallelo classico è possibile sfruttare la tecnica detta del **conditioning**, che consiste nello sfruttare il teorema della probabilità condizionata, andando ad effettuare il condizionamento su un componente chiave del diagramma.
Per tale analisi, nel caso in esame, si è scelto di condizionare la probabilità di funzionamento dell'intero sistema al corretto funzionamento del componente E. Dal momento che la reliability *R(t)* di un sistema è definita come la probabilità che il sistema non fallisca fino all'istante t, è possibile ottenere la reliability tramite la formula

$$R_{sys} = P(sys) = R_i P(sys|i) + (1 - R_i) P(sys|\bar{i}) \quad (4.6)$$

dove

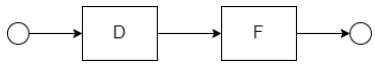
- $P(sys)$ è la probabilità di funzionamento del sistema;
- R_i è la reliability dell'i-esimo componente;
- $P(sys|i)$ è la probabilità che il sistema funzioni dato che il componente i funziona;
- $P(sys|\bar{i})$ è la probabilità che il sistema funzioni dato che il componente i non funziona;

Condizionando il sistema al funzionamento del componente E, si ottengono due sottosistemi:

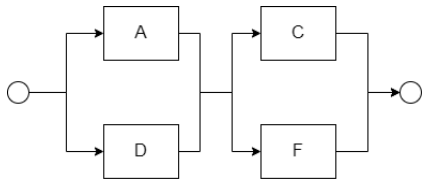


(a) Sistema nel caso di fallimento di E (b) Sistema nel caso di funzionamento di E

Se per il caso in figura (a), riferita al fallimento di E, il calcolo della reliability è riconducibile ad una struttura serie-parallelo, nel caso di funzionamento del componente E è necessario attuare un ulteriore condizionamento, ad esempio al componente B.
Da tale ulteriore scomposizione si trovano i seguenti sotto-diagrammi:



(a) Funzionamento di E e fallimento di B



(b) Funzionamento di E e di B

La reliability totale quindi sarà composta della reliability dei sistemi scomposti:

- **Fallimento di E**:
  $R_{sys|\overline{E}} = (1-(1-R_A)(1-R_D))R_B(1-(1-R_C)(1-R_F)) = (1-(1-e^{-\lambda t})^2)^2 e^{-\lambda t}$
- **Funzionamento di E**:
  $R_{sys|E} = R_B R_{EB} + (1-R_B)R_{E\overline{B}} = e^{\lambda t}(1-(1-e^{-\lambda t})^2)^2 + (1-e^{\lambda t})(e^{-2\lambda t})$
  - **Funzionamento di E + Fallimento di B**:
    $R_{E\overline{B}} = R_D R_F = e^{-2\lambda t}$
  - **Funzionamento di E + Funzionamento di B**:
    $R_{EB} = (1-(1-R_A)(1-R_D))(1-(1-R_C)(1-R_F)) = (1-(1-e^{-\lambda t})^2)^2$

Concludendo, la reliability del sistema totale sarà calcolabile come segue.

$$R_{sys}(t) = R_E R_{sys|E} + (1-R_E)R_{sys|\overline{E}} =$$
$$e^{\lambda t}\big[e^{\lambda t}(1-(1-e^{-\lambda t})^2)^2\big] + (1-e^{\lambda t})\big[(e^{-2\lambda t}) + (1-e^{\lambda t})(1-(1-e^{-\lambda t})^2)^2 e^{-\lambda t}\big] = ...$$
$$... = e^{-5\lambda t} - 5e^{-4\lambda t} + 5e^{-3\lambda t}$$

Dalla reliability è possibile facilmente calcolare anche il *Mean Time To Failure* (MTTF); esso infatti è dato dall'integrale della reliability. Si ottiene quindi la seguente formula

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

Analisi grafica

Per confermare le ipotesi che la reliability ha come upper bound quella calcolata come produttoria dei success paths, è possibile andare a graficare l'andamento di quest'ultima, ad un fissato valore di λ, come ad esempio 0.5. Si ottiene quindi il grafico seguente.

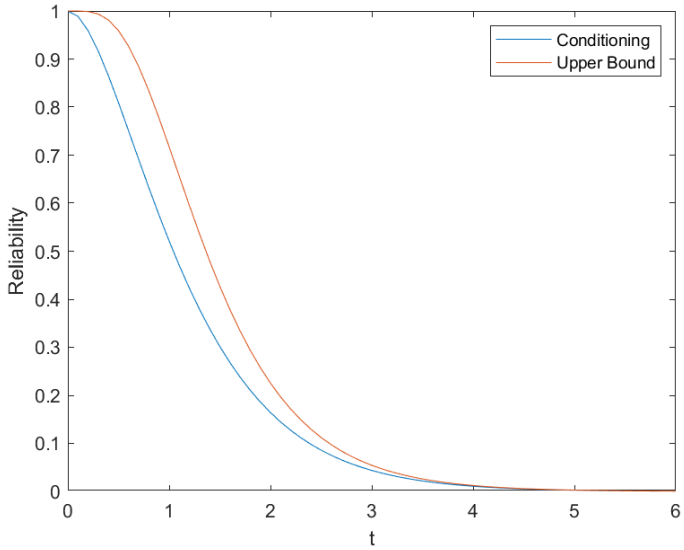


Figura 4.4: Confronto tra i due metodi applicati

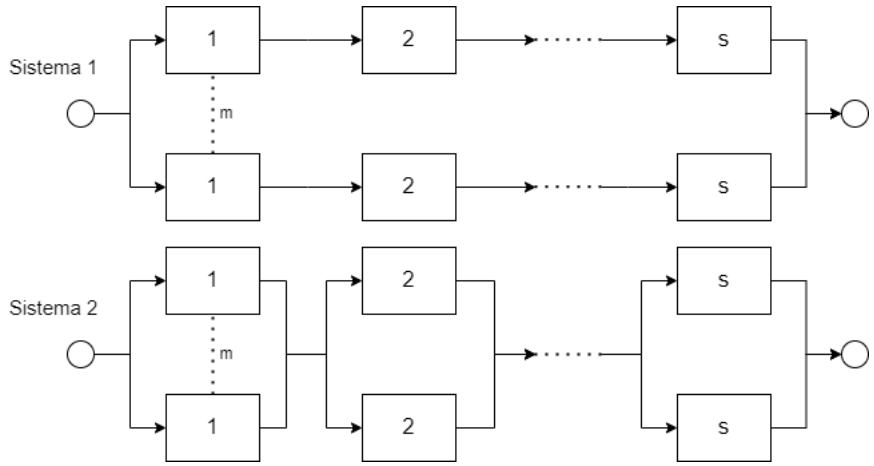
Avendo il grafo della reliability calcolata tramite conditioning sempre al di sotto del suo valore trovato tramite combinazione di success paths, si può concludere che il valore dei success paths da un upper bound della reliability.

4.2 Esercizio 2

4.2.1 Traccia

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

1. Dato che la reliability di un componente è r, derivare l'espressione per le reliability delle due configurazioni.
2. Confrontare le due espressioni per $m = 3$ ed $s = 4$ in funzione del mission time t.
3. Ipotizzando un MTTF pari a 100, quale dei due schemi mostrati in figura assicura una reliability maggiore?
4. Modificare lo schema con la reliability più bassa al fine di ottenere la stessa reliability per entrambi.



4.2.2 Svolgimento

Punto 1

Dal momento che entrambe le configurazioni sono riconducibili ad una struttura serie-parallelo, è possibile calcolare semplicemente la reliability di entrambi:

- **Sistema 1**: il primo sistema si può vedere come un parallelo di m serie, composta ognuna da s componenti. Si ha

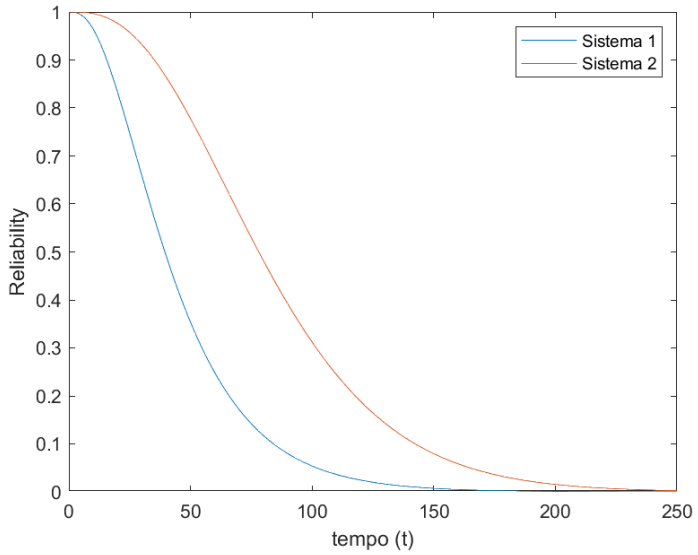
$$R_{sys1} = 1 - \prod_{i=1}^{m}(1 - r^s) = 1 - (1 - r^s)^m \tag{4.7}$$

- **Sistema 2**: il secondo sistema è riconducibile ad una serie di s paralleli, ognuno dei quali conta m componenti. Si ha

$$R_{sys2} = \left(1 - \prod_{i=1}^{m}(1-r)\right)^{s} = (1-(1-r)^{m})^{s} \tag{4.8}$$

Punto 2-3

Per confrontare la due espressioni è possibile avvalersi di una facile tecnica: se un sistema contiene più success paths rispetto all'altro, allora avrà necessariamente una reliability maggiore. Nell'esempio, il primo sistema presenterà unicamente m success paths, composti dalle m serie tra i componenti. Se infatti fallisce un solo componente di una serie, tutta la serie, e quindi il corrispondente success path associato a quella serie risulterà inutilizzabile. Per il secondo sistema si hanno invece un numero di success paths maggiore, pari nello specifico a m^s. Tale risultato è ottenibile considerando le m combinazioni possibili di s componenti, che è pari proprio a m^s. Di conseguenza, visto il maggior numero di success paths, il sistema 2 risulta avere una reliability maggiore.
Per avvalorare la tesi, come richiesto, si è confrontato l'andamento della reliability dei due sistemi al variare del parametro r, che rappresenta la reliability di un singolo componente, e fissando i valori di m e di s rispettivamente a 3 e 4. Graficandone gli andamenti in funzione del tempo si ottiene la figura seguente.



Punto 4

Per ottenere la reliability uguale per entrambi i sistemi, è necessario aumentare quella del primo sistema. Il valore a cui arrivare dipende dal numero di success paths del secondo sistema: infatti, per poter rendere uguali a livello di reliability i due sistemi è necessario che abbiano lo stesso numero di success paths. Per fare ciò è necessario eguagliare gli upper bound, tenendo presente però che essendo la prima configurazione composta da soli success paths indipendenti, l'upper

bound è pari proprio al valore effettivo della reliability. Per calcolare il numero di success paths si eguagliano le formule di reliability:

$$R_{sys1} = R_{sys2} \rightarrow 1 - (1 - r^s)^{m_1} = (1 - (1 - r)^m)^s \tag{4.9}$$

Siccome il valore di m_1 è quello che indica nel primo sistema il numero di success paths, è necessario fissare gli altri parametri per andare a calcolare il valore di m_1 al variare di r. Posti $m = 3$ e $s = 4$, si ha:

$$1 - (1 - r^4)^{m_1} = (1 - (1 - r)^3)^4 \rightarrow m_1 = \frac{\ln(1 - (1 - (1 - r)^3)^4)}{\ln(1 - r^4)} \tag{4.10}$$

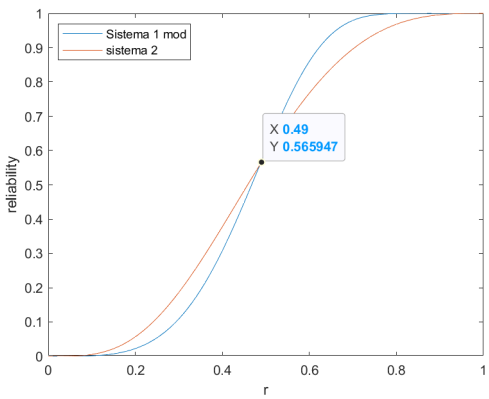
Per il calcolo del valore effettivo di m_1, si è scelto di considerare un mission time di 72 ore, per andare a definire il valore della reliability dei componenti per cui si avesse tale mission time, avendo un MTTF pari a 100. Calcolando tale valore si ottiene:

$$r(72) = e^{-\frac{72}{100}} \simeq 0.49 \tag{4.11}$$

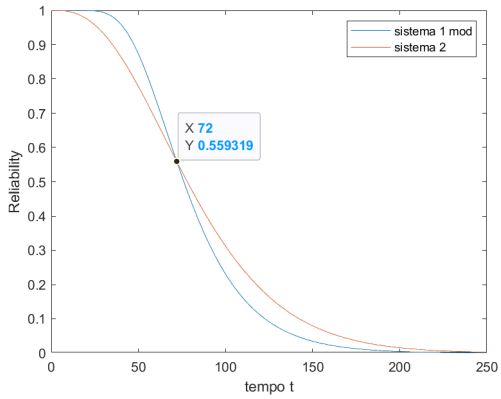
Con tale valore di reliability del componente, il valore di success paths che deve avere il sistema 1 sarà pari a:

$$n = \frac{\ln(1 - (1 - (1 - 0.49)^3)^4)}{\ln(1 - 0.49^4)} \simeq 14 \tag{4.12}$$

Il numero di success paths necessari ad avere uguale reliability per il sistema 1 è quindi pari a 14. Di seguito il grafico di confronto tra il sistema modificato e il sistema 2, considerando $m = 3$ ed $s = 4$.



(a) Valori di reliability al variare di r



(b) Valori di reliability al variare del tempo t

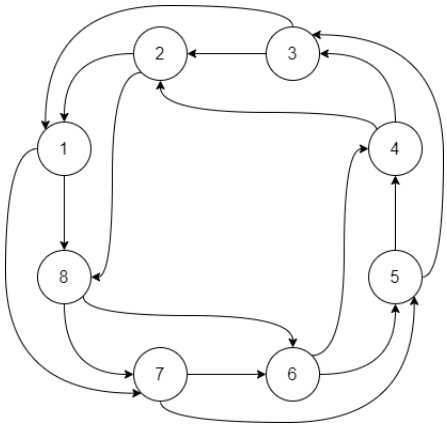
4.3 Esercizio 3

4.3.1 Traccia

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

1. Assumendo che i collegamenti siano perfetti e che ogni nodo abbia una reliability R_m, derivare l'espressione per la reliability della rete.

2. Se R_m segue una legge esponenziale di fallimento ed il failure rate di ogni nodo è di 0.005 fallimenti per ora, determinare la reliability del sistema alla fine di un periodo di 48 ore.



4.3.2 Svolgimento

Punto 1

Per il calcolo della reliability della rete *skip-ring*, è necessario andare a identificare tutti i possibili scenari di fallimento. Nel caso in esame, si ha che non possono fallire un numero di nodi superiore a 4. Analizzando caso per caso i vari tipi di fallimento si hanno le seguenti considerazioni:

- Se non si ha il fallimento di alcun nodo varrà la serie tra i nodi, e tale valore sarà indicato come $P(sys|0 - tot - fails)$;

- Se si ha il fallimento di un solo nodo, si avranno 8 differenti serie di componenti, a seconda del nodo che è fallito. Tale valore indica la $P(sys|1 - tot - fail)$ e sarà quindi moltiplicato per il numero di combinazioni possibili;

- Se si ha il fallimento di 2 nodi, si avranno 20 combinazioni diverse di serie tra componenti, ognuna delle quali è possibile definirla dalla $P(sys|2 - tot - fails)$. Per il totale è necessario moltiplicare tale probabilità per il numero di combinazioni, cioè 20;

- Se si ha il fallimento di 3 nodi, si avranno 16 possibili serie di componenti, ognuna delle quali è identificata tramite la $P(sys|3-tot-fails)$. Per ottenere la reliability di tutte le configurazioni occorre moltiplicare tale valore per 16.
- Se si ha il fallimento di 4 nodi, si hanno solo 2 combinazioni possibili di serie tra i componenti, e ognuna di queste serie è caratterizzata da $P(sys|4-tot-fails)$.

La reliability totale sarà quindi la somma delle reliability trovate per ogni caso. Si ottiene:

$$\begin{aligned} R_{sys} = P(sys) &= P(sys|0-tot-fails) + 8P(sys|1-tot-fail) + 20P(sys|2-tot-fails) \\ &\quad + 16P(sys|3-tot-fails) + 2P(sys|4-tot-fails) \\ &= R_m^8 + 8R_m^7(1-R_m) + 20R_m^6(1-R_m)^2 + 16R_m^5(1-R_m)^3 + 2R_m^4(1-R_m)^4 \end{aligned} \tag{4.13}$$

Punto 2

Supponendo che la reliability di ogni componente abbia un andamento esponenziale $e^{-\lambda t}$, si vuole calcolare la reliability ad un mission time pari a 48. Sostituendo nell'espressione della reliability $\lambda = 0.005$, e valutando l'espressione al punto $t = 48$, si ottiene

$$R_{sys}(48) \simeq 0.729 \tag{4.14}$$

Graficando l'andamento della reliability in funzione del mission time, si individua sul grafico tale punto, ottenendo la figura seguente.

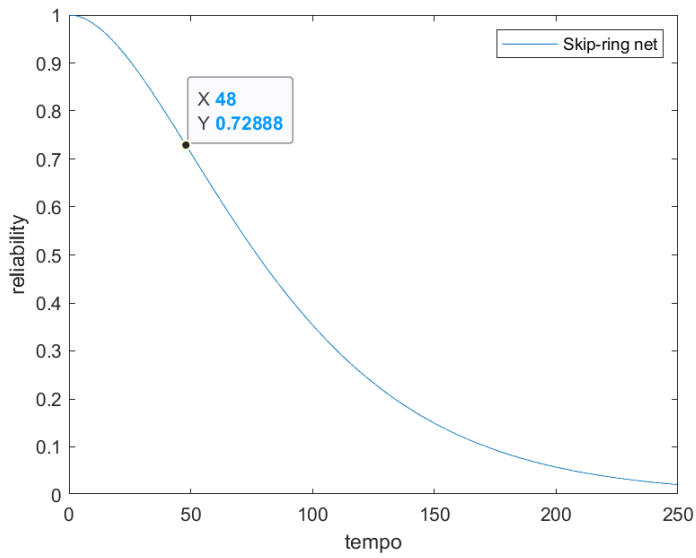


Figura 4.6: Andamento della reliability nella rete skip-ring

La rete non è di semplice analisi, ma quello che può aiutare è costruirne un Fault Tree o un diagramma RBD, che semplificano il lavoro. Unico vincolo da rispettare come detto è quello di avere un fallimento di non più di due componenti consecutive, e tale caratteristica della rete porta alla costruzione dei seguenti Fault Tree e RBD per la rete.

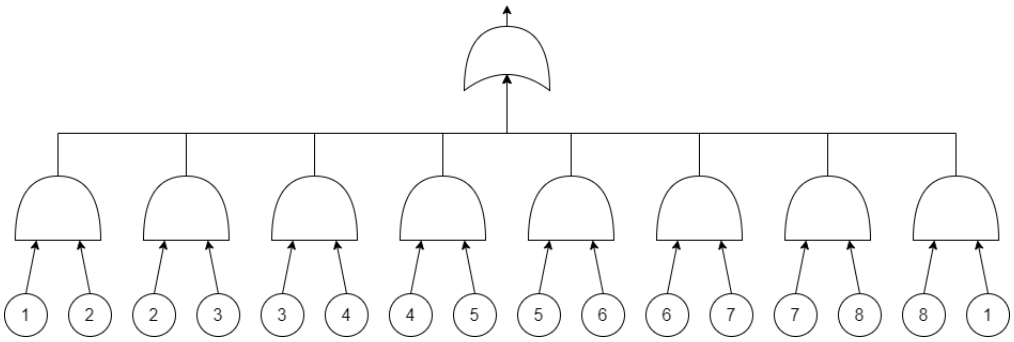


Figura 4.7: Fault Tree della rete skip-ring

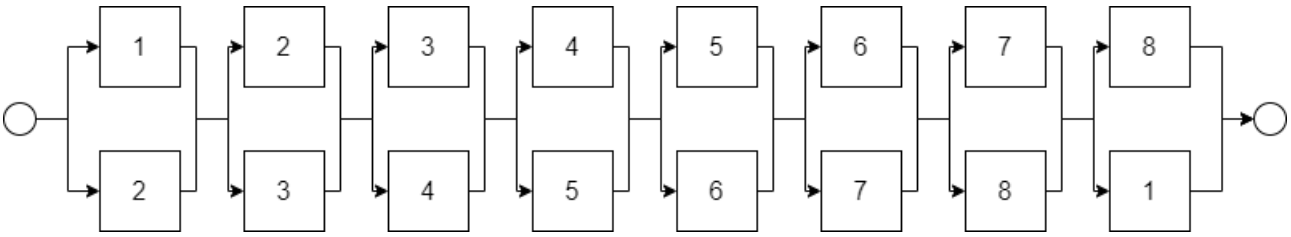


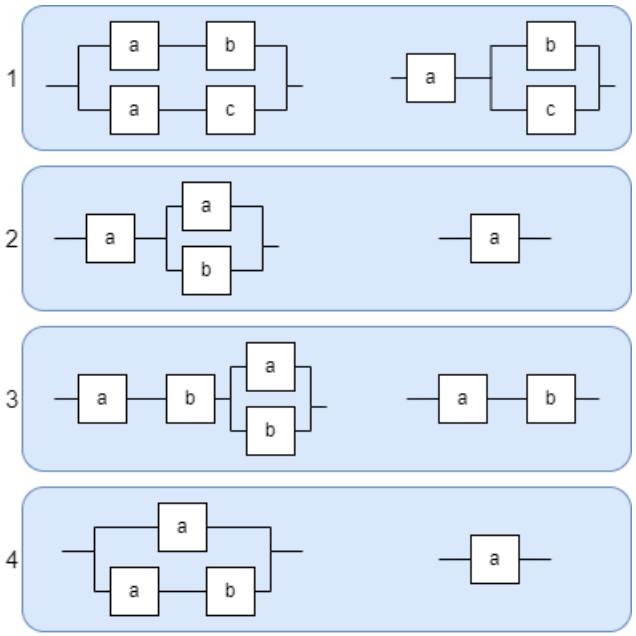
Figura 4.8: RBD della rete skip-ring

4.4 Esercizio 4

4.4.1 Traccia

Comparare la reliabiliy dei sistemi in figura, assumendo un tempo di fallimento esponenziale e i seguenti valori:

- $\text{MTTF}_a = 1000$ h
- $\text{MTTF}_b = 9000$ h
- $\text{MTTF}_c = 2000$ h



4.4.2 Svolgimento

Per il calcolo delle reliability, essendo ogni configurazione esprimibile come combinazioni di serie e paralleli, è sufficiente applicare le regole di risoluzione già precendentemente utilizzate. Per le successive analisi specifiche, si considererà la configurazione di sinistra come *configurazione 1*, mentre quella di destra sarà indicata con *configurazione 2*. Esprimendo quindi le reliability dei singoli componenti come:

- $R_a(t) = e^{-\frac{t}{1000}}$
- $R_b(t) = e^{-\frac{t}{9000}}$
- $R_c(t) = e^{-\frac{t}{2000}}$

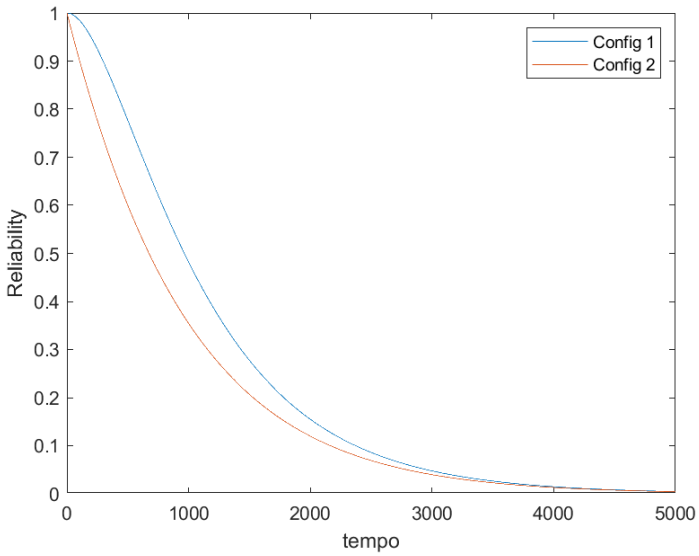
si va ad analizzare la reliability per ognuna delle coppie date.

Primo Confronto

Nel primo caso, il sistema 1 presenta un parallelo di due serie, ognuna composta da 2 componenti, mentre il sistema 2 presenta la serie tra il componente a, e il parallelo dei componenti b e c. Le reliability calcolate hanno quindi tali espressioni:

- $R_{sys1} = 1 - (1 - R_a R_b)(1 - R_a R_c) = -e^{-0.0026t} + e^{-0.0011t} + e^{-0.0015t}$
- $R_{sys2} = R_a[1 - (1 - R_b)(1 - R_c)] = -e^{-0.0016t} + e^{-0.0011t} + e^{-0.0015t}$

Dall'espressione analitica ci si può accorgere che **il sistema 1 è più reliable del sistema 2**. Tale risultato è confermato anche dall'analisi grafica dei due andamenti.

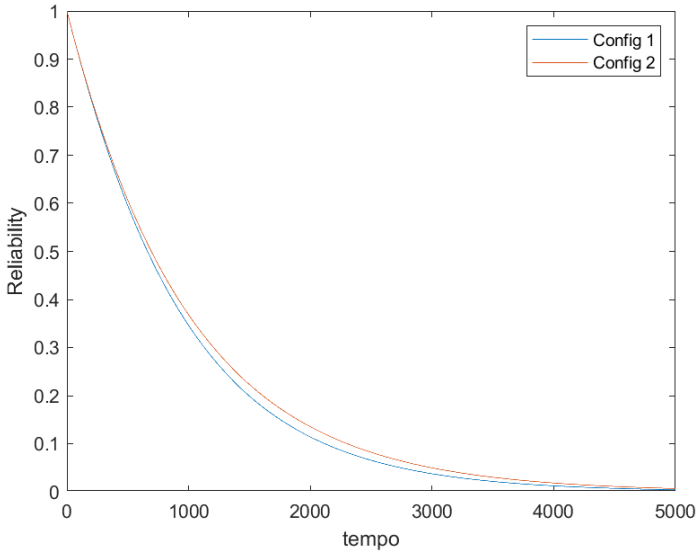


Secondo Confronto

Nel secondo caso, il sistema 1 presenta un serie tra un componente e un parallelo, mentre il sistema 2 è formato da un solo componente. Le reliability calcolate hanno quindi tali espressioni:

- $R_{sys1} = R_a[1 - (1 - R_b)(1 - R_a)] = -e^{-0.0021t} + e^{-0.002t} + e^{-0.0011t}$
- $R_{sys2} = R_a = e^{-0.001t}$

Dall'espressione analitica ci si può accorgere che **il sistema 2 è più reliable del sistema 1**. Tale risultato è confermato anche dall'analisi grafica dei due andamenti.

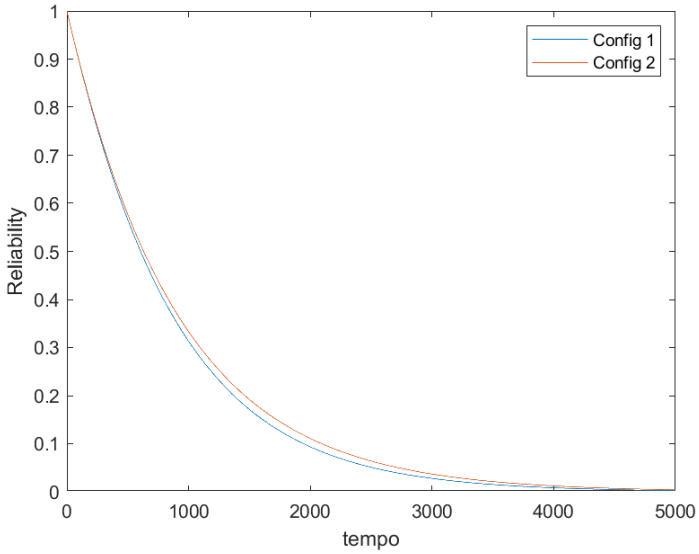


Terzo Confronto

Nel terzo caso, il sistema 1 presenta un serie tra i componenti a e b e un parallelo degli stessi, mentre il sistema 2 è formato dalla sola serie dei due componenti a e b. Le reliability calcolate hanno quindi tali espressioni:

- $R_{sys1} = R_a R_b[1 - (1 - R_a)(1 - R_b)] = -e^{-0.0021t} + e^{-0.002t} + e^{-0.0011t}$
- $R_{sys2} = R_a R_b = e^{-0.0011t}$

Dall'espressione analitica ci si può accorgere che **il sistema 2 è più reliable del sistema 1**. Tale risultato è confermato anche dall'analisi grafica dei due andamenti.

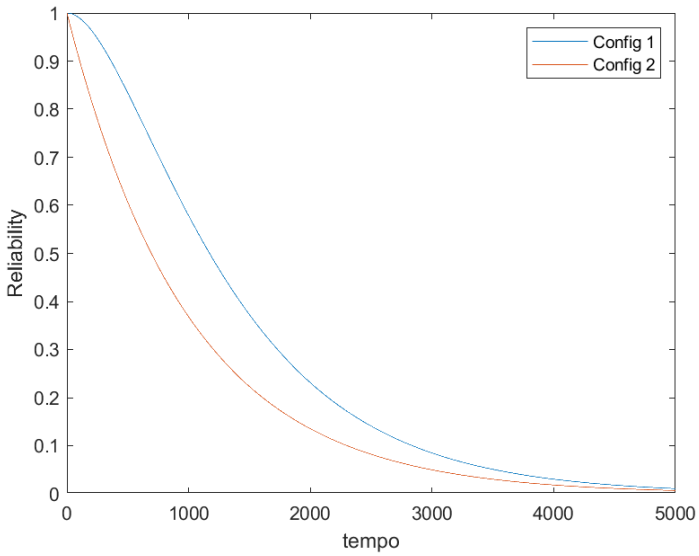


Quarto Confronto

Nel terzo caso, il sistema 1 presenta un parallelo tra il componente a e la serie dei componenti a e b, mentre il sistema 2 è formato dal solo componente a. Le reliability calcolate hanno quindi tali espressioni:

- $R_{sys1} = 1 - (1 - R_a)(1 - R_a R_b) = -e^{-0.0021t} + e^{-0.0011t} + e^{-0.001t}$
- $R_{sys2} = R_a = e^{-0.001t}$

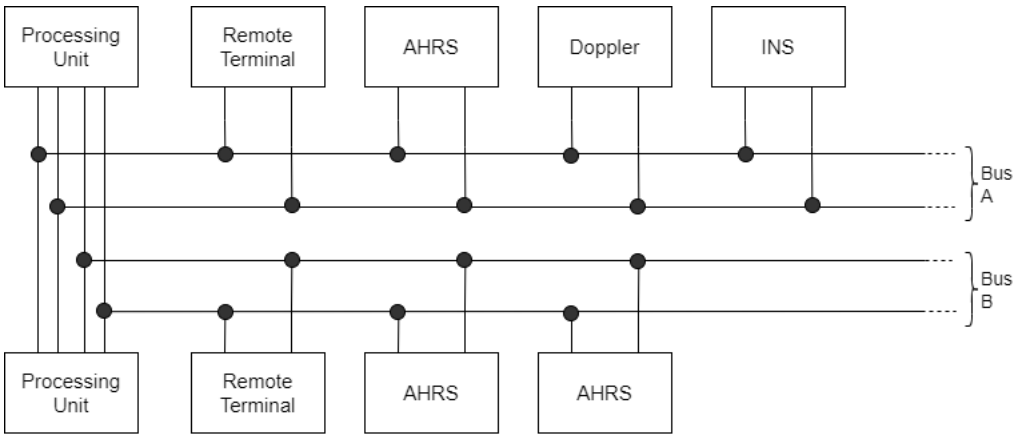
Dall'espressione analitica ci si può accorgere che **il sistema 1 è più reliable del sistema 2**. Tale risultato è confermato anche dall'analisi grafica dei due andamenti.



4.5 Esercizio 5

4.5.1 Traccia

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



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

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

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

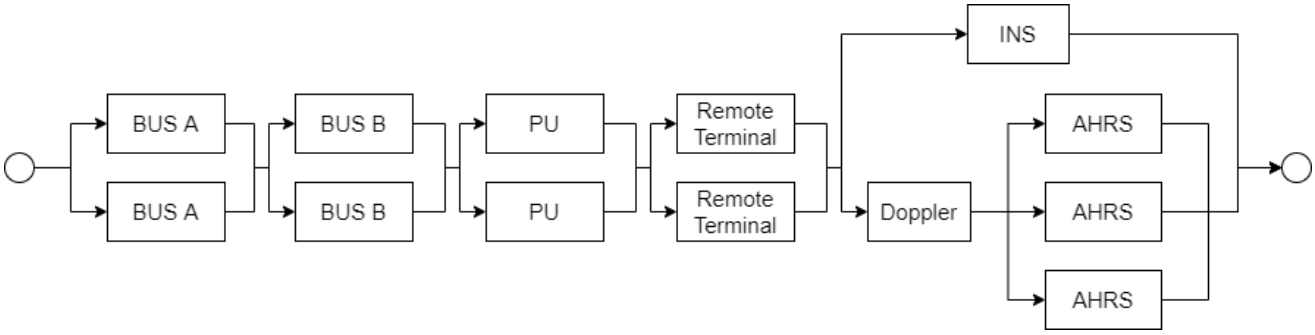
4.5.2 Svolgimento

Punto 1

Per tracciare l'RBD di tale sistema, è necessario fare alcune considerazioni sulla composizione del sistema stesso, al fine di delineare come saranno disposti gli elementi all'interno dell'RBD.

- Avendo una configurazione dual-redundant per i bus, è necessario avere un parallelo tra ogni coppia di bus A e bus B, che devono essere posti in successivamente in serie, visto che è necessaria almeno una linea di bus di entrambi per il corretto funzionamento del sistema;
- Siccome è presente una configurazione dual-redundant anche per i blocchi PU e RT, si avrà, in serie ai bus anche i paralleli delle due copie ridondanti di Processing Unit e Remote Terminal, visto che per il funzionamento del sistema è necessario almeno un componente PU e un componente RT;
- Successivamente, in serie a tutto quello che si è trovato fino ad ora, c'è il blocco relativo alla navigazione. Siccome la navigazione dell'elicottero è garantita se è funzionante il sistema INS, oppure la combinazione di un Doppler e di almeno un componente AHRS, allora l'RBD relativo ai sistemi di navigazione si compone di un parallelo tra il sistema INS e la serie del Doppler e del parallelo dei 3 componenti AHRS.

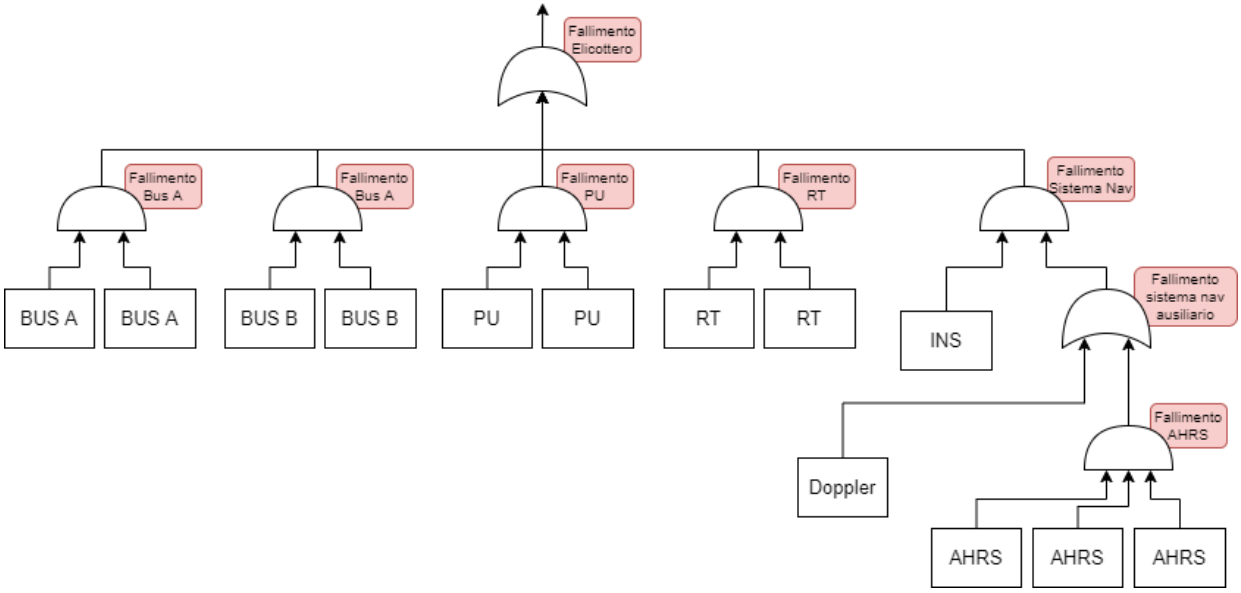
Graficamente, tali considerazioni portano alla costruzione del seguente RBD.



Punto 2

A partire dal diagramma RBD del sistema, la costruzione del relativo Fault Tree risulta alquanto semplice. Infatti a componenti in serie corrispondono componenti in OR all'interno dell'albero,

mentre un parallelo tra due componenti si traduce nell'AND di tali componenti all'interno dell'albero. Considerate tali regole, per l'RBD costruito precedentemente si ha il seguente Fault Tree.



Un **cut-set** in un FT è un insieme di basic event i quali, se occorrono simultaneamente, fanno si che occorra anche il top level event, ovvero il fallimento del sistema. Un cut-set è detto **minimal cut-set** se non può essere ridotto senza perdere il suo status di cut-set. Al fine di evidenziare i minimal cut-set del sistema, è utile esprimere il Fault Tree in forma analitica tramite la sua **funzione di struttura**, che associa ad ogni AND il prodotto dei componenti che concorrono all'operazione, mentre le OR vengono tradotte come delle somme. Da ciò si ottiene :

$$\begin{aligned}\phi = (BUS_A \cdot BUS_A) + (BUS_B \cdot BUS_B) + (PU \cdot PU) + (RT \cdot RT) + \\ + [INS \cdot (Doppler + (AHRS \cdot AHRS \cdot AHRS))] = \\ = (BUS_A \cdot BUS_A) + (BUS_B \cdot BUS_B) + (PU \cdot PU) + (RT \cdot RT) + (INS \cdot Doppler) + \\ + (AHRS \cdot AHRS \cdot AHRS \cdot INS) \quad (4.15)\end{aligned}$$

Da tale formulazione, i mininul cutset sono determinati dai mintermini presenti all'interno della funzione, e sono quindi i seguenti:

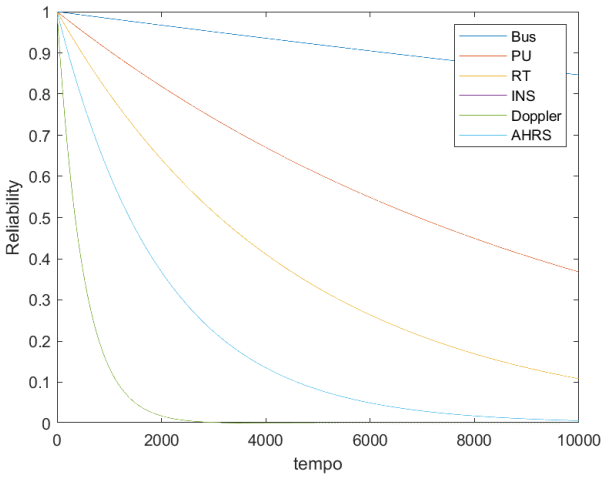
- $MCS_1 = \{BUS_A, BUS_A\}$;
- $MCS_2 = \{BUS_B, BUS_B\}$;
- $MCS_3 = \{PU, PU\}$;
- $MCS_4 = \{RT, RT\}$;
- $MCS_5 = \{INS, Doppler\}$;
- $MCS_6 = \{AHRS, AHRS, AHRS, INS\}$;

Punto 3

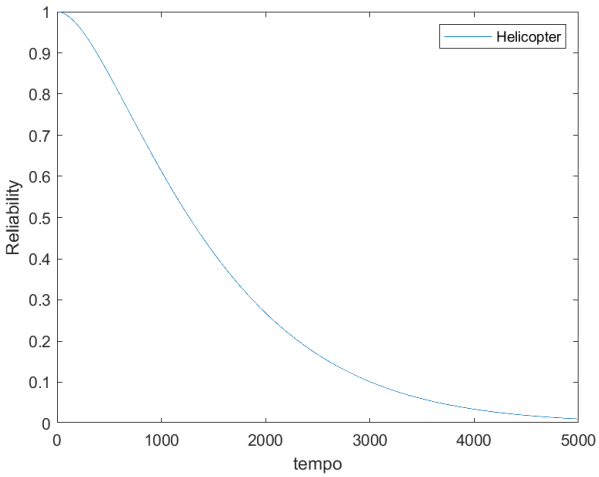
Per il calcolo della reliability del sistema è possibile affidarsi alle formule di risoluzione degli RBD per serie e parallelo, dal momento che la struttura del diagramma è esprimibile come serie-parallelo. Ipotizzando un tempo al fallimento esponenziale per ogni componente e i MTTF dati, si ricava che:

$$\begin{aligned} R_{sys} &= [1-(1-R_{bus})^2][1-(1-R_{bus})^2][1-(1-R_{PU})^2][1-(1-R_{RT})^2]\cdot \\ &\qquad \cdot[1-(1-R_{INS})(1-R_{doppler})][1-(1-R_{INS})(1-R_{AHRS})^3] = \\ &= [1-(1-e^{-\frac{t}{60000}})^2][1-(1-e^{-\frac{t}{60000}})^2][1-(1-e^{-\frac{t}{10000}})^2][1-(1-e^{-\frac{t}{4500}})^2]\cdot \\ &\qquad \cdot[1-(1-e^{-\frac{t}{2000}})(1-e^{-\frac{t}{500}})][1-(1-e^{-\frac{t}{2000}})(1-e^{-\frac{t}{2000}})^3] \end{aligned} \tag{4.16}$$

Graficando i vari andamenti si ottengono le seguenti figure.



(a) Reliability dei componenti



(b) Reliability dell'intero sistema

Calcolando il valore della reliability al tempo $t = 1$, si ottiene il valore cercato dopo 1 ora di volo, che è pari a

$$R_{sys}(1) \simeq 0.99999894132 \tag{4.17}$$

Punto 4

Nelle analisi precedenti si è supposto che ogni sistema sia in grado di rilevare perfettamente un proprio fallimento e conseguentemente disabilitarsi. Considerando invece che un componente possa non riconoscere un suo fallimento e il sistema possa non riadattarsi per evitare il fallimento dello stesso, è necessario considerare un **fattore di coverage** per alcuni componenti che non sia perfetto. Dai dati forniti, evince che il componente su cui è necessario considerare un coverage non perfetto è la Processing Unit. Chiamato c il fattore di coverage, è necessario quindi modificare la formula della reliability nel caso delle PU. Infatti, la reliability del parallelo dei componenti PU è data dalla probabilità che una replica funzioni più la probabilità che tale replica fallisca per la probabilità di

rilevare il fallimento, moltiplicato la probabilità che l'altra replica funzioni.

$$[R_{PC} + c(1 - R_{PC})R_{PC}] \tag{4.18}$$

L'espressione della reliability assume quindi la seguente forma:

$$R_{sys} = [1 - (1 - R_{bus})^2][1 - (1 - R_{bus})^2][R_{PC} + c(1 - R_{PC})R_{PC}][1 - (1 - R_{RT})^2]\cdot \\ \cdot [1 - (1 - R_{INS})(1 - R_{doppler})][1 - (1 - R_{INS})(1 - R_{AHRS})^3] \tag{4.19}$$

Per ottenere una reliability maggiore di 0.99999 dopo un ora, basta calcolare l'espressione precedente per $t = 1$ e calcolare la disequazione $R_{sys} \geq 0.99999$ nella sola variabile c. Si ottiene:

$$R_{sys} \geq 0.99999 \implies c \geq 0.91057 \tag{4.20}$$

Volendo analizzare tale valore sul grafico, è possibile graficare l'andamento della reliability ad 1 ora di volo al variare del paramentro di coverage, per ottenere la figura seguente.

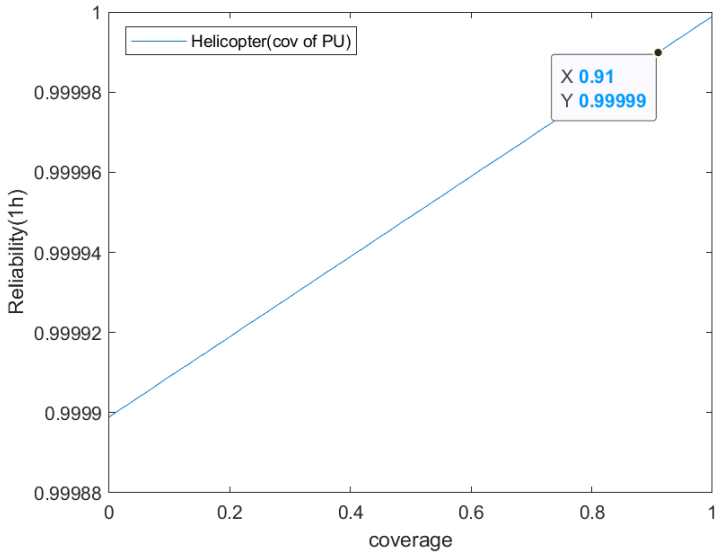


Figura 4.10: Valore di $R_{sys}(1)$ al variare della coverage

Capitolo 5

Field Failure Data Analysis

5.1 Traccia

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

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

Rispondere poi ai seguenti quesiti:

- Si possono usare le stesse finestre di coalescenza in diversi nodi (sia per Mercury che per BG/L) e categorie di errore (solo per Mercury)?
- Cosa si può dire della relazione tra l'affidabilità del sistema e l'affidabilità dei nodi?
- Esistono colli di bottiglia dell'affidabilità (cioè, i principali contributori al numero totale di fallimenti)?
- Nodi funzionali simili (per esempio, due nodi di calcolo Mercury tg-cX o nodi BG/L I/O Ri:Mx:Nz) presentano parametri di affidabilità simili?
- Esiste una relazione tra i tipi di errore e il nodo (solo per Mercury)?

5.2 Field Feilure Data Analysis

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

1. **Data Collection**: si raccolgono i dati del funzionamento del sistema in file tipicamente detti di *log*. Tali file sono grezzi;

2. **Data Filtering**: si effettua un filtraggio dei dati raccolti per concentrare l'analisi solo sui dati significativi;

3. **Data Manipulation**: i dati vengono manipolati al fine di ottenere una corrispondenza tra dati raccolti e fallimenti del sistema;

4. **Data Analysis**: i dati sono sottoposti ad analisi, per differenti scopi, che possono essere quello di trovare la curva di *reliability* del sistema, fino all'identificazione della distribuzione del *Time to Failure*.

L'analisi proposta in seguito sui due sistema precedentemente citati, parte a valle della fase di filtering, dal momento che i file proposti risultano già pre-elaborati.

5.3 Mercury

5.3.1 Descrizione del sistema

Mercury è un super calcolatore composto da diversi componenti, avente un architettura a 3 livelli con un nodo centrale che governa la struttura chiamato *master*. Ogni livello ha differenti responsabilità: *login* si occupa degli accessi, *computation* si occupa delle operazioni elaborative, mentre *storage* gestisce gli accessi alle memorie. L'architettura è schematizzata nella figura seguente.

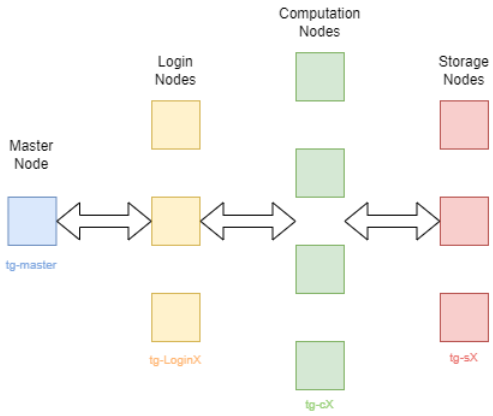


Figura 5.1: Architettura di Mercury

I dati da analizzare sono stati raccolti sfruttando un processo demone *syslog*. A valle dell'esecuzione di tale processo si ottiene il file di log chiamato *MercuryErrorLog.txt*. Tale file contiene 80854 entry, che riguardano solo errori fatali del sistema, formattate come segue:

- **timestamp**: valore di intero temporale in formato UNIX;

- **nodo origine**: stringa del tipo *tg-x* 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. Esistono 5 tipi di sottosistemi e un nodo può contenerne più di uno: DEV, MEM, I-O, NET, e PRO;
- **messaggio**: stringa che contiene il messaggio dell'errore.

5.3.2 Fase di Data Manipulation del sistema

L'obiettivo della fase di manipulation è identificare i fallimenti del sistema. A partire dal log filtrato infatti, si vuole estrarre una relazione tra errori e fallimenti del sistema, ma tale relazione non è mai 1:1.
Per identificare i fallimenti si utilizza ad esempio la tecnica di **coalescenza**, in grado di rilevare la correlazione tra le entry del file di log.
Un tipo di coalescenza che è possibile utilizzare è quella **temporale**: si parla di coalescenza temporale quando entry differenti vengono raggruppate secondo un algoritmo che misura la distanza tra esse a livello temporale. L'algoritmo è detto *tupling*, e i gruppi di entry che si generano vengono dette **tuple**. L'algoritmo lavora come segue: se la differenza tra due timestamp di due entry consecutive è minore di una soglia W, detta **finestra di coalescenza**, allora le due entry fanno parte della stessa tupla.
La scelta cruciale è quindi quella del valore della finestra W. Tale valore influenza i risultati dell'algoritmo:

- se si considera una finestra di coalescenza troppo grande si incorre in **collisioni** tra le entry di una stessa tupla: guasti differenti saranno contenute in una stessa tupla;
- se viceversa, la finestra di coalescenza è troppo piccola, si incorre in **troncamenti**: eventi relativi allo stesso guasto sono inseriti in tuple differenti.

Analizzando la tuple, si tende a preferire una finestra di coalescenza piccola, in modo da effettuare una stima pessimistica della curva di reliability stessa. Si accettano quindi più volentieri troncamenti tra le tuple piuttosto che collisioni. La scelta del valore di W è demandata alla fase di **sensitivity analysis**. Per prima cosa si applica l'algoritmo di tupling al variare di W e, dopo aver graficato l'andamento delle tuple, si sceglie come valore ottimo della finestra di coalescenza, il valore di W immediatamente successivo al valore corrispondente al gomito della curva.
Nel caso dell'analisi del sistema Mercury, tale fase ha prodotto il seguente andamento.

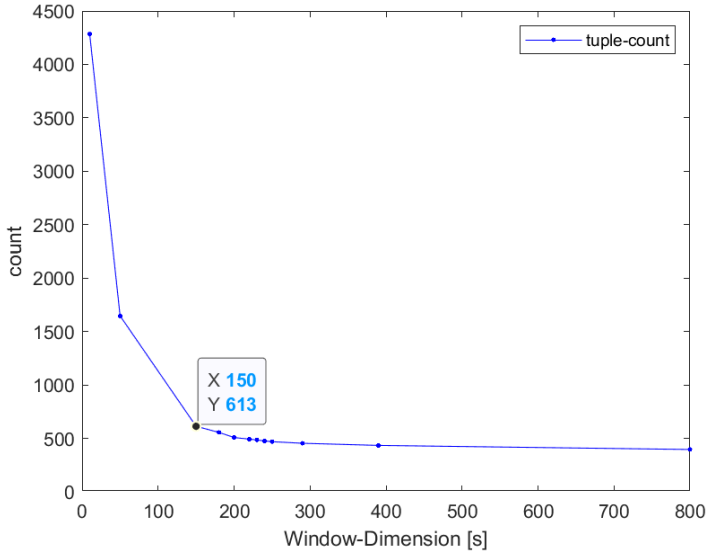


Figura 5.2: Sensitivity Analysis - Mercury

Si è quindi deciso di scegliere un valore di finestra pari a 200 secondi (circa 4 minuti). Una volta fissato il valore della finestra di coalescenza, si passa a raggruppare le entry sfruttando l'algoritmo di tupling con il valore specifico di W, ottenendo un file diverso per ogni gruppo di entry.
Date le tuple così composte è possibile ricavare alcune informazioni aggiuntive:

- *lenght*: differenza tra i timestamp della prima e dell'ultima entry della tupla, indica la durata della tupla;
- *starting points*: timestamp della prima entry della tupla con annessa durata della stessa.
- *interarrival*: tempo (in secondi) che intercorre tra la prima entry della tupla corrente e l'ultima entry della tupla precedente;

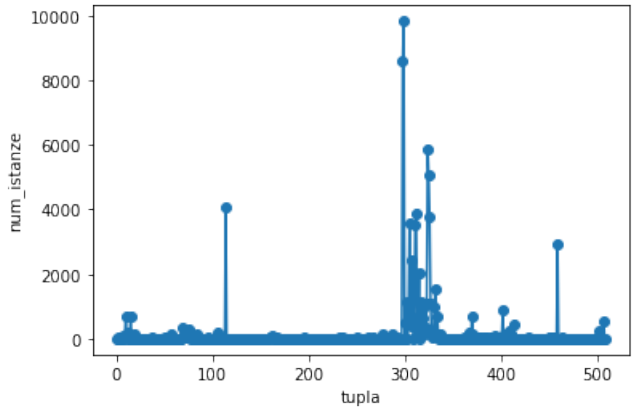
Bottleneck

Una prima analisi per verificare la presenza o meno di un bottleneck del sistema è verificare se ci siano nodi particolarmente avvezzi ad errori e se tali errori sono contenuti in un intervallo temporale specifico. Tale analisi è possibile farla andando a considerare il numero di istanze che sono presenti nelle varie tuple; le tuple che conterranno maggiormente errori identificheranno un problema specifico, che se si dimostra essere dovuto ad un singolo componente, è possibile identificare come collo di bottiglia del sistema stesso. Per la verifica delle distribuzioni e per trovare le tuple maggiormente popolose, si è scelto di usare uno script python:

```
1  ist_count={}
2  perc = '/FFDA/ffdatoolset_21/tupling_MercuryErrorLog_200'
3  for i in range(1,509):
4      doc=open(perc+'/tuple_'+str(i))
5      ist_count[i]=len(doc.readlines())
```

```python

for i in range(0,6):
    max_key = max(ist_count, key = ist_count.get)
    print("massimo num istanze: tuple_" + str(max_key)+ " - totale istanze: " +
                                                +str(ist_count[max_key]))
    ist_count.pop(max_key)
```

Tale esecuzione ha portato i seguenti risultati:



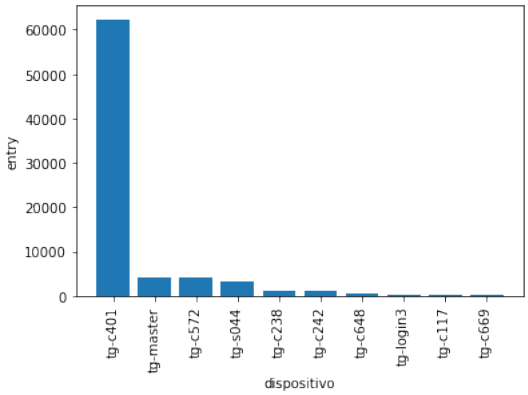
Tupla	Number of Entries
tuple_299	9825
tuple_298	8590
tuple_324	5883
tuple_325	5065
tuple_114	4049
tuple_312	3884

Figura 5.3: Distribuzione delle entry nelle tuple

Trovate quindi le tuple che contengono il maggior numero di entry, si passa allo studio dell'intorno della tupla, per verificare se vi sono interessamente di un singolo componente che si riterrebbe quindi essere il bottleneck del sistema. Per fare ciò, come evidenziato dal grafico, si nota che la maggioranza delle entry è contenuta tra la tupla 298 e la tupla 332. In questo intervallo sono comprese 60151 istanze, che compongono circa il 74.5 % di tutto il file di log. É evidente quindi che in questo intervallo temporale sono concentrati la maggioranza degli errori.
Analizzando nel dettaglio tali tuple, si vede come la maggioranza degli errori è dovuta al nodo computazionale numero 401 (*tg-c401*), trend che però si verifica all'interno di tutto il file log.
Tramite il seguente script in Python si è quindi analizzata la distribuzione degli errori di tutto il file log che sono riconducibili al nodo 401.

```python
import matplotlib.pylab as plt
import pandas as pd

with open(perc+'/Stat_tot_Merc.txt') as f:
    data = [_.rstrip('\n') for _ in f.readlines()[13:23]]
print(data)

statistics = {}

for i in range(len(data)):
    stat = data[i].split(' ')
    statistics[stat[0]] = int(stat[1])
```

```
print(statistics)


disp = statistics.items()
x, y = zip(*disp)

plt.bar(x,y)
plt.xlabel("dispositivo")
plt.xticks(rotation=90)
plt.ylabel("entry")
plt.show()
```



Tale esecuzione fa emergere che il nodo tg-c401 è responsabile di circa il 77.1 % degli errori nel file di log, ed è quindi plausibile che sia un probabile bottleneck per il sistema. Per avere una ulteriore conferma dell'ipotesi di bottleneck, è possibile verificare che nelle tuple con il maggior numero di entry, come l'intervallo [298,332] identificato in precedenza, siano contenute entry appartenenti ad un intervallo temporale relativamente breve. Andando infatti a convertire i timestamp Unix della tupla 298 e delle tupla 332, si scopre che gli errori raggruppati in tali tuple fanno riferimento ad un intervallo temporale estremamente breve, in relazione all'intervallo temporale totale del file di log. Convertendo tali tempi si ottengono i tempi descritti in tabella.

Descrizione	**Timestamp Unix**	**Data effettiva**
Prima entry del log	1167637660	01/01/2007 ore 08:47:40
Prima entry tupla 298	1171321113	12/02/2007 ore 23:58:33
Ultima entry tupla 332	1171482448	14/02/2007 ore 20:47:28
Ultima entry del log	1174943943	23/03/2007 ore 23:19:03

Come evidenziato dalla tabella, l'intervallo temporale a cui fanno riferimento le entry evidenziate è circa il 2% dell'intero dataset, cosa che avvalora maggiormente l'ipotesi di bottleneck del nodo c401.

5.3.3 Analisi Troncamenti e Collisioni

Per confermare la bontà del tupling eseguito sul sisteme Mercury, è stato eseguito uno script per la verifica del numero di possibili collisioni e troncamenti ottenuti a valle di tale raggruppamento. Lo script è indicato in basso:

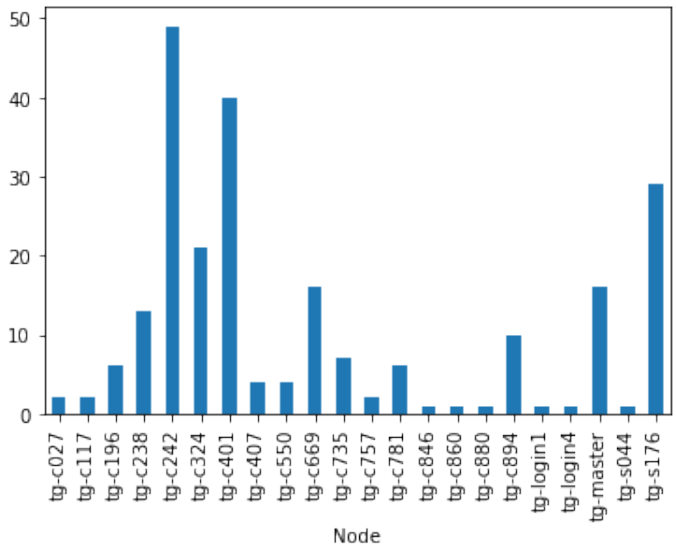
```
trunc_countMerc = 0
nodiMerc = pd.DataFrame()

for t in istanzeMerc['Tupla'].unique():
    if t != 508:
        mask = (istanzeMerc['Tupla']==t)
        mask_next = (istanzeMerc['Tupla']==t+1)
        istanze_filtrate = istanzeMerc.loc[mask]
        istanze_filtrate_next = istanzeMerc.loc[mask_next]
        if (istanze_filtrate.loc[istanze_filtrate.shape[0]-1, :].Node
        == istanze_filtrate_next.loc[0, :].Node):
            trunc_countMerc = trunc_countMerc + 1
            print('Troncamento : tupla '+str(t)+' - tupla '+str(t+1) +
                  '--- Nodo: '+ str(istanze_filtrate.loc
                  [istanze_filtrate.shape[0]-1, :].Node))
            nodiMerc = nodiMerc.append(istanze_filtrate.loc
            [istanze_filtrate.shape[0]-1, :])
    else:
        print("il file è finito")
print('Troncamenti : ' + str(trunc_countMerc))
print('Percentuale di troncamenti: '+str(trunc_countMerc/508*100)+' %')
```

A valle dell'esecuzione si ottiene il seguente output:

$$\text{Troncamenti : } 233$$

$$\text{Percentuale di troncamenti : } 45.87\%$$

Come è possibile notare, il numero di troncamenti è elevato, ma se si analizzano i troncamenti stessi, è possibile notare che la maggior parte coinvolge in particolare solo alcuni nodi chiave come c401, c242 e s176 ad esempio, che concorrono anche alla maggior parte delle entry nel file di log, il che permette di accettare anche il gran numero di troncamenti ottenuti. La distribuzione dei troncamenti per nodo è mostrata in figura.



Per quanto riguarda invece le collisioni, è stato utilizzato il seguente script:

```
coll_countMerc = 0
collMerc = {}

for t in istanzeMerc['Tupla'].unique():
    if t != 508:
        mask = (istanzeMerc['Tupla']==t)
        istanze_filtrate = istanzeMerc.loc[mask]
        if (len(istanze_filtrate.Node.unique())>1):
            coll_countMerc = coll_countMerc+1
            print('Collisione -- tupla: '+ str(t))
            collMerc[t]=istanze_filtrate.Node.unique()
    else:
        print('File finito')
print('Collisioni : ' + str(coll_countMerc))
print('Percentuale di collisioni: '+str(coll_countMerc/508*100)+' %')
```

L'esecuzione di tale script ha prodotto il seguente output:

Collisioni : 47

Percentuale di collisioni : 9.25%

Come è possibile dedurre dalla scarsa percentuale di collisioni, il valore scelto di finestra è da ritenersi valido

5.3.4 Fase di Data Analysis del sistema

Terminata la fase di data manipulation è possibile proseguire con l'analisi dei dati vera e propria. L'obiettivo di questa fase è quello di estrarre dai dati alcune informazioni relative alla reliability del sistema. Per prima cosa è possibile estrarre dai dati la distribuzione di probabilità empirica del **time to failure**, ovvero del tempo oltre il quale si verifica un fallimento del sistema. A tal fine, si applicano ai dati relativi ai tempi interarrivo una serie di test di **Bontà di adattamento** (*GoF*), test statistici che verificano la presenza di una distribuzione nota nei dati. Tali test sono stati eseguiti tramite la libreria di Curve Fitting di Matlab, attraverso tale script:

```
load interarrivals_merc.txt
[y,t] = cdfcalc(interarrivals_merc);
empTTF=y(2:size(y,1));
empRel=1-empTTF;
plot(t,empTTF,'.-b',t,empRel,'.-r');
xlabel('time [s]'); ylabel('p-value');
legend('empirical TTF', 'empirical Rel');
```

Tale esecuzione porta a definire il grafico del TTF empirico e della relativa reliability empirica, come da figura seguente.

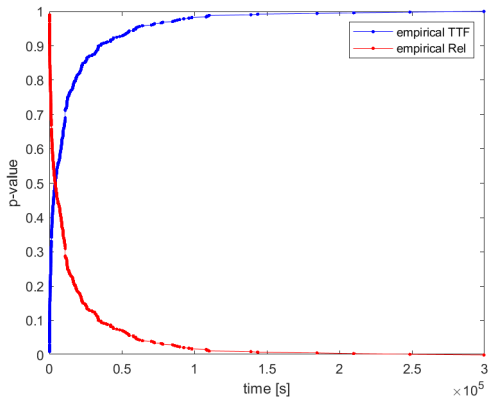
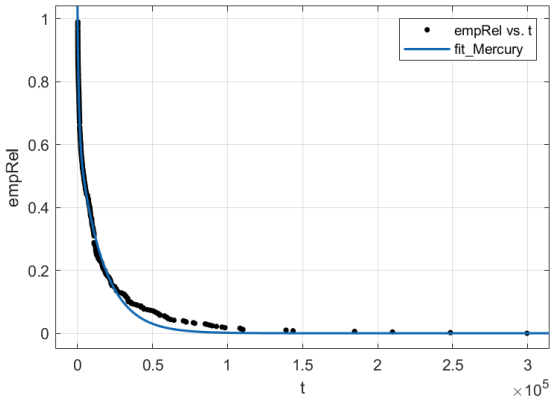


Figura 5.4: Distribuzioni empiriche - Sistema Mercury

Da tale distribuzione empirica è possibile, attraverso lo strumento di Curve Fitting, testare la bontà di adattamento della curva a varie distribuzioni. Quella evidenziata come curva di fitting migliore è un **esponenziale di secondo grado**, avente i coefficienti della tabella a lato.
Per evidenziare la bontà di tale approssimazione, in prima analisi è possibile verificare il valore di alcuni parametri:

- l'indicatore **SSE** rappresenta la *Sum of Square Errors*, somma quadratica degli errori, e per avere una buona approssimazione deve essere prossimo a zero. Nel caso della GoF eseguita tale valore è pari a 0.14;



$$fit - Mercury(x) = \alpha \cdot e^{\beta \cdot x} + \gamma \cdot e^{\delta \cdot x}$$

α	β	γ	δ
0.3922	-0.001036	0.6297	-6.07 $\cdot 10^{-5}$

- l'indicatore **R-square** misura il legame tra la variabilità dei dati e la correttezza del modello statistico utilizzato, ed è evidente che deve essere il più possibile simile ad 1. Nel caso di tale modello l'R-quadro è pari a 0.9961.

Tali parametri indicano già una corretta approssimazione del modello esponenziale alla curva empirica, ma per avere una conferma definitiva è possibile eseguire il test non parametrico di **Kolmogorov-Smirnov**: usando tale test è possibile affermare con un livello di significatività di 0.05 che è possibile accettare o rigettare l'ipotesi che i dati provengono dalla distribuzione scelta, avente i parametri riportati in precedenza. Dai risultati del test in tabella si nota come è possibile

H- value	**p-value**	**k-value**
0	0.6666	0.0472

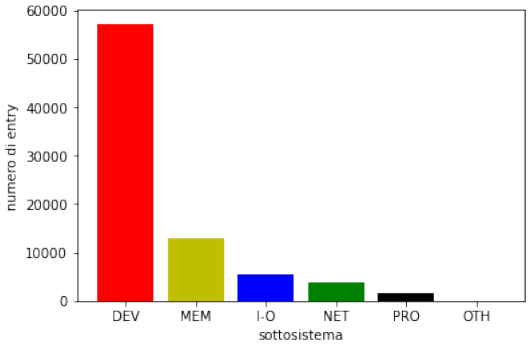
accettare l'ipotesi nulla e confermare la bontà dell'adattamento della reliability empirica ad una reliability esponenziale. É possibile trarre alcune conclusioni da tale test: un tipo di TTF empirica esponenziale è tipicamente associato ad un guasto a livello hardware o firmware, tipicamente semplice, mentre altre tipologie di curve empiriche possono essere associate a degenerazione di un componente, o a guasti di tipo software, o a guasti di natura più complessa. Avendo evidenziato un tipo di curva empirica esponenziale, è facile intuire che probabilmente la reliability del sistema Mercury dipende principalmente da errori e fallimenti che sono riconducibili all'hardware. Tale teoria viene confermata dal fatto che la maggioranza delle entry presenti nel file di log è appartenente alla categoria DEV, che rappresentano errori legati a schede PCI e alle periferiche in generale.

5.3.5 Fase di Data Manipulation dei componenti

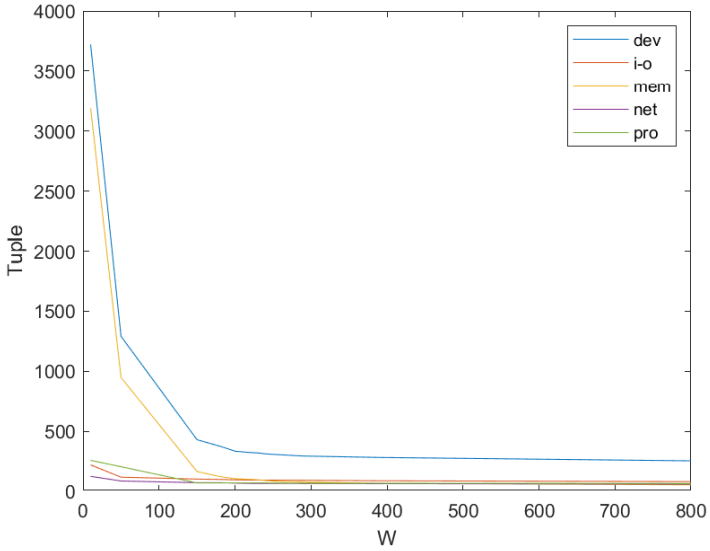
Per studiare il comportamento dei vari componenti del sistema Mercury, si è scelto di studiare la reliability in funzione del componente che ha generato il fallimento. Come prima analisi si è deciso di graficare la distribuzione degli errori del sistema Mercury per componente fallito. Tale analisi è stata fatta utilizzando il seguente script Python.

```
with open(perc+'/Stat_tot_Merc.txt') as f:
```

```
    data = [_.rstrip('\n') for _ in f.readlines()[5:11]]

stat_cat = {}

for i in range(len(data)):
    stat = data[i].split(' ')
    stat_cat[stat[0]] = int(stat[1])

cat = stat_cat.items()
x, y = zip(*cat)

plt.bar(x,y, color=['r', 'y', 'b', 'g', 'k'])
plt.xlabel("sottosistema")
plt.ylabel("numero di entry")
plt.show()
```



Come è possibili notare, la maggioranza degli errori è dovuta a componenti hardware, data la tipologia DEV predominante rispetto alle altre. Per analizzare la reliability dei componenti di ogni tipo di sottosistema, si è studiato l'andamento dell'algoritmo di tupling al variare del valore della finestra di coalescenza, facendo quindi una sensitivity analysis dei file di log raggruppati per categoria. Si sono ottenute le seguenti curve.



Tramite la sensitivity analysis si è trovato che c'è un valore di finestra di coalescenza adatto sia per MEM che per DEV, così come per PRO, e tale valore è 200; per le altre categorie utilizzare una finestra di coalescenza pari a 200 implicherebbe l'incorrere di un numero elevato di troncamenti, cosa che renderebbe troppo pessimistica l'approssimazione della reliability. Tale risultato non sorprende in quanto DEV e MEM sono i sottocomponenti maggiormente soggetti a guasti. Quindi si è deciso di scegliere delle finestre di coalescenza differenziate per effettuare la fase di Data Analysis.

Sottosistema	**Finestra di Coalescenza**
DEV	200
MEM	200
I-O	100
NET	100
PRO	200

5.3.6 Fase di Data Analysis dei componenti

Successivamente allo studio sulla finestra di coalescenza migliore, si sono studiate le reliability dei singoli componenti, graficando l'andamento degli interarrivi, esattamente come fatto in precedenza per l'analisi del sistema completo. Sono stati quindi ottenuti i risultati in figura: come era facile pronosticare, il sottosistema DEV è quello che espone la peggiore reliability, vista la gran mole di entry presenti nel log dovuti a tale sottosistema.

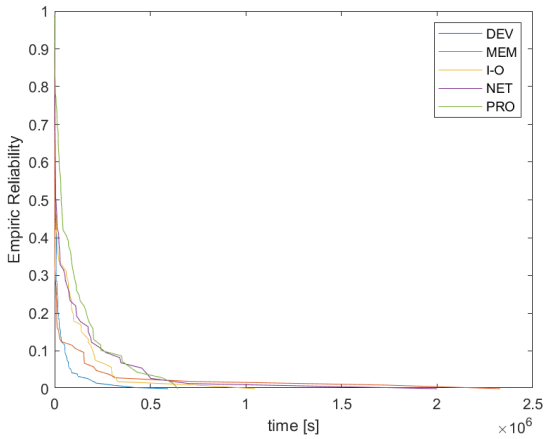


Figura 5.5: Reliability empiriche per categoria - Mercury

Calcolate le tuple è possibile quindi andare a testare la bontà di adattamento di ogni sottosistema, graficandone le distribuzioni teoriche. Si ottengono quindi i risultati in figura.

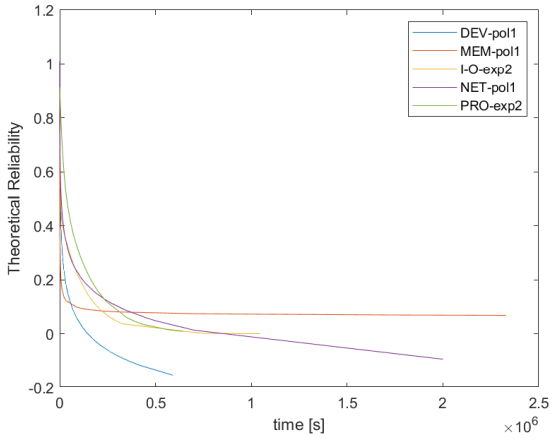


Figura 5.6: Reliability teoriche per categoria - Mercury

A valle dell'ottenimento di tali distribuzioni teoriche, si vanno ad effettuare i test per la verifica dell'adattamento.

	Reliability teorica	**H-value**	**p-value**	**k-value**
DEV	polinomiale di primo grado	0	0.3802	0.0736
MEM	polinomiale di primo grado	0	0.8938	0.0800
I-O	esponenziale di secondo grado	0	0.9659	0.0680
NET	polinomiale di primo grado	0	0.9998	0.0548
PRO	esponenziale di secondo grado	0	0.9913	0.0725

La totalità dei test parametrici effettuati accetta l'ipotesi nulla con un grado di significatività pari a 0.05.

5.3.7 Fase di Data Manipulation dei nodi

Oltre allo studio sui sottosistemi, si è scelto di analizzare anche l'andamento degli errori considerando la tipologia del nodo che ha generato tale entry. Di seguito la distribuzione delle entry per nodo, ottenuta tramite script Python.

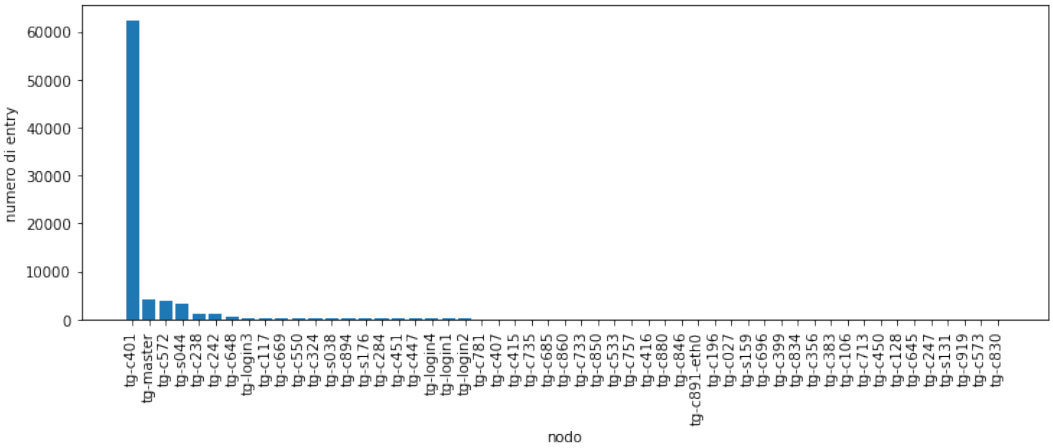
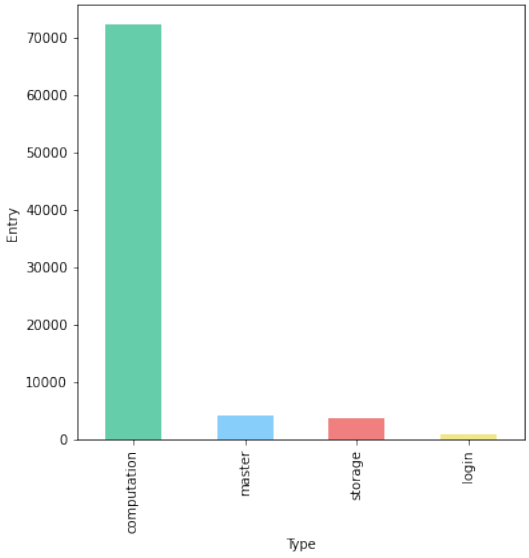


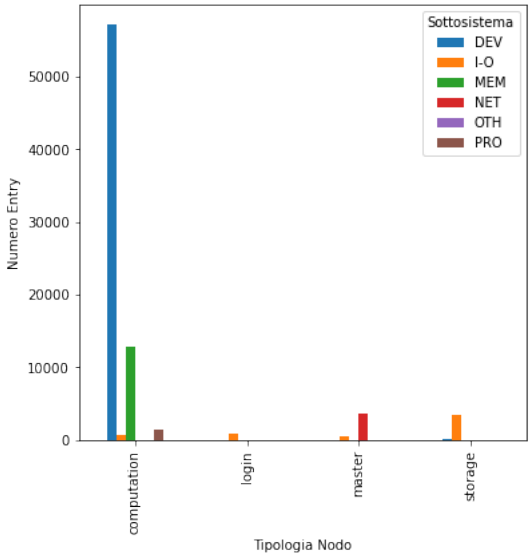
Figura 5.7: Reliability teoriche per categoria - Mercury

Per verificare la tipologia di nodo che concorre al maggior numero di entry, si è scelto di raggruppare i nodi per tipologia, identificate tramite script Python, e studiare la distribuzione delle entry al variare del tipo di nodo.



Come si evince dalla figura, i nodi di tipo computazionale sono i nodi che concorrono alla maggior parte degli errrori. Un risultato del genere era prevedibile in quanto la maggioranza degli errori sono di tipo DEV, e i nodi computazionali espongono la quasi totalità degli errori sul sottosistema DEV, come è possibile vedere dalla figura in basso.

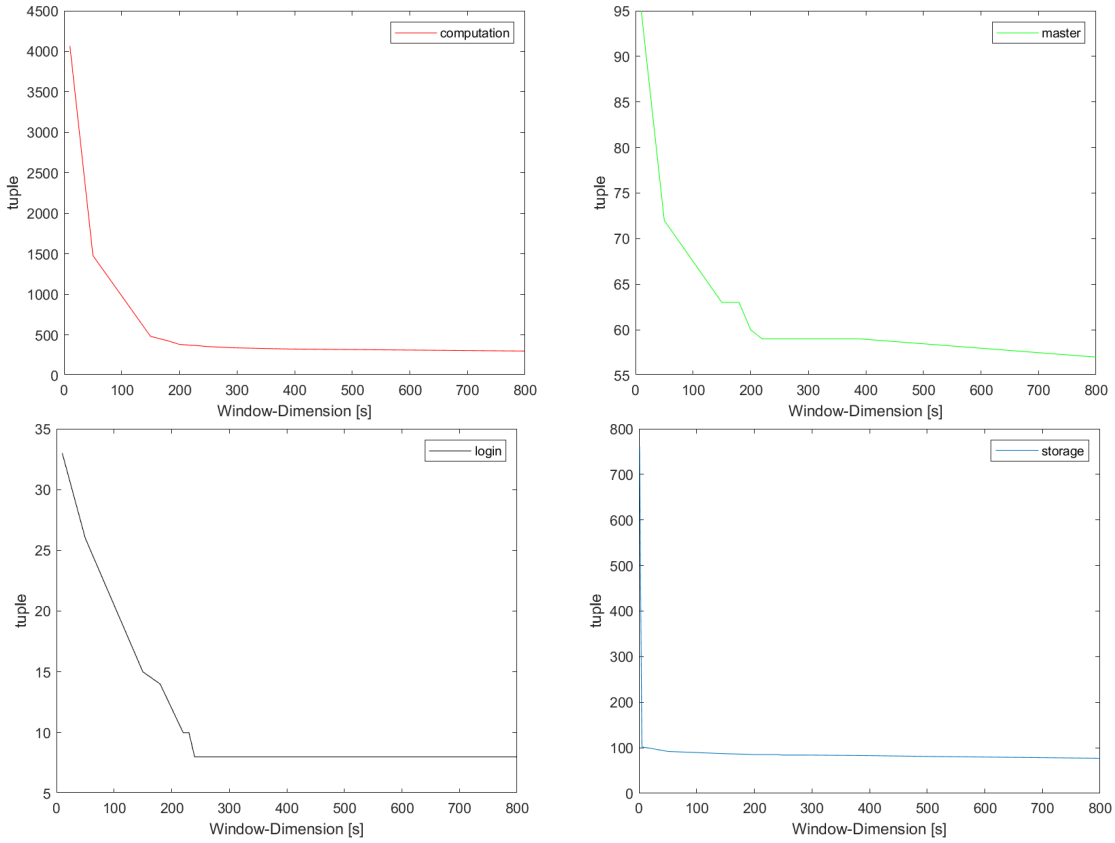
Nel caso di nodi di diversa tipologia, è possibile verificare se per nodi di tipo differente sia possibile



utilizzare una finestra di coalescenza unica. Tale risultato è ottenibile effettuando la sensitivity analysis sulle varie categorie di nodi. Tramite script Python sono state realizzate delle maschere al file log per raggruppare entry appartenenti alla stessa categoria di nodi.

```
perc = '/FFDA/ffdatoolset_21'

df_mask_comp=df_mercury['Type']=='computation'
df_comp = df_mercury[df_mask_comp]
df_comp.to_csv(perc+'/MercuryComputation.txt')

df_mask_master=df_mercury['Type']=='master'
df_master = df_mercury[df_mask_master]
df_master.to_csv(perc+'/MercuryMaster.txt')

df_mask_login=df_mercury['Type']=='login'
df_login = df_mercury[df_mask_login]
df_login.to_csv(perc+'/MercuryLogin.txt')

df_mask_storage=df_mercury['Type']=='storage'
df_storage = df_mercury[df_mask_storage]
df_storage.to_csv(perc+'/MercuryStorage.txt')
```

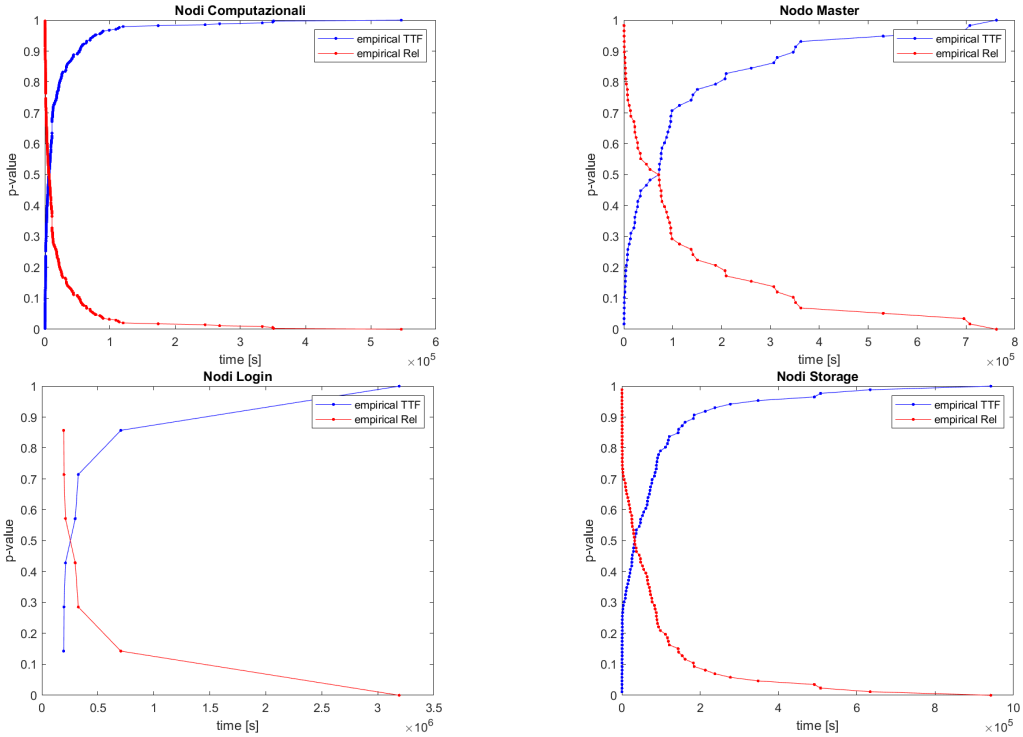
A valle di tale filtraggio, è stata eseguita la fase di analisi per trovare la migliore finestra di coalescenza. I grafici in figura mostrano come non sia possibile definire una finestra di coalescenza univoca, ma è necessario differenziare la scelta di quest'ultima per evitare una sovrastima o una sottostima delle reliability empiriche. I risultati con le relative finestre di coalescenza sono indicati in basso.



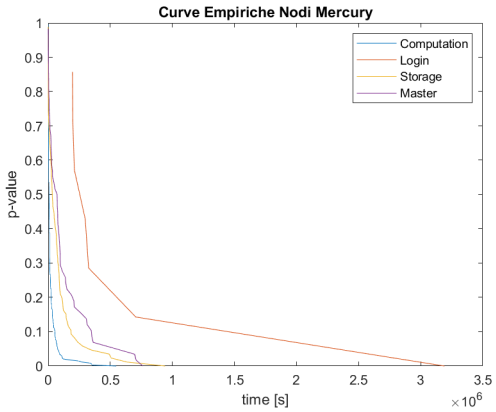
Tipologia di nodi	Finestra
Computation	300
Master	300
Login	300
Storage	150

5.3.8 Fase di Data Analysis dei nodi

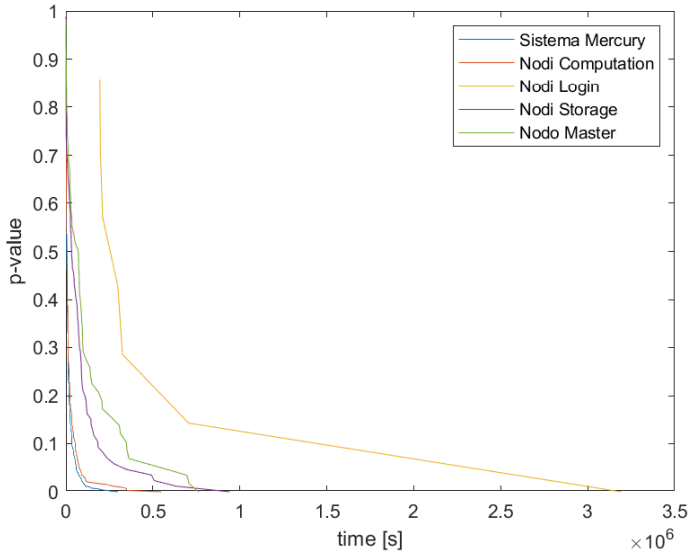
Scelte le finestre di coalescenza, si è eseguito l'algoritmo di tupling, e tramite Matlab si è graficato l'andamento delle curve empiriche ottenute. I suddetti andamenti sono indicati in figura.



Confrontando le reliability è possibile affermare che i nodi computazionali sono i nodi che hanno la peggiore reliability all'interno del sistema Mercury, mentre i nodi di login espongono la reliability migliore. In figura gli andamenti a confronto. Per verificare la tipologia di nodi che

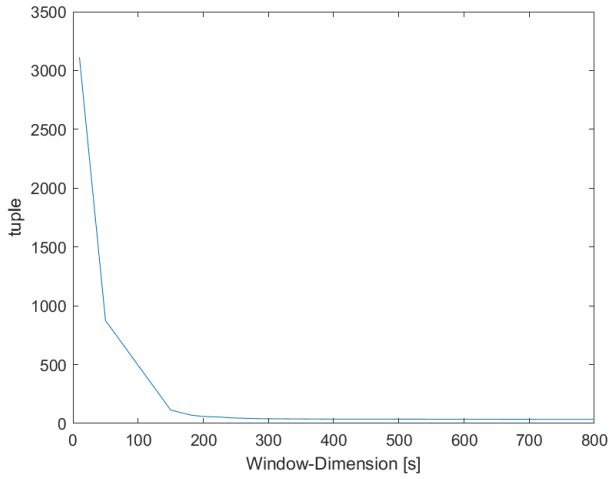


influenza maggiormente la reliability del sistema, si è scelto di confrontare quest'ultima con le curve appena trovate. Se una tipologia di nodo presentasse un andamento di reliability simile confermerebbe l'ipotesi di dipendenza della reliability del sistema da tali componenti. In figura è possibile notare che i nodi computazionali esprimono una reliability comparabile a quella del sistema, ed è quindi facile intuire che siano loro il fattore di maggiore influenza sulla reliability del sistema Mercury.

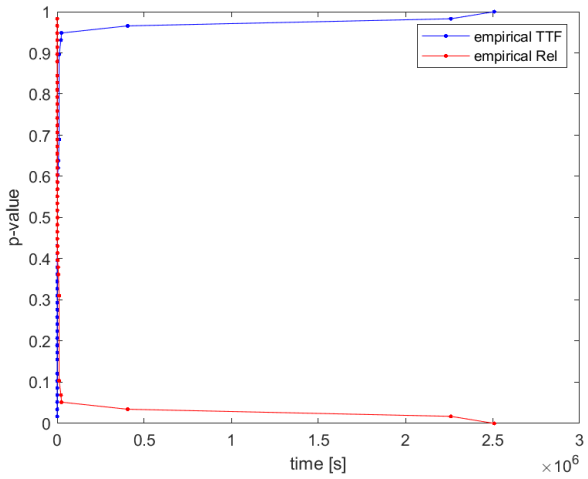


Approfondimento nodo tg-c401

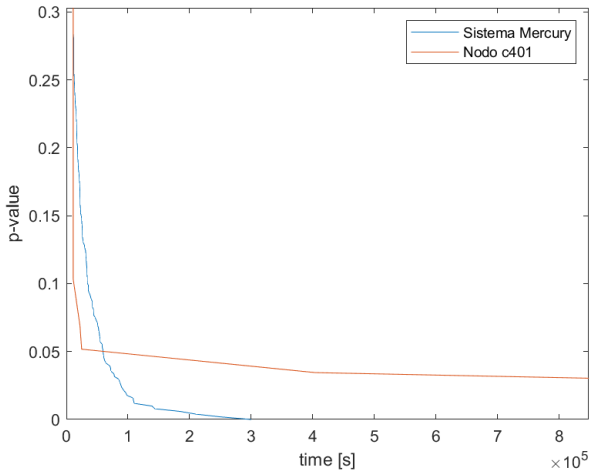
Avendo evidenziato un particolare nodo che concorre ad una enorme quantità di errori all'interno del file di log, si è ritenuto necessario studiare l'andamento di tale nodo per verificare se fosse effettivamente un bottleneck del sistema Mercury. Per verificare l'ipotesi, si eseguono le operazioni di manipulation e di analysis dei dati di tale nodo al fine di individuarne una reliability empirica. La fase di Data Manipulation ha evidenziato che la finestra di coalescenza migliore si ottiene al valore 200, ottenendo un totale di 59 tuple, come mostrato dal seguente grafico.



Trovata la finestra di coalescenza ottimale, si passa all'esecuzione dello script di tupling, e alla costruzione dei grafici delle curve empiriche, mostrati in figura.



La curva di reliability mostra un brusco calo per valori di tempo bassi, e confrontata con la curva di reliability dell'intero sistema, si nota che per un certo valore di tempo si ha che il nodo ha una reliability peggiore dell'intero sistema. Questo di fatto conferma l'ipotesi di bottleneck del nodo c401.



5.3.9 Domande

Si possono usare le stesse finestre di coalescenza in diversi nodi e in diverse categorie di nodi?

Come si è potuto osservare dall'analisi, effettuando un raggruppamento sia per tipologia di nodo che per categoria di nodo, non è possibile definire una finestra di coalescenza univoca per i vari livelli di tale categorizzazione, in quanto per alcuni si avrebbe una stima troppo pessimistica di tale valore, con conseguente aumento di probabilità di incorrere in troncamenti.

Cosa si può dire della relazione tra l'affidabilità del sistema e l'affidabilità dei nodi?

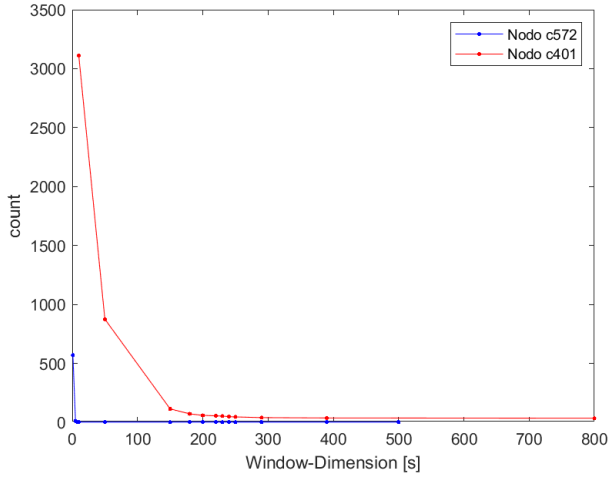
Dall'analisi effettuata sulle tipologie di nodi del sistema Mercury, è emerso che i nodi che contribuiscono maggiormente all'andamento della reliability del sistema sono i nodi computazionali, avendo essi una curva di reliability che espone un andamento simile a quella del sistema. In ogni caso l'intero sistema presenta una reliability che è peggiore di ogni categoria di nodo, quindi non è possibile etichettare una singola categoria di nodi come estremamente problematica per il sistema, fatta eccezione per il nodo c401, che espone come visto una reliability che è peggiore anche dell'intero sistema per un certo tempo t.

Esistono colli di bottiglia dell'affidabilità (cioè, i principali contributori al numero totale di fallimenti)?

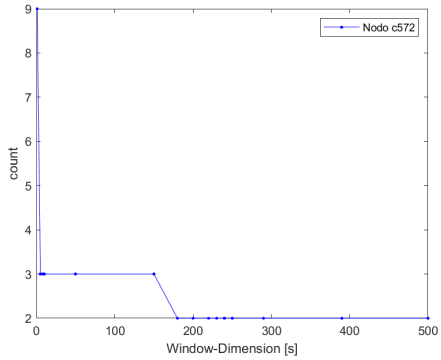
Come spiegato per la questione precedente, è evidente che il nodo c401 rappresenta un ipotetico collo di bottiglia per l'affidabilità avendo esso una reliability peggiore del sistema stesso. Ciò conferma l'ipotesi di bottleneck a priori fatta considerando il numero alto di entry nel log appartenenti a tale nodo.

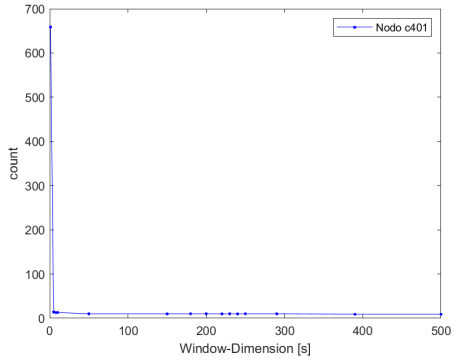
Nodi funzionali simili (per esempio, due nodi di calcolo) presentano parametri di affidabilità simili?

Per confrontare le reliability di nodi simili si è scelto di considerare due nodi computazionali, in particolare il nodo c401 e il nodo c572, che presentano il maggior numero di entry nel log. A valle dell'analisi sulla finestra di coalescenza si è visto che, a differenza del nodo c401, il nodo c572 presenta un valore di finestra di coalescenza decisamente più basso. In particolare poi, superato il valore di 1 secondo di finestra di coalescenza, non ha più senso eseguire l'algoritmo di tupling per il nodo, visto che si otterranno solo 2 tuple. In figura gli andamenti delle finestre di coalescenza.

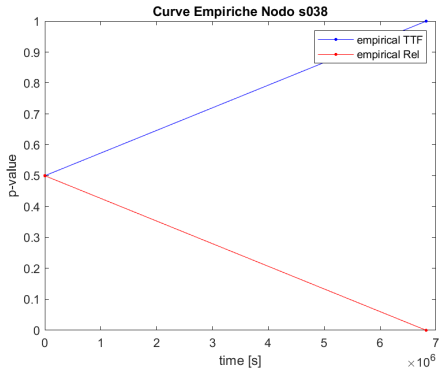


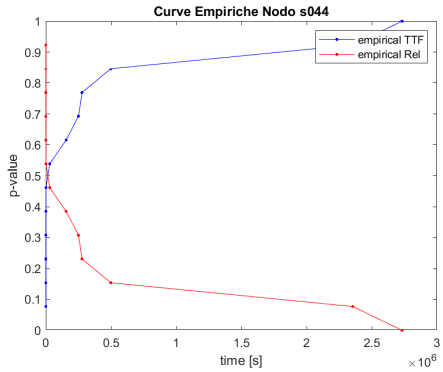
Non è quindi possibile comparare gli andamenti della reliability nel caso di tali nodi, e si è quindi deciso di passare all'analisi di altri due nodi, in particolare i nodi storage s044 e s038. In questo caso, è possibile definire una finestra di coalescenza univoca, pari a 5, come si può vedere dai grafici.





A valle della fase di tupling, anche per il nodo s038, così come per il nodo computazionale analizzato in precedenza, sono state create soltanto 3 tuple, con una conseguente reliability empirica a cui non è possibile attribuire alcun andamento preciso, come si vede in figura. In definitiva non è possibile comparare le reliability anche in questo caso.





Esiste una relazione tra i tipi di errore e il nodo?

Dall'analisi fatta sugli errori raggruppati per tipologia di nodo, è possibile vedere che a nodi computazionali corrisponde la quasi totalità degli errori di tipo DEV e MEM, mentre per le altre tipologie di nodi la distribuzione delle categorie di errore è leggermente più bilanciata. Inoltre gli errori originati dai nodi di tipo storage e login riguardano principalmente errori di I-O, mentre per gli errori di tipo NET il principale responsabile è il nodo master.

5.4 Blue Gene/L

5.4.1 Descrizione del sistema

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

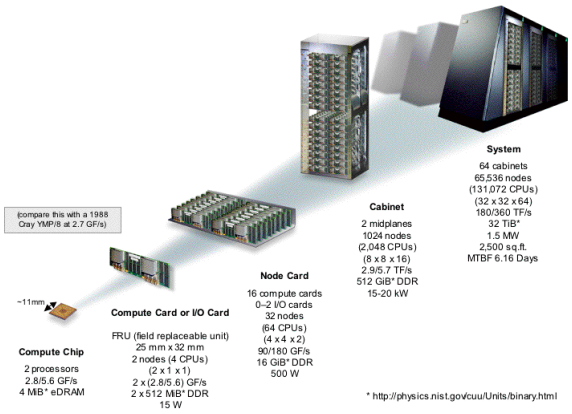


Figura 5.8: Architettura di Blue Gene/L

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

- *timestamp*: intero in formato UNIX;
- *nodo di origine*: stringa del tipo Rxx-Mx-Nx indicante il nodo in cui si è verificato l'errore. In particolare, sono indicati **rack**(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 indicante il messaggio relativo all'errore.

5.4.2 Fase di Data Manipulation del sistema

Come fatto per il sistema Mercury, tramite bash-scripting, si vanno a definire le tuple al variare della finestra di coalescenza, per la scelta della migliore dimensione di quest'ultima. Si ottengono i seguenti risultati.

Come evidenziato dalla figura, il punto di knee della curva si trova in corrispondenza di un valore di finestra pari a 150, quindi è ragionevole considerare come valore di finestra ottimale 200, così

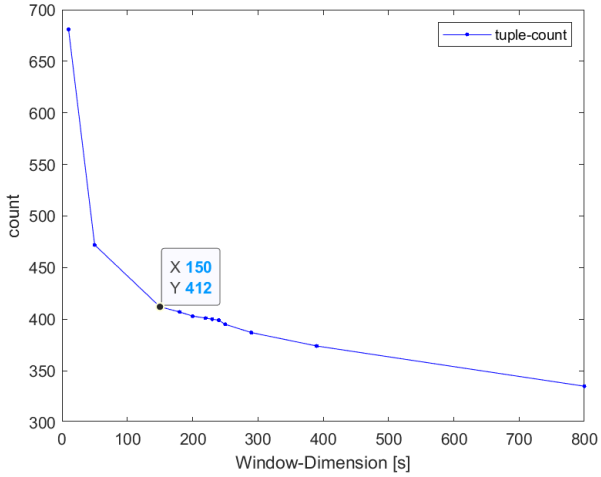
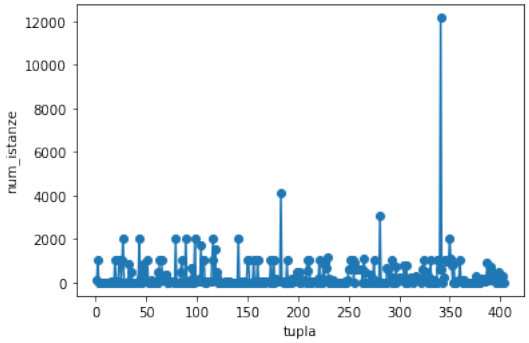


Figura 5.9: Sensitivity Analysis - BGL

come accade per Mercury. Per i due sistemi è quindi possibile stabilire una finestra di coalescenza equivalente.

Bottleneck

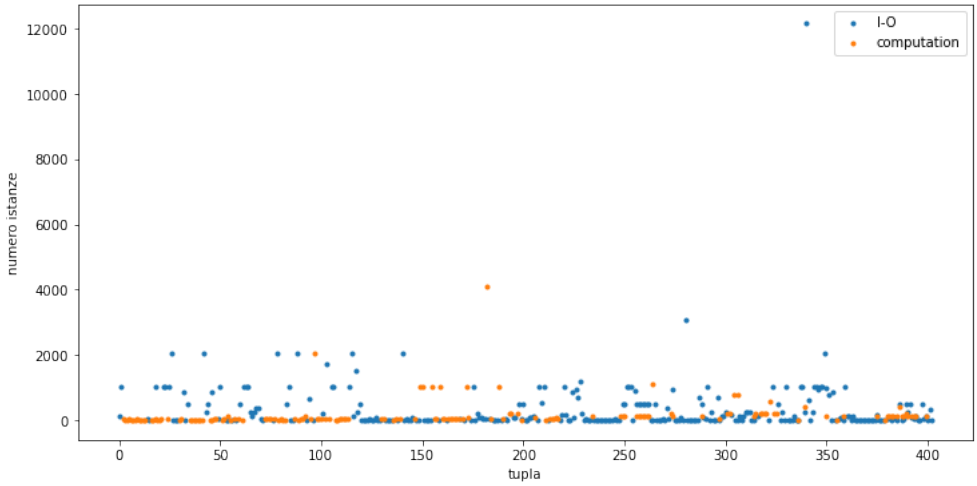
Così come fatto per il sistema Mercury, si è deciso di analizzare la distribuzione delle entry all'interno della tuple, per verificare se ci siano particolari intervalli temporali in cui capita il maggior numero di errori. Seguendo la stessa strategia attuata per Mercury, si sono evidenziate le seguenti distribuzioni.



Tupla	Number of Entries
tuple_341	12152
tuple_183	4096
tuple_281	3071
tuple_27	2048
tuple_43	2048
tuple_79	2048

Figura 5.10: Distribuzione delle entry nelle tuple

Rispetto al sistema Mercury, si può vedere come la distribuzione nelle entry sia estremamente più omogenea. Non c'è dunque un intervallo temporale interessante da analizzare. Per capire meglio se qualche sottoparte del sistema BGL sia un ipotetico bottleneck, è possibile evidenziare all'interno delle tuple la tipologia di errori, vista la possibilità di separare errori dovuti alle card di I-O ed errori relativi alle card di tipo computazionale, etichettando come errore di I-O un entry che riguarda la card J18, responsabile delle operazioni di I-O all'interno del sistema. Graficando l'andamento delle due tipologie di errori all'interno delle tuple, si ottiene il seguente andamento.



Una conclusione che è possibile trarre da tale analisi è che la maggioranza delle entry del log appartengono ad errori classificati come di tipo I-O. Siccome l'omogeneità dei due tipi di entry nelle varie tuple non permette di evidenziare alcun colpevole principale, è ragionevole pensare che un ipotetico bottleneck sia l'intero sistema di I-O di BGL. Infatti, le entry relative a operazioni I-O sono circa il 79.59% del file di log, mentre quelle computazionali sono circa il 20.41%. Altre considerazioni riguardo tale ipotesi saranno tratte durante le prossime fasi dell'analisi.

5.4.3 Analisi Troncamenti e collisioni

Come fatto per il sistema Mercury, si vanno a verificare il numero di troncamenti e collisioni ottenuti a valle della fare di data manipulation. Modificando lo script Python per Mercury, si è ottenuto il seguente risultato per quanto riguarda i troncamenti:

$$\text{Troncamenti} : 21$$

$$\text{Percentuale di troncamenti} : 5.21\%$$

Il numero di troncamenti risulta estremamente basso, sintomo di una buona scelta della finestra di coalescenza.
Per quanto riguarda le collisioni tra i nodi si ottiene invece il seguente risultato:

$$\text{Collisioni} : 267$$

$$\text{Percentuale di collisioni} : 66.25\%$$

Il numero di collisioni sembra essere decisamente eccessivo, ma le collisioni sono state verificate per nodo.
Si è scelto quindi di considerare una collisione solo se all'interno della stessa tupla sono inseriti errori relativi al sistema di I-O e al sistema computazionale, visto che la maggior parte delle collisioni trovate in precedenza erano dovute a differenti card computazionali o di I-O presenti all'interno di

una stessa tupla. A valle di tale modifica, si ottiene il seguente risultato:

Collisioni : 39

Percentuale di collisioni : 9.68%

Il numero di collisioni così considerate scende quindi a valori decisamente più accettabili, ed è ragionevole confermale la scelta del valori di finestra anche per il sistema BG/L.

5.4.4 Fase di Data Analysis del sistema

A valle dell'analisi di sensitività, si passa allo studio della reliability empirica, tramite il medesimo script utilizzato per il sistema Mercury. Dall'esecuzione dello script si ottengono le seguenti distribuzioni empiriche.

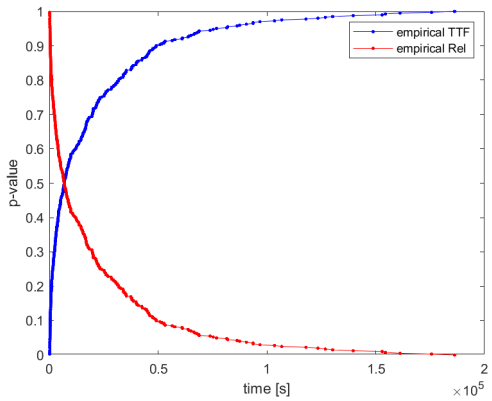
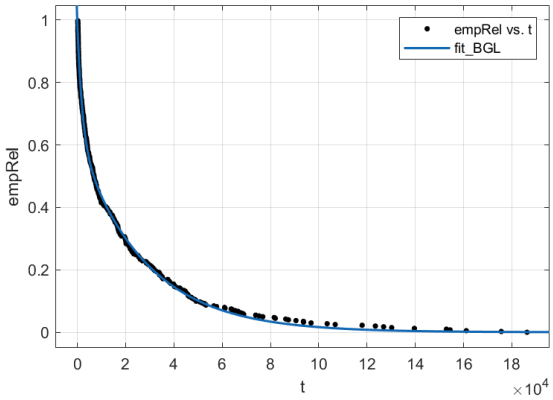


Figura 5.11: Distribuzioni empiriche - Sistema BGL

Da tale distribuzione empirica è possibile, attraverso lo strumento di Curve Fitting, testare la bontà di adattamento della curva a varie distribuzioni. Quella evidenziata come curva di fitting migliore è un **esponenziale di secondo grado**, avente i coefficienti della tabella a lato.



$$fit - BGL(x) = \alpha \cdot e^{\beta \cdot x} + \gamma \cdot e^{\delta \cdot x}$$

α	β	γ	δ
0.3675	-0.0004837	0.6222	-3.63 $\cdot 10^{-5}$

Come fatto per Mercury, per evidenziare la bontà di tale approssimazione, si verifica il valore di alcuni parametri:

- l'indicatore **SSE** risulta essere pari a 0.06929;
- l'indicatore **R-square** nel caso di tale modello è pari a 0.9978.

Tali parametri indicano già una corretta approssimazione del modello esponenziale alla curva empirica, ma per avere una conferma definitiva si esegue il test non parametrico di Kolmogorov-Smirnov; i risultati sono riportati in tabella.

H- value	**p-value**	**k-value**
0	0.8954	0.0406

Dai risultati del test in tabella si nota come è possibile accettare l'ipotesi nulla e confermare la bontà dell'adattamento della reliability empirica ad una reliability esponenziale. Come evidenziato per il sistema Mercury, anche il supercomputer BGL presenta una reliability empirica di tipo esponenziale, e quindi i suoi guasti possono essere ricondotti a tipi di fallimenti hardware, esattamente come avveniva per Mercury.

5.4.5 Fase di Data Manipulation dei nodi

Così come fatto per il sistema Mercury, si è deciso di effettuare un analisi dei nodi anche del sistema BGL. La struttura del supercalcolatore non permette un analisi semplice, ma è necessario raggruppare il dataset secondo la sua struttura, e quindi evidenziando errori relativi ai vari *rack*, ai *midplane* e ai *nodi* appartenenti ai vari midplane. L'analisi si svolgerà quindi in fasi, secondo un livello di dettaglio sempre crescente.

Analisi dei Rack

Raggruppando le istanze per rack, si ottiene la seguente distribuzione di entry.

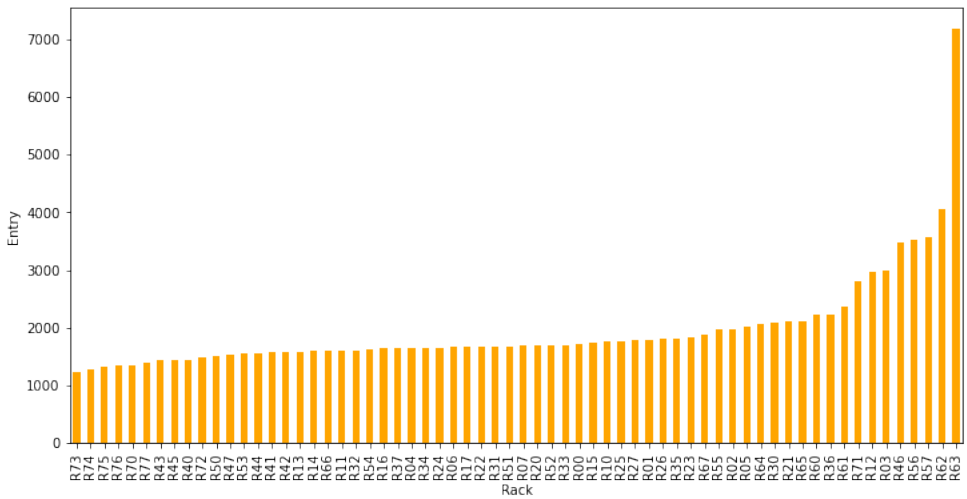


Figura 5.12: Distribuzione entry per Rack

Da tale grafico si evidenzia che alcuni rack sono responsabili della maggior parte delle entry del file di log, e tali rack sono *R63, R62, R57, R56, R46, R03, R71, R12*.

Analisi dei Midplane

Come specificato nella descrizione architetturale del sistema BG/L, ogni rack è composto da due midplane, indicati all'interno del log come *M0* e *M1*. Si è scelto di analizzare quindi i rack più critici identificati nella fase precedente, separando gli errori per midplane. Il risultato è riportato nel grafico seguente.

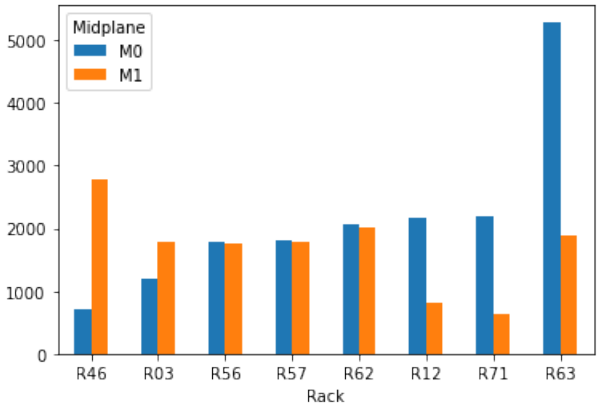


Figura 5.13: Distribuzione entry per Rack e Midplane

L'analisi porta alla luce uno squilibrio di errori tra i midplane per il rack R63, che presenta più errori sul midplane M0

Analisi dei Nodi

Trovati quindi i rack critici e la distribuzione delle entry per i midplane di tali rack, si è analizzata la distribuzione delle entry per nodo appartenente ai due midplane dei rack più critici. Tale analisi ha portato il seguente grafico.

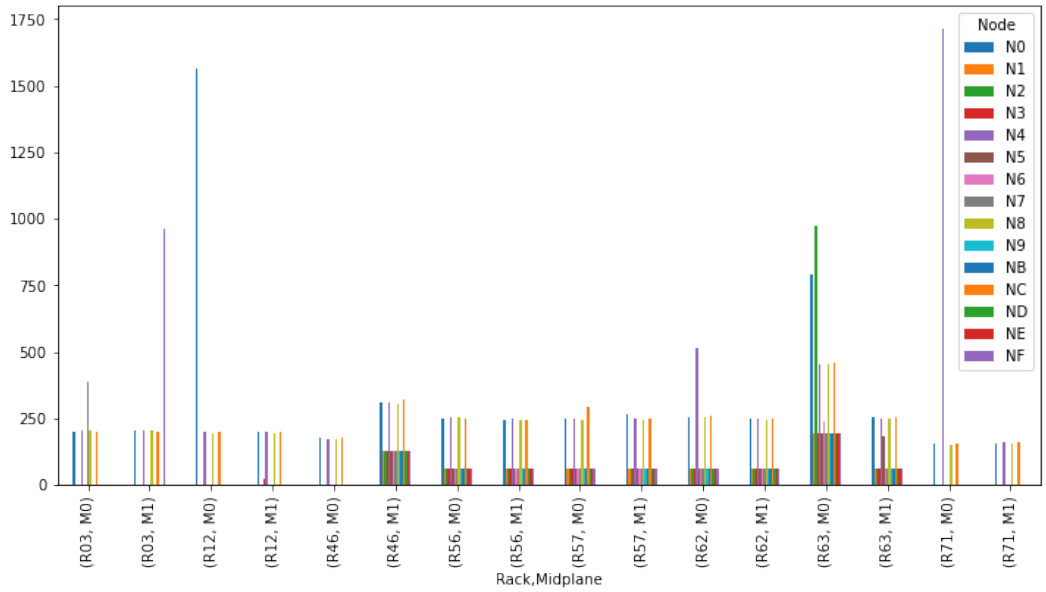


Figura 5.14: Distribuzione entry per Rack, Midplane e Nodo

Data la poca leggibilità di tale grafico dovuto al numero elevato di nodi da raffigurate, per evidenziare meglio le problematiche legate ai nodi e la loro relazione con i rispettivi rack e midplane, si è scelta una vista tabellare dei dati graficati. La tabella è riportata in basso.

Rack	Midplane	N0	N1	N2	N3	N4	N5	N6	N7	N8	N9	NB	NC	ND	NE	NF
R03	**M0**	202	0	0	0	207	0	0	390	204	0	0	202	0	0	0
	M1	205	0	0	0	205	0	0	0	205	0	0	202	0	0	960
R12	**M0**	1563	0	0	0	199	0	0	0	197	0	0	199	0	0	0
	M1	199	0	0	24	199	0	0	0	197	0	0	199	0	0	0
R46	**M0**	179	0	0	0	172	0	0	0	173	0	0	176	0	0	0
	M1	313	128	128	128	311	128	128	128	305	128	128	320	128	128	128
R56	**M0**	250	64	64	64	254	64	64	64	254	64	64	249	64	64	64
	M1	246	64	64	64	249	64	64	64	244	64	64	245	64	64	64
R57	**M0**	248	64	64	64	251	64	64	64	242	64	64	292	64	64	64
	M1	268	64	64	64	250	64	64	64	246	64	64	248	64	64	64
R62	**M0**	254	64	64	64	515	64	64	64	255	64	64	261	64	64	64
	M1	249	64	64	64	251	64	64	64	244	64	64	252	64	64	64
R63	**M0**	791	192	976	192	452	192	238	192	454	192	192	460	192	194	192
	M1	253	64	64	64	252	184	64	64	251	64	64	255	64	64	64
R71	**M0**	157	0	0	0	1716	0	0	0	152	0	0	156	0	0	0
	M1	158	0	0	0	160	0	0	0	156	0	0	160	0	0	0

Dalla tabella è possibile trarre alcune conclusioni che riguardano le entry appartenenti ai rack peggiori:

- **Dettaglio sui rack**: dalla tabella di evince che i rack R56, R57, R62 ed R63 presentano dei fallimenti bilanciati tra i nodi, mentre non è possibile dire lo stesso per gli altri rack analizzati. Avendo un bilanciamento di errori tra i nodi, si può dire che tali rack non cono la causa di soli errori di I-O, ma anche di errori di tipo computazionale. Per quanto riguarda poi il rack R63, che si ricorda essere quello che compare nel maggior numero di fallimenti, presenta un altra percentuale di errori sul midplane M0, così come avviene per il rack R46 sul midplane M1.

- **Dettaglio sui nodi**: analizzando la tabella dal punto di vista dei nodi, si nota che la maggiore concentrazione di quest'ultimi interessa i nodi N0, N4, N8 ed NC, che sono i nodi responsabili delle operazioni di I-O del sistema BG/L. Per tali nodi infatti non è possibile individuare un particolare rack a cui associare gli errori, ma essi sono uniformemente distribuiti tra i midplane dei vari rack analizzati. In particolare però, si evidenzia che il nodo N4 presenta il valore di errori su un singolo nodo più alto in corrispondenza del rack R71, così come avviene per i rack R12 per quanto riguarda il nodo N0.

Finestre di Coalescenza per Rack

A valle delle considerazioni fatte, si è pensato fosse necessario confrontare le reliability dei rack che sono stati analizzati. Si è quindi proceduto al calcolo delle finestre di coalescenza ottimale per ogni rack. L'analisi di sensitività per tali rack ha evidenziato gli andamenti graficati in basso.

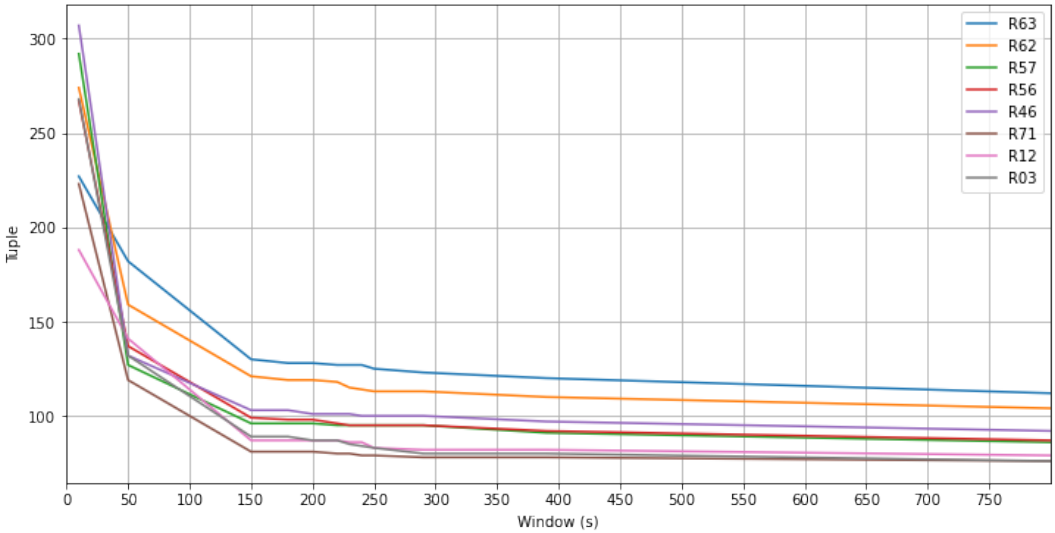


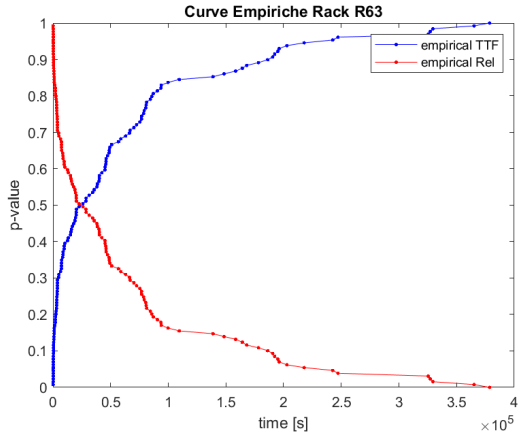
Figura 5.15: Sensitivity analysis per i rack critici

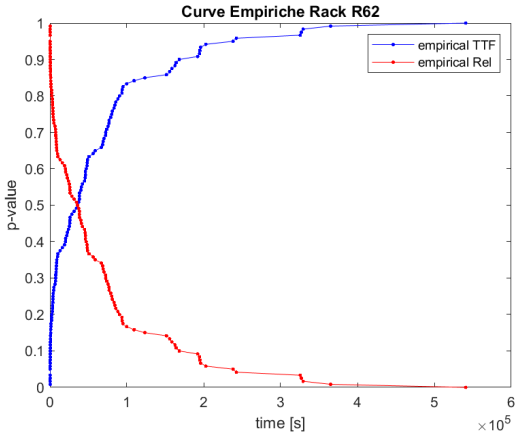
Come è possibile vedere dal grafico, è possibile definire una finestra univoca per tutti i Rack. La finestra ideale sembra essere un valore tra 100 e 150, e quindi si è deciso di scegliere un valore di finestra di coalescenza pari a 150. Il numero di tuple ottenute è riportato in tabella.

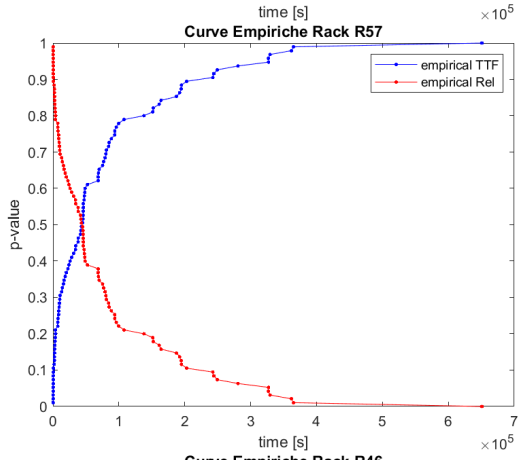
Rack	Coalescence Window	Numero Tuple
R63	150	130
R62	150	121
R57	150	96
R56	150	99
R46	150	103
R71	150	81
R03	150	87
R12	150	89

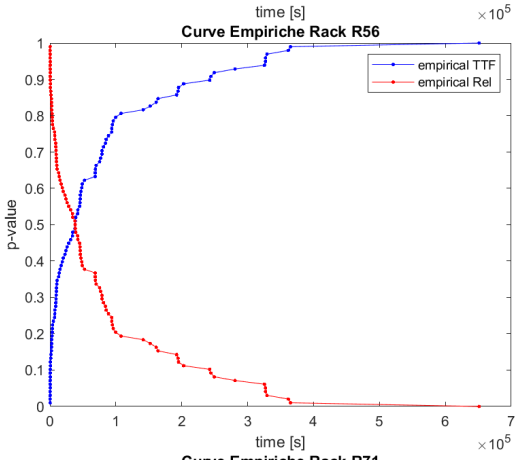
5.4.6 Fase di Data Analysis dei rack

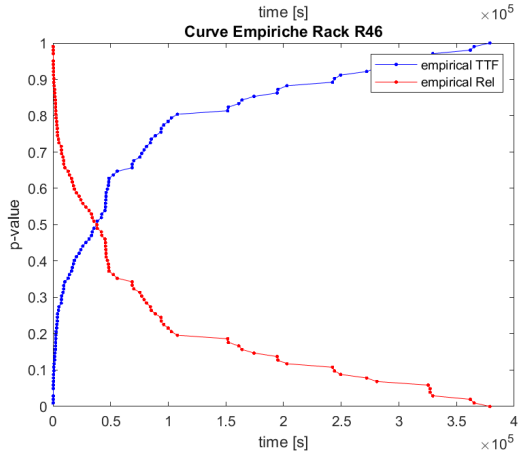
A valle dell'analisi di sensitività, si passa allo studio della reliability empirica, adattando lo script alla trattazione di reliability multiple. Dall'esecuzione dello script si ottengono le seguenti distribuzioni empiriche.

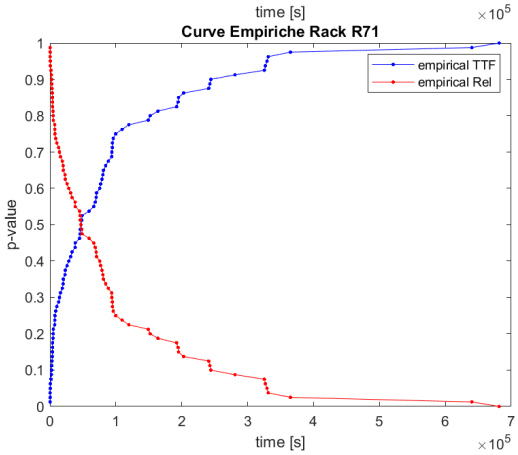

Curve Empiriche Rack R63
empirical TTF
empirical Rel
p-value
time [s]
×10^5

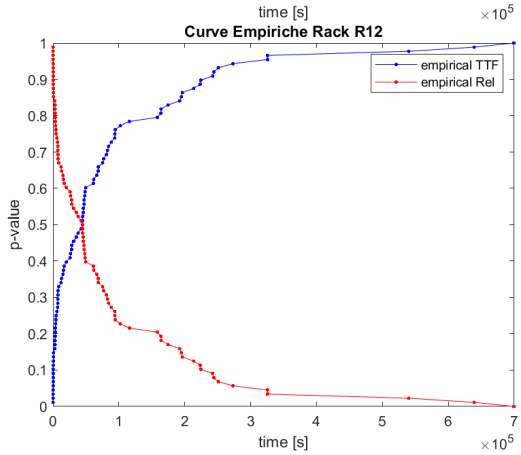

Curve Empiriche Rack R62
empirical TTF
empirical Rel
p-value
time [s]
×10^5

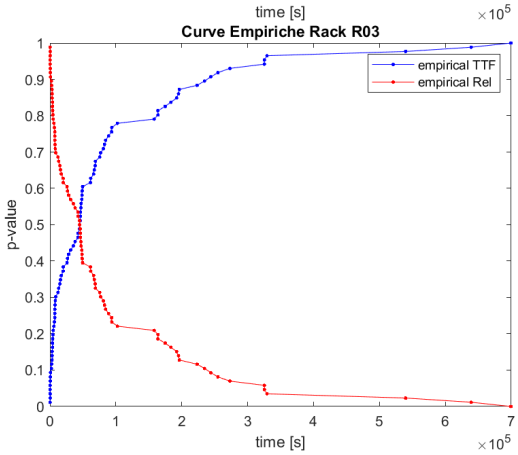

Curve Empiriche Rack R57
empirical TTF
empirical Rel
p-value
time [s]
×10^5


Curve Empiriche Rack R56
empirical TTF
empirical Rel
p-value
time [s]
×10^5

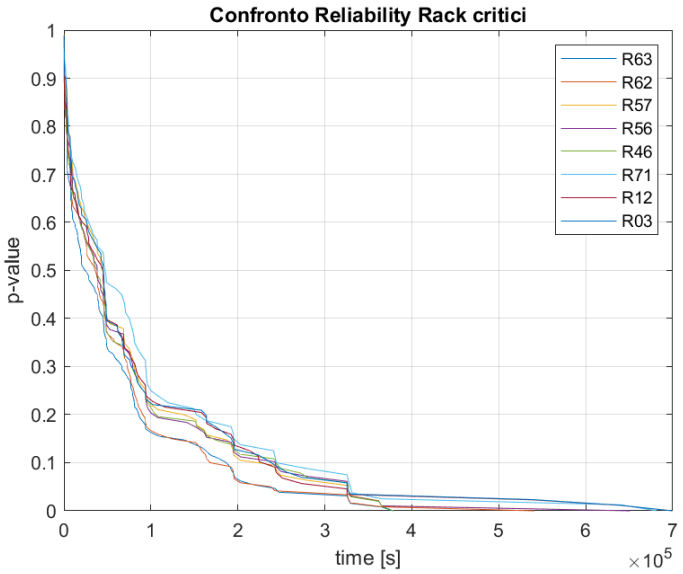

Curve Empiriche Rack R46
empirical TTF
empirical Rel
p-value
time [s]
×10^5


Curve Empiriche Rack R71
empirical TTF
empirical Rel
p-value
time [s]
×10^5


Curve Empiriche Rack R12
empirical TTF
empirical Rel
p-value
time [s]
×10^5


Curve Empiriche Rack R03
empirical TTF
empirical Rel
p-value
time [s]
×10^5

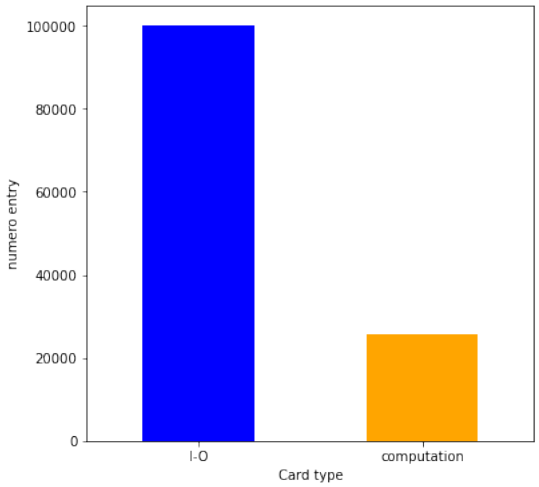
Per confrontare tali reliability è utile riportare gli andamenti all'interno di un unico grafico; da tale grafico è possibile verificare come il rack R63 risulti quello a reliability mediamente peggiore, anche se per valori di tempo alti sembrerebbe essere migliore degli altri rack considerati.



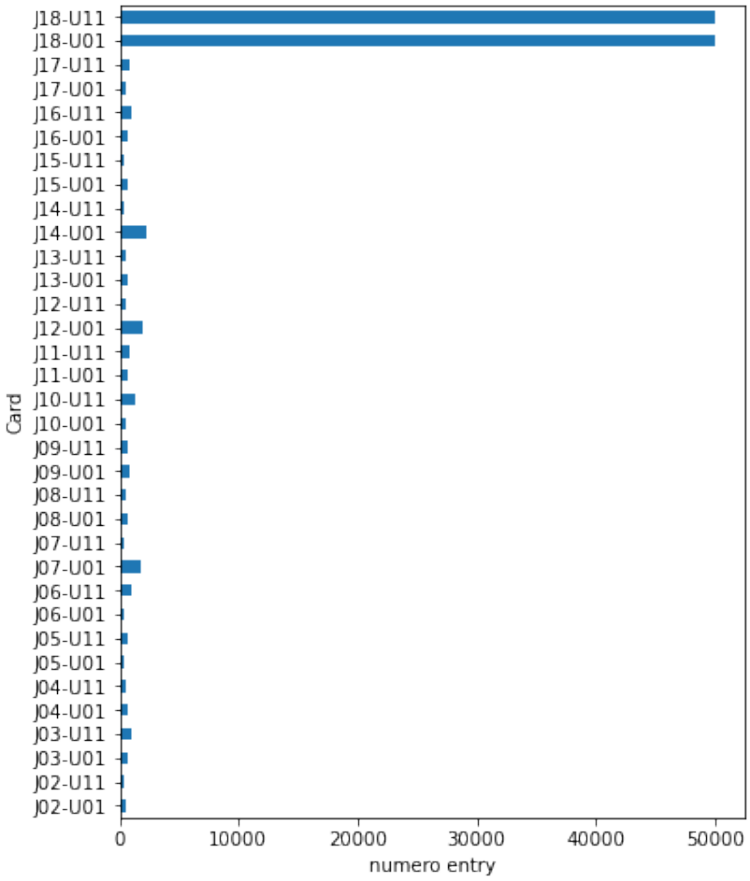
Dal confronto emerge anche che le reliability non sono poi così differenti tra i rack, tanto da risultare a volte anche indistinguibili.

5.4.7 Fase di Data Manipulation delle card

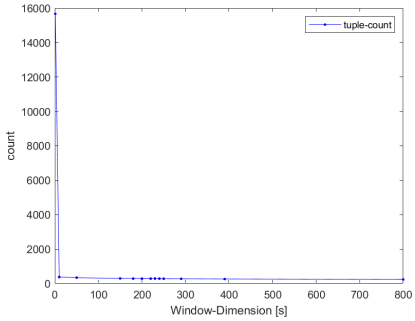
A valle del confronto di reliability sui rack, e vista l'ipotesi di un ipotetico bottleneck nel sistema di I-O del sistema BGL, si è scelto di analizzare la reliability andando a categorizzare gli eventi del log come appartenenti alla categoria I-O o alla categoria computazionale. Questo è possibile grazie al fatto che si conosce l'effettivo componente che è responsabile delle sole operazioni di I-O, ed è quindi possibile isolare tale componente e considerare le entry ad esso afferenti come entry di tipo I-O. Graficando la distribuzione delle entry così categorizzate si ottiene il seguente istogramma.



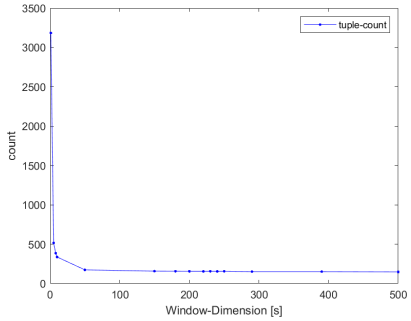
Da tale distribuzione appare chiaro che c'è un problema che riguarda il sistema di I-O del supercalcolatore. Analizzando infatti le entry raggruppate per Card, si ottiene che le card di I-O sono responsabili di un numero eccessivo di errori, che rappresentano la quasi totalità del file log, come si nota dal grafico in figura.



Sembra quindi chiaro che il bottleneck del sistema è rappresentato dall'intero sistema di I-O. Si va quindi a graficare l'andamento del numero di tuple in funzione della finestra di coalescenza, ottenendo i seguenti andamenti.



(a) Finestre di coalescenza card di I-O



(b) Finestre di coalescenza card computazionali

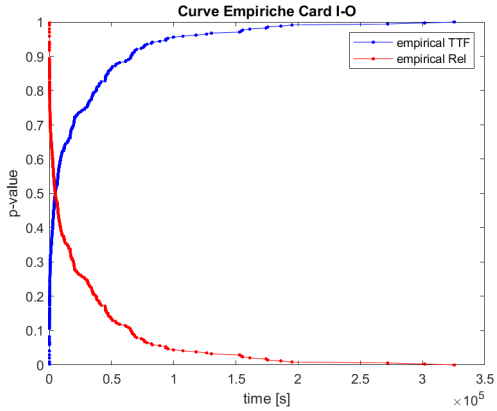
Come si intuisce dai grafici, non è possibile definire un valore di finestra di coalescenza univoca per

entrambe le tipologie di card, ed è necessario quindi differenziare tale valore. I valori di finestra scelti sono riportati in tabella.

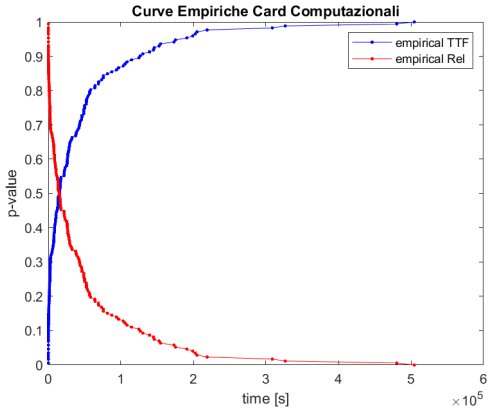
Nodi	Coalescence Window
I-O	50
Computazionali	75

5.4.8 Fase di Data Analysis delle card

A valle della decisione di una finestra di coalescenza, si è eseguito l'algoritmo di tupling per le due partizioni di log, e sono stati graficati gli andamenti delle reliability empiriche. Tali grafici sono riportati in figura.

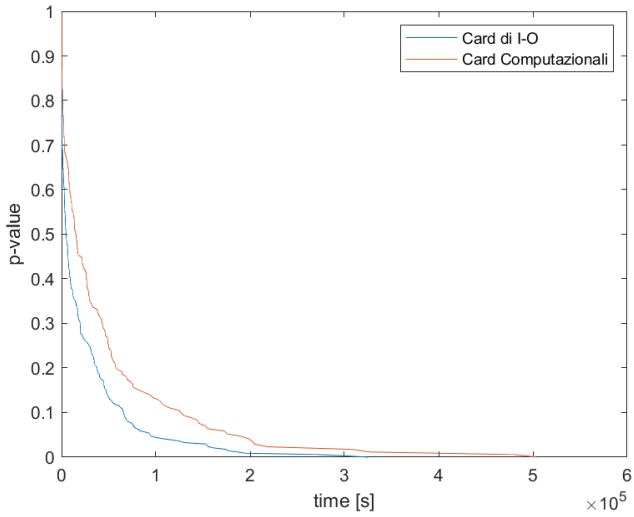


(a) Curve empiriche card di I-O

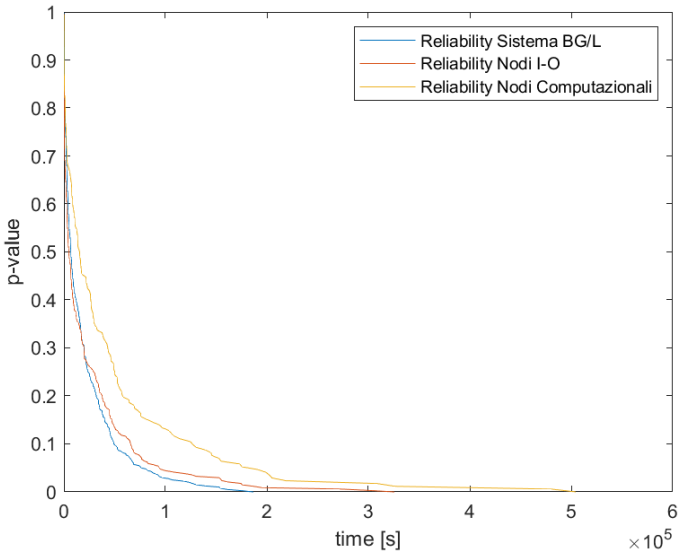


(b) Curve empiriche card computazionali

Confrontando le reliability empiriche si ottiene che il sistema di I-O del supercalcolatore BG/L ha una reliability peggiore rispetto a quella del sistema computazionale. Tale risultato da una ulteriore conferma all'ipotesi di bottleneck fatta per il sistema di I-O. Il confronto delle reliability è riportato in figura.



La conferma definitiva all'ipotesi di bottleneck è data dal confronto tra la reliability del sistema BGL e la reliability del sistema di I-O. Avendo le due curve andamento simile è deducibile che la reliability del sistema di I-O sia quella che influenza maggiormente quella dell'intero sistema. Tali grafici sono riportati in figura.



5.4.9 Domande

Si possono usare le stesse finestre di coalescenza in diversi nodi?

Per quanto riguarda l'analisi effettuata sui Rack, si è visto come è stato possibile definire una finestra di coalescenza univoca per tutti i rack critici. Per quanto riguarda le card invece, ciò non avviene in quanto le card dedicate all'I-O presentano una finestra di coalescenza minore.

Cosa si può dire della relazione tra l'affidabilità del sistema e l'affidabilità dei nodi?

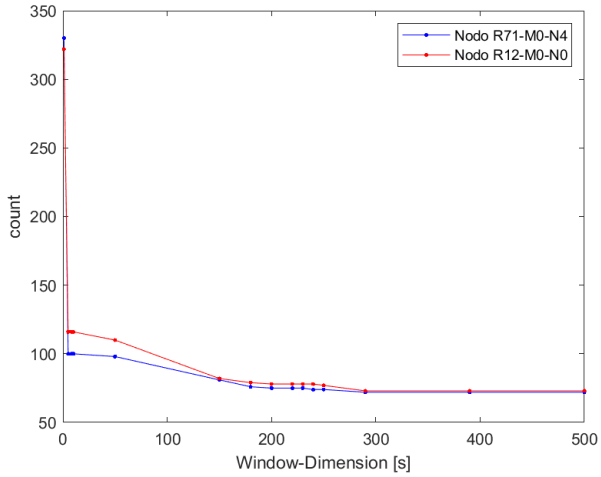
I nodi dedicati all'I-O sembrano esporre una reliability che è molto simile a quella del sistema stesso, cosa che invece non si verifica per i nodi di tipo computazionale, che risultano più affidabili.

Esistono colli di bottiglia dell'affidabilità (cioè, i principali contributori al numero totale di fallimenti)?

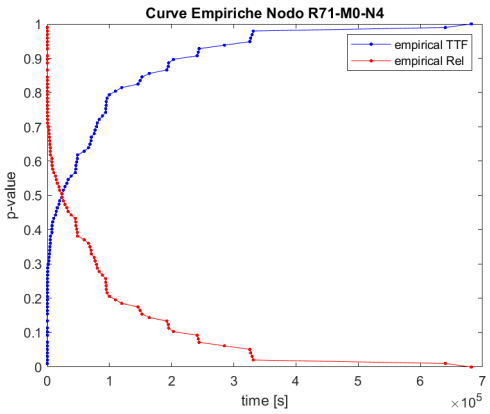
Un ipotetico collo di bottiglia, per le considerazioni fatte precedentemente, è da attribuire all'intero sistema di I-O, visto che la reliability del sistema sembra dipendere fortemente da quella di tali nodi. Non si verifica però il fenomeno che è stato analizzato nel sistema Mercury, dove un nodo esprime una reliability peggiore anche dell'intero sistema.

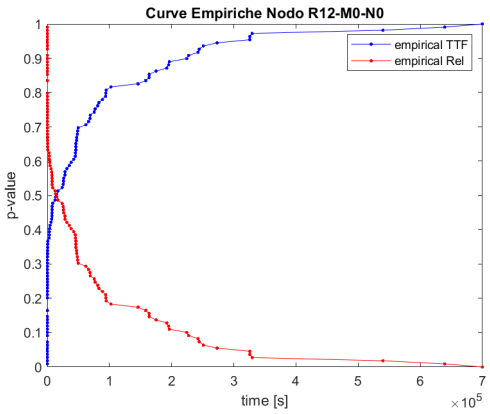
Nodi funzionali simili (per esempio, due nodi BG/L I/O Ri:Mx:Nz) presentano parametri di affidabilità simili?

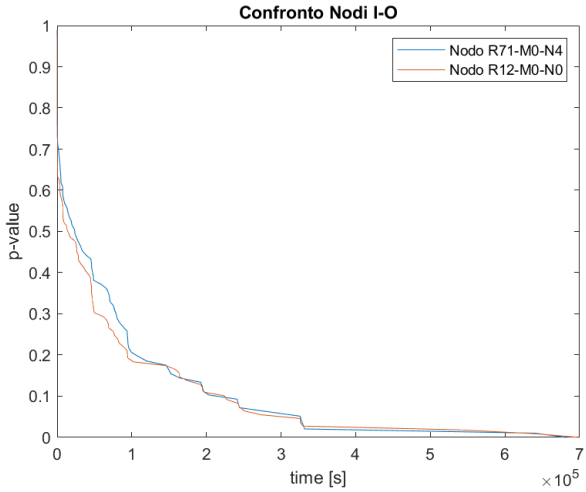
Per il sistema BGL si è deciso di analizzare i primi 4 nodi che contengono più entry nel file di log, cioè R71-M0-N4 ed R12-M0-N0 come nodi relativi all'I-O, e R63-M0-N2 e R03-M1-NF come nodi di tipo computazionale. Come è visibile dal grafico in basso, i due nodi considerati di I-O espongono un andamento di tuple al variare della finestra che è pressochè identico, ed è quindi ragionevole considerare la stessa finestra di coalescenza, in tal caso pari ad esempio a 50.



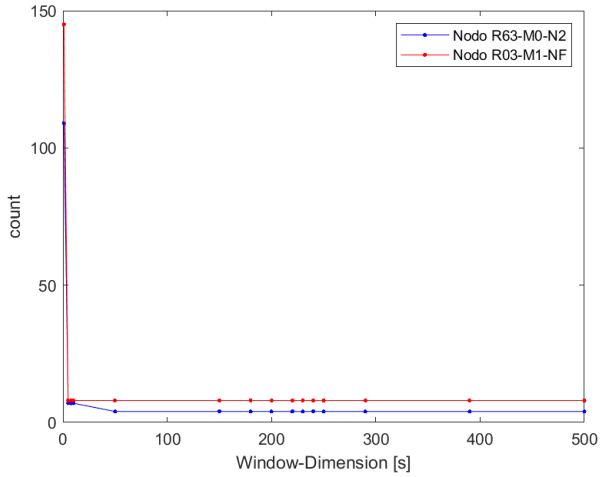
Scelta la finestra si passa a considerare le reliability empiriche: come si vede dal grafico, entrambi i nodi esprimono una reliability comparabile, e a tratti sovrapponibile, ed è quindi ragionevole considerare che i nodi che contengono card di I-O esprimano tutti una reliability simile.



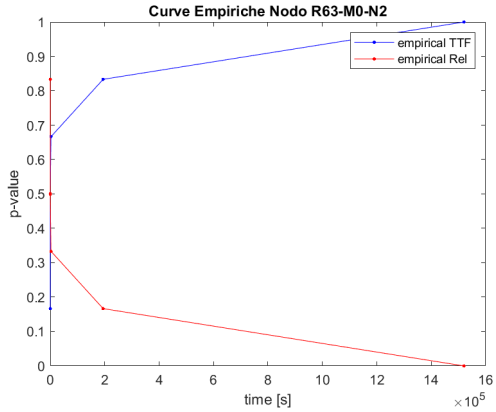


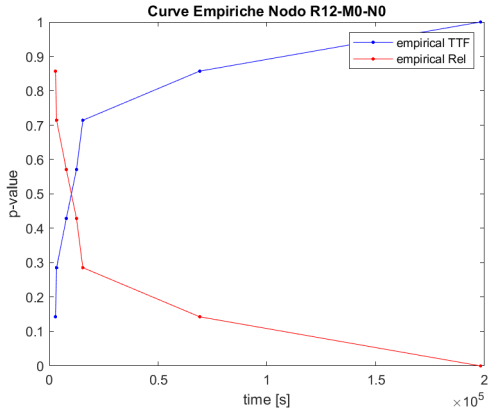


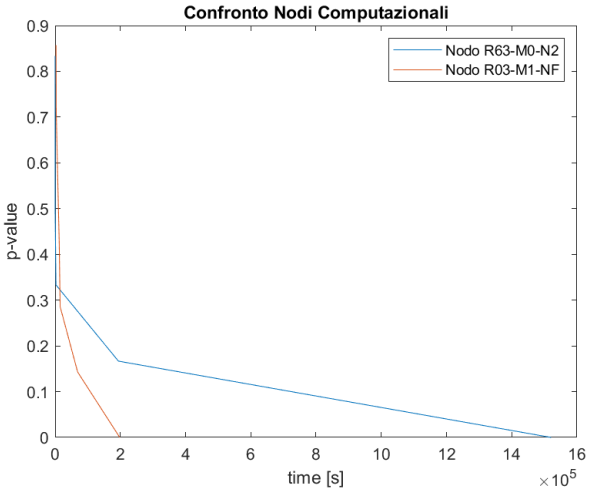
Si eseguono quindi gli stessi confronti tra nodi di tipo computazionale: come è possibile vedere dal grafico, anche per tali nodi è possibile utilizzare lo stesso valore di finestra di coalescenza, e in particolare si è scelto di definirla pari a 8 secondi.



Successivamente si è verificato l'andamento delle reliability empiriche: come si vede dai grafici, le reliability sembrano avere una andamento simile, ma il nodo R12-M0-N0 presenta dei tempi decisamente più lunghi. Ciò che valeva per i nodi di I-O quindi, non è possibile confermarlo anche per tali tipologia di nodi.







In generale quindi non è possibile confermare l'evidenza di similitudini tra le reliability di nodi della stessa tipologia per tutti i tipi di nodi, ma l'ipotesi è confermabile solo per nodi contenenti card di tipo I-O.

5.5 Confronto tra i due sistemi

Per concludere, si evidenziano le differenze riscontrate durante la trattazione precedente tra i sistemi Mercury e Blue Gene/L. In primo luogo è possibile affermare che il bottleneck di Mercury è rappresentano da un singolo nodo, tg-c401. Al contrario, il bottleck di BGL non è rappresentato da un singolo nodo ma dall'intero sottosistema di I/O. Inoltre, mentre gli errori di BGL sono distribuiti in maniere uniforme nello spazio e nel tempo, non è possibile affermare lo stesso nel caso di Mercury. La maggior parte delle entry relative a quest'ultimo sono infatti originate non solo sul singolo nodo tg-401, ma anche in un arco di tempo limitato della durata di due giorni circa. In conclusione, si riportano in figura le reliability empiriche ottenute per i due sistemi. Dal grafico si evince che il sistema Mercury è meno reliable rispetto a Blue Gene/L.

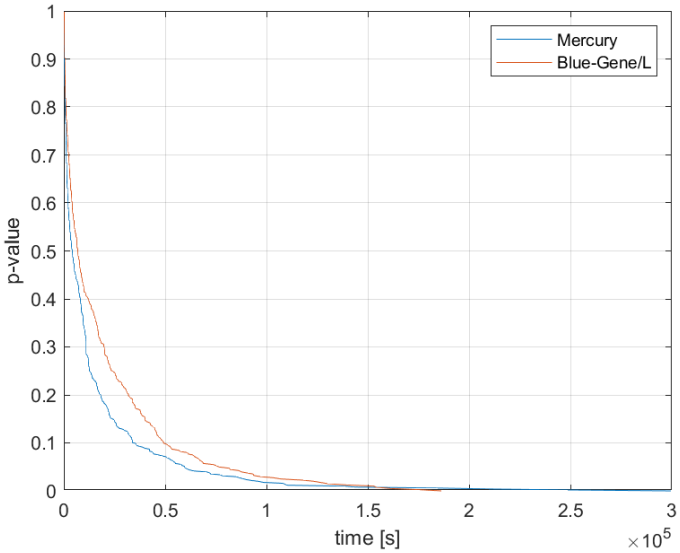


Figura 5.18: Grafico di comparazione delle Reliability di Mercury e di BlueGene/L