Tesina di Impianti di Elaborazione

Pafundi Vincenzo - Mat. XX/XXXX Riccio Marco - Mat. M63/760
Russo Davide - Mat. M63/820

4 febbraio 2019

Indice

Capitolo 1

Benchmark

1.1 Traccia

Si implementi un benchmark relativo ad operazioni I/O su Linux, in particolare per la lettura e scrittura di un fle binario.

- grandezza del file: [5 MB, 20 MB, 100 MB]
- grandezza del blocco: [50 KB, 200 KB, 1 MB]

ps. Si pulisca la cache del processore prima di eseguire il benchmarking.

1.2 Soluzione

Si riportano nella tabella le caratteristiche hardware e software del calcolatore utilizzato per effettuare il benchmarking:

Marca e modello	Lenovo YOGA 55-14IBD
CPU	Intel Intel Core i3-5005U 2 GHz 2 core fisici
RAM	4 GB GDDR3
Memoria secondaria	500 GB HDD
Sistema operativo	Ubuntu 18.04

Il primo passo è stato quello di effettuare un'analisi preliminare, con 7 osservazioni per ogni possibile combinazione dei fattori FileSize e BlockSize, per ottenere una prima stima dei tempi di read e write. Tali tempi sono stati utilizzati per calcolare la dimensione di ogni campione, e quindi il numero minimo di osservazioni necessarie affinchè potesse essere garantito un determinato grado di accuratezza ed una determinata confidenza; la formula per ricavare tale numero minimo è la seguente.

$$n = \left(\frac{100 \cdot z \cdot s}{r \cdot \bar{x}} \right)^2$$

I parametri per ogni campione da considerare sono:

- z, livello di confidenza;
- s, deviazione standard;
- $\bar{x}$, media campionaria;
- r, percentuale di accuratezza.

L'analisi effettuata prevede, dopo aver calcolato per ogni combinazione media e deviazione standard, una scelta approriata di accuratezza e confidenza in maniera tale da trovare un compromesso tra numero di esperimenti e complessità computazionale per condurli. A tal fine, è stata realizzata una funzione MATLAB che implementa la formula mostrata prendendo in ingresso media, deviazione standard e z-quantile. Per quanto riguarda il valore di accuratezza, bisogna fare alcune considerazioni: ogni campione è caratterizzato da una media e una deviazione standard; la scelta di r non può essere vincolata semplicemente alla media, ma deve tenere in considerazione anche come i valori si distribuiscono attorno ad essa. Se così non fosse, il numero di osservazioni per alcuni campioni risulterebbe eccessivo da calcolare, mentre per altri sarebbe molto basso.

FIGURE

Sulla base di tali considerazioni, per il calcolo di r è stato quindi tenuto conto per ogni campione non il valore della media, bensì il coefficiente di variazione (COV), scegliendo il valore di r proprio pari a 1/4 di quest'ultimo. Considerando come confidenza il 99%, si è ottenuto come risultato un numero di esperimenti per ogni campione pari a 49, che si è stimato possibili eseguire in un tempo complessivo di circa 4 ore.

SCRIPT

Com'è possibile notare dai risultati dello script, la peggiore accuracy che si ottiene per le scritture è del 20%, mentre per le letture è di circa il 6%. Tali valori sono isolati a pochi campioni, il che ci permette di affermare che in generale le scelte effettuate siano comunque accurate.

Dopo aver estratto 9 campioni da 49 osservazioni ognuno, sono stati messi a confronto i campioni relativi alle diverse configurazioni di block size per medesimo file size. Sono state applicate in maniera preliminare delle statistiche per verificare che i campioni appartenenti a configurazioni diverse di block size appartengano effettivamente a popolazioni differenti tra loro. Per la verifica è stato utilizzato il t-test

IMMAGINE

Verificato ciò sono state quindi calcolate le medie campionarie e gli intervalli di confidenza per ogni campione.

TABELLA

Per il Teorema del Limite Centrale, essendo la numerosità campionaria maggiore di 30 (49), è possibile dire che le medie campionarie stimano con una confidenza del 95% la media della popolazione da cui provengono. Essendo quindi significativamente diversi i risultati è possibile infine effettuare una valutazione, fissato ogni file size, riguardo a quale block size garantisca tempi medi di lettura e scrittua migliori.

1.2.1 Codice

```
1 #include "benchmark.hpp"
2
3 long int filesize[N] = {5*1048576, 2*10485760, 104857600};
```

```
4  long int blocksize[N] = {5*10240, 2*102400, 1*1048576};
5
6  int main() {
7    long int numInteraction = 0;
8    unsigned int ex = 0, rep = 0;
9
10   for(int i=0;i<N;i++) {
11     for(int j=N-1;j>=0;j--) {
12       numInteraction = filesize[i]/blocksize[j];
13       cout << "RESULTS OF EXPERIMENT NUMBER: " << ++ex << endl;
14       cout << "FILESIZE: " << filesize[i] << " BLOCKSIZE: " << blocksize[j]
             << " NUMINTERACTION: " << numInteraction << endl;
15       for(int k=NUM_ESPERIMENTI;k>0;--k) {
16         cout << rep++%30 << ";";
17         analysisWrite(blocksize[j], numInteraction);
18         //checkFileSize(filesize[i]);
19         analysisRead(blocksize[j], numInteraction);
20       }
21     }
22   }
23   return 0;
24 }
```

Codice Componente 1.1: main.cpp

```
1  #ifndef _BENCHMARK_H_
2  #define _BENCHMARK_H_
3
4  #include <stdlib.h>
5  #include <stdio.h>
6  #include <iostream>
7  #include <fstream>
8  #include <fcntl.h>
9  #include <unistd.h>
10 #include <sys/random.h>
11 #include <chrono>
12 #include <errno.h>
13
14 using namespace std;
15
16 #define NUM_ESPERIMENTI 30
17 #define N 3
18
19 void initBufferWrite(long int);
20 void initBuffe(void);
21 void initBufferRead(void);
22 void termBufferWrite(void);
23 void termBufferRead(void);
```

```
24 void analysisWrite(long int, long int);
25 void analysisRead(long int, long int);
26 long int fileSize();
27 void checkFileSize(long int);
28
29 #endif //_BENCHMARK_H_
```

Codice Componente 1.2: benchmark.hpp

```
1  #include "benchmark.hpp"
2
3  void *bufferRead;
4  void *bufferWrite;
5
6  void termBuffer() {
7    termBufferRead();
8    termBufferWrite();
9    remove("file_prova.bin");
10 }
11
12 void termBufferRead() {
13   free(bufferRead);
14 }
15
16 void termBufferWrite() {
17   free(bufferWrite);
18 }
19
20 void initBufferWrite(long int bs) {
21   int errMemAlign = posix_memalign(&bufferWrite, 4096, bs);
22   system("sync; echo 3 > /proc/sys/vm/drop_caches");
23   //cout << pathconf("file_prova.bin",_PC_REC_XFER_ALIGN) << endl;
24   if (errMemAlign!=0) {
25     cout << errMemAlign << endl;
26     exit(0);
27   }
28   if (getrandom(bufferWrite, bs, GRND_NONBLOCK)==-1) {
29     perror("get random error");
30     termBuffer();
31     exit(0);
32   }
33 }
34
35 void initBufferRead(long int bs) {
36   int errMemAlign = posix_memalign(&bufferRead, 4096, bs);
37   //cout << pathconf("file_prova.bin",_PC_REC_XFER_ALIGN) << endl;
38   if (errMemAlign!=0) {
39     cout << errMemAlign << endl;
```

```
40      exit(0);
41    }
42  }
43
44  void analysisWrite(long int bs, long int num) {
45    std::chrono::duration<double> elapsedAccumulatorWrite;
46    elapsedAccumulatorWrite.zero();
47    int writeSuccessful = 0;
48
49    //open file and control
50    int fd = open("file_prova.bin", O_CREAT|O_TRUNC|O_WRONLY|O_APPEND|O_DIRECT
          |O_SYNC, S_IRWXU);
51    if(fd==-1) {
52      perror("open file error");
53      termBuffer();
54      exit(0);
55    }
56
57    for(int i=num-1;i>=0;--i) {
58
59      //initialize buffer
60      initBufferWrite(bs);
61
62      //start timer
63      auto start_time = std::chrono::system_clock::now();
64
65      //write buffer
66      writeSuccessful = write(fd, bufferWrite, bs);
67
68      //end timer
69      auto end_time = std::chrono::system_clock::now();
70
71      //control write
72      if (writeSuccessful != bs) {
73        perror("write error");
74        termBuffer();
75        exit(0);
76      }
77
78      //free memory buffer
79      termBufferWrite();
80
81      //update accumulator
82      elapsedAccumulatorWrite += end_time-start_time;
83    }
84    //close file
85    close(fd);
86
87    //print write time
```

```
88    cout << elapsedAccumulatorWrite.count() << ";";
89  }
90
91  void analysisRead(long int bs, long int num) {
92    std::chrono::duration<double> elapsedAccumulatorRead;
93    elapsedAccumulatorRead.zero();
94    int readSuccessful = 0;
95
96    int fd = open("file_prova.bin", O_RDONLY|O_DIRECT|O_SYNC, S_IRWXU);
97    if(fd==-1) {
98      perror("open file error");
99      termBuffer();
100     exit(0);
101   }
102
103   for(int i=num-1;i>=0;--i) {
104     //initialize buffer
105     initBufferRead(bs);
106
107     auto start_time = std::chrono::system_clock::now();
108
109     readSuccessful = read(fd, bufferRead, bs);
110
111     auto end_time = std::chrono::system_clock::now();
112
113     if (readSuccessful != bs) {
114       perror("read error");
115       termBuffer();
116       exit(0);
117     }
118
119     termBufferRead();
120
121     elapsedAccumulatorRead += end_time-start_time;
122   }
123
124   close(fd);
125   cout << elapsedAccumulatorRead.count() << ";" << endl;
126 }
127
128 long int fileSize() {
129   streampos begin, end;
130   ifstream fd ("file_prova.bin", ios::binary);
131   begin = fd.tellg();
132   fd.seekg (0, ios::end);
133   end = fd.tellg();
134   fd.close();
135   return end-begin;
136 }
```

```
137
138 void checkFileSize(long int fs) {
139   if(fileSize()<fs) {
140     cout << endl <<"File size mismatch" << endl;
141     cout << fileSize() << " " << fs << endl;
142     exit(0);
143   }
144 }
```

Codice Componente 1.3: function.cpp

Capitolo 2

PCA & Clustering

2.1 Traccia

Analizzare i dati contenuti all'interno del file fornito, ottenuti dal monitoraggio di un file system di Unix. Si vuole ridurre il dataset ed ottenere un workload sintetico che sia rappresentativo di quello reale, preservando gran parte della varianza.

2.2 Soluzione

Per rispondere al problema si è scelto di utilizzare il tool JMP per l'analisi del dataset al fine di ridurne il numero di dati. Dopo un'analisi preliminare sui dati, la riduzione effettiva del dataset avviene utilizzando due tecniche: Principal Component Analysis (PCA) e clustering.

2.2.1 Analisi preliminare

L'analisi preliminare prevede, dopo aver importato il file nel tool, di analizzare le feature presenti, al fine di eliminare quelle prive di contenuto informativo. A tale scopo si è proceduto all'analisi delle distribuzioni di tali attributi osservando, in particolare, il coefficiente di variazione (CV) di ognuna. Il coefficiente di variazione è la normalizzazione della varianza con la media. Se il coefficiente di variazione è nullo, il parametro misurato è costante, e quindi si sceglie di non includere quella feature nelle successive analisi.

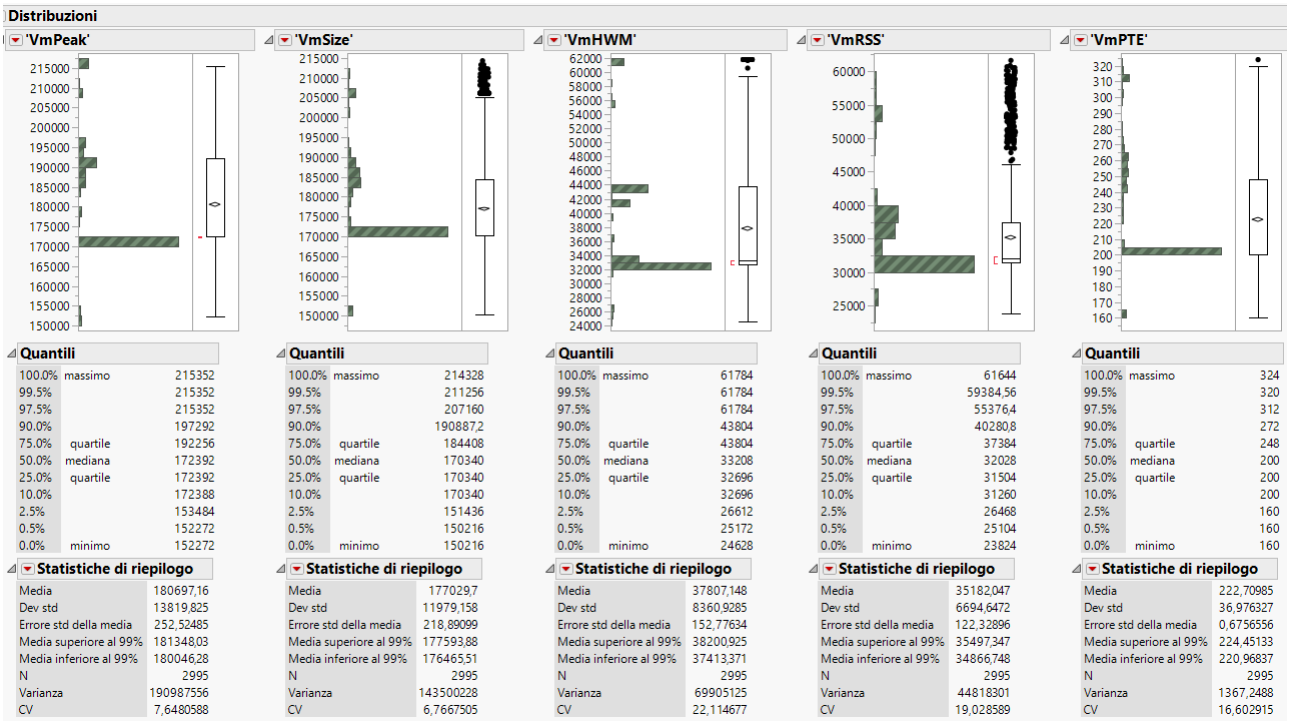


Figura 2.1: Distribuzioni delle feature

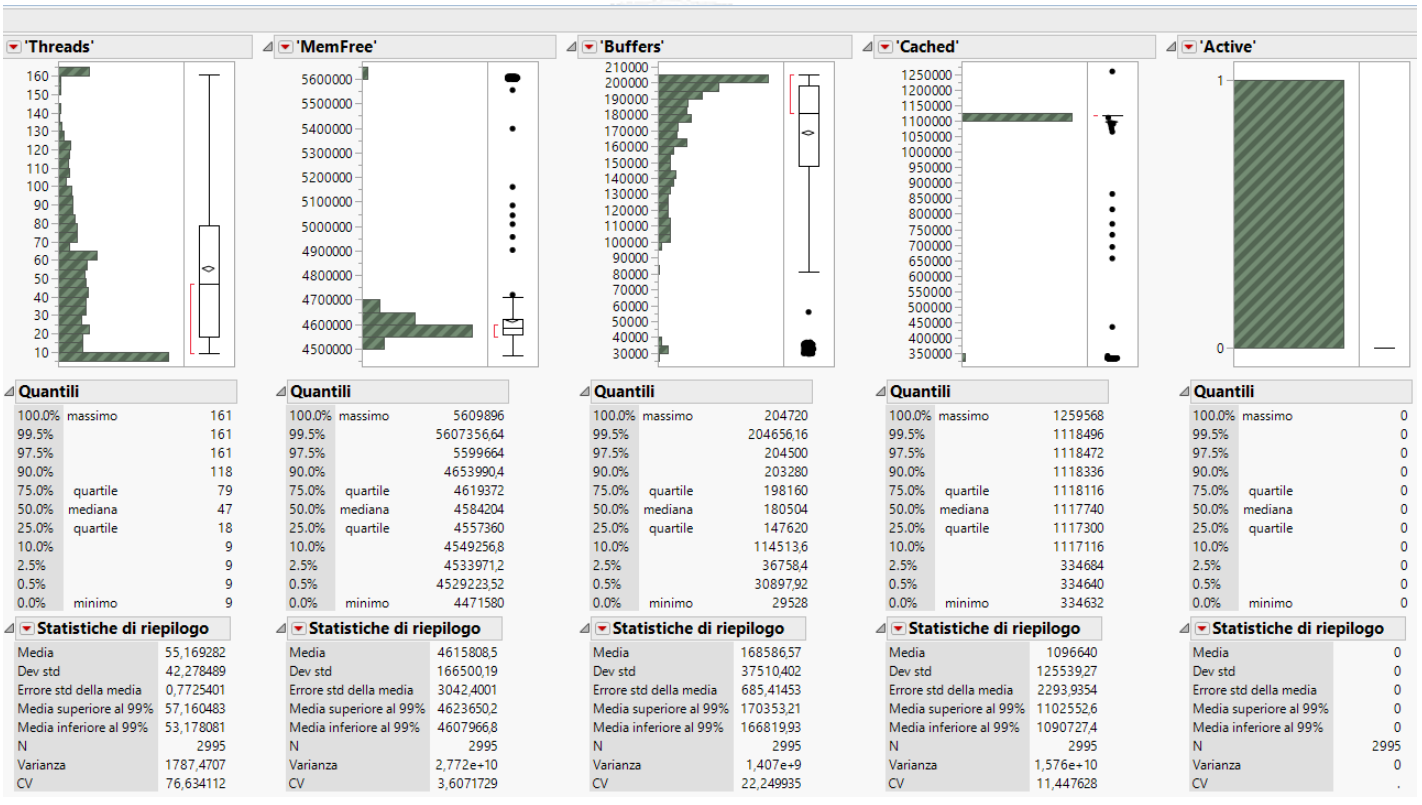


Figura 2.2: Distribuzioni delle feature

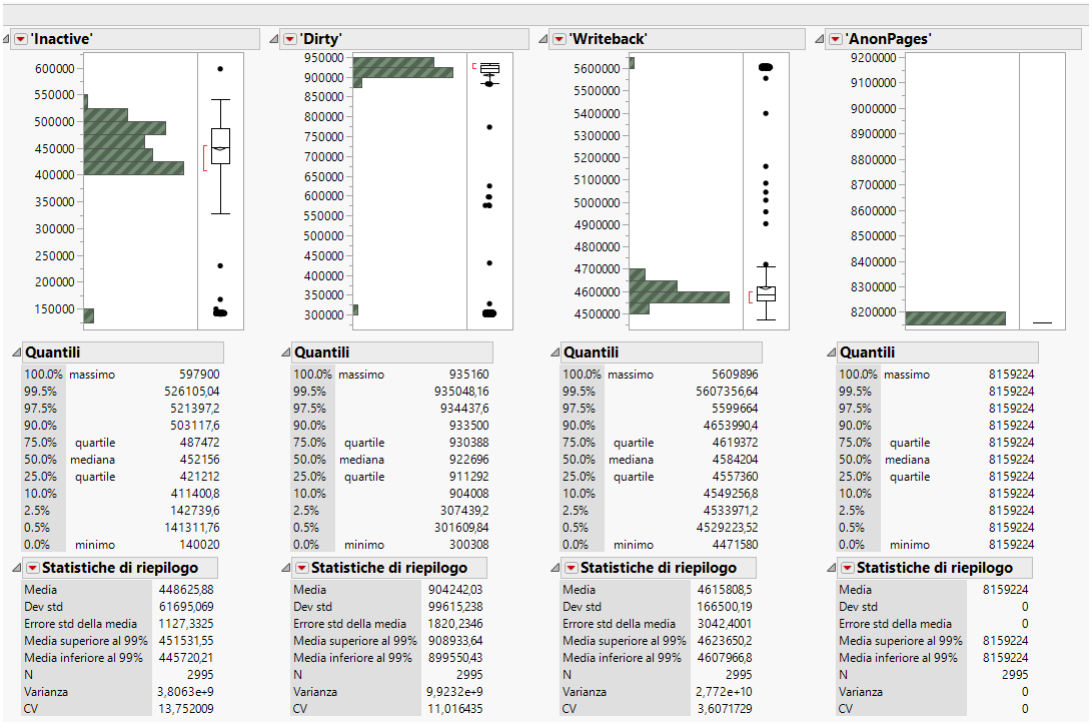
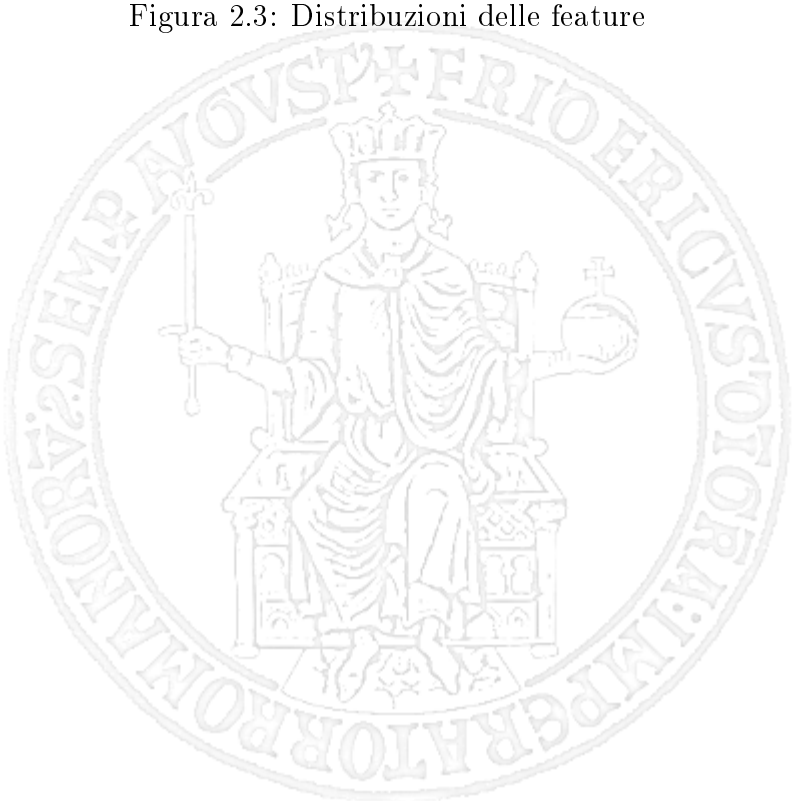


Figura 2.3: Distribuzioni delle feature



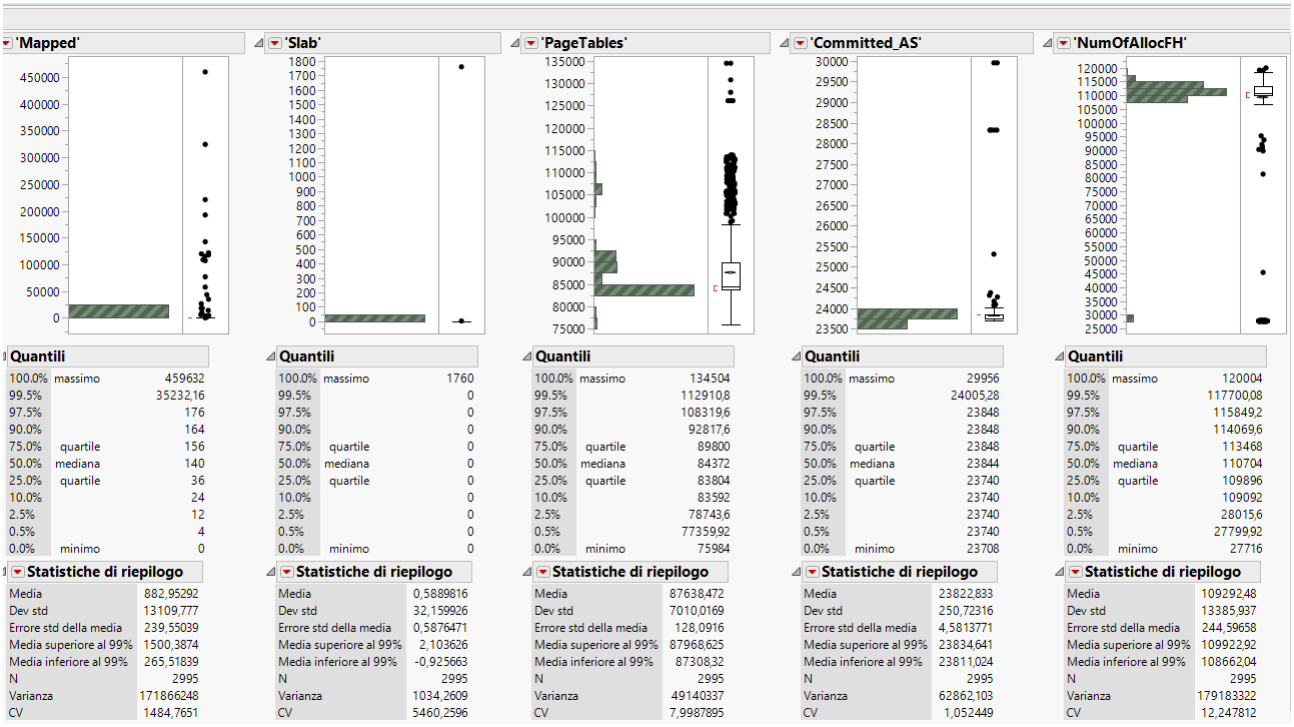


Figura 2.4: Distribuzioni delle feature

Figura 2.5: Distribuzioni delle feature

Come si può osservare dalle distribuzioni mostrate, gli attributi che presentano CV nullo sono: *active*, *anonpages*, *avglatency* ed *errors*.

Alle colonne già eliminate, si aggiunge la colonna *slab* che risulta essere non significativa ai fini dell'analisi in quanto, nonostante la varianza diversa da zero, contribuisce poco alla discriminazione dei punti rispetto alle altre feature.

Un'ulteriore scrematura delle feature si può effettuare osservando colonne che presentano stessi valori. Ad occhio, le colonne *memfree* e *writeback* risultano essere uguali. Per esserne certi, si è utilizzata la funzionalità del tool che permette di applicare delle formule sui dati delle colonne.

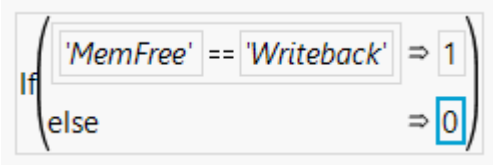


Figura 2.6: Formula confronto colonne

Del risultato prodotto, si è analizzata la distribuzione verificando che questa abbia coefficiente di variazione nullo.

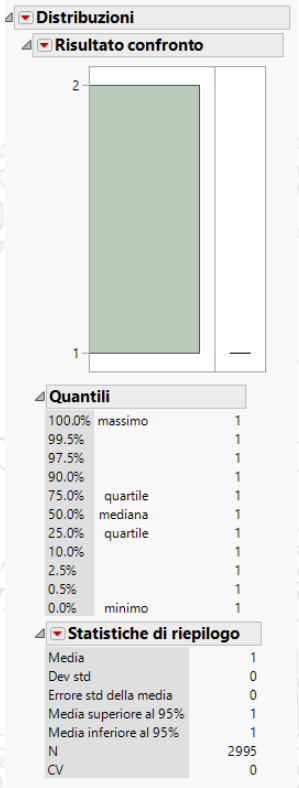


Figura 2.7: Distribuzioni del risultato del confronto

Il risultato di questa fase è una selezione di 18 feature a partire dalle 24 iniziali.

2.2.2 Principal Component Analysis

A questo punto si è proceduto ad effettuare un'analisi delle componenti principali, allo scopo di ridurre ulteriormente il numero di feature conservando la maggior parte della varianza. Ci si pone come obiettivo di conservare almeno il 95% della varianza totale.

Il risultato può essere osservato in forma grafica tramite uno score plot e loading plot.

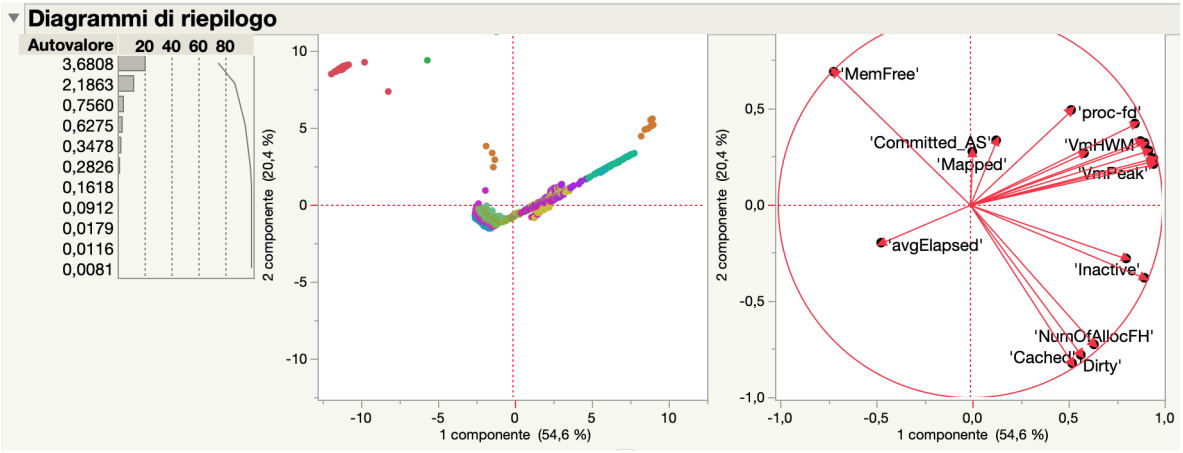


Figura 2.8: Score plot e loading plot generate dal tool

Il tool genera una vista degli autovalori della matrice di correlazione con la relativa percentuale di varianza per ogni componente ottenuta.

Autovalori

Numero	Autovalore	Percentuale	20 40 60 80	Percentuale cumulativa
1	9,8192	54,551		54,551
2	3,6808	20,449		75,000
3	2,1863	12,146		87,146
4	0,7560	4,200		91,346
5	0,6275	3,486		94,832
6	0,3478	1,932		96,764
7	0,2826	1,570		98,334
8	0,1618	0,899		99,233
9	0,0912	0,507		99,740
10	0,0179	0,100		99,839
11	0,0116	0,065		99,904
12	0,0081	0,045		99,949
13	0,0062	0,035		99,983
14	0,0014	0,008		99,991
15	0,0009	0,005		99,996
16	0,0007	0,004		100,000
17	0,0000	0,000		100,000
18	0,0000	0,000		100,000

Figura 2.9: Autovalori della matrice di correlazione

Per rispondere all'obiettivo, si sceglie un numero di componenti principali pari a 6, rappresentativi del **96.764%** della varianza totale.

2.2.3 Clustering

A valle della PCA effettuata, si vuole ridurre ulteriormente il dataset, con la differenza di voler diminuire il numero di istanze. A tale scopo si utilizza la tecnica del clustering di tipo gerarchico sulle componenti principali individuate, tramite la quale, scegliendo come metrica la distanza di Ward, si vuole individuare un trade-off tra la necessità di conservare una buona percentuale di varianza e quella di avere un numero accettabile di cluster.

Il risultato della fase di clustering è apprezzabile tramite il dendrogramma, prodotto dal tool.

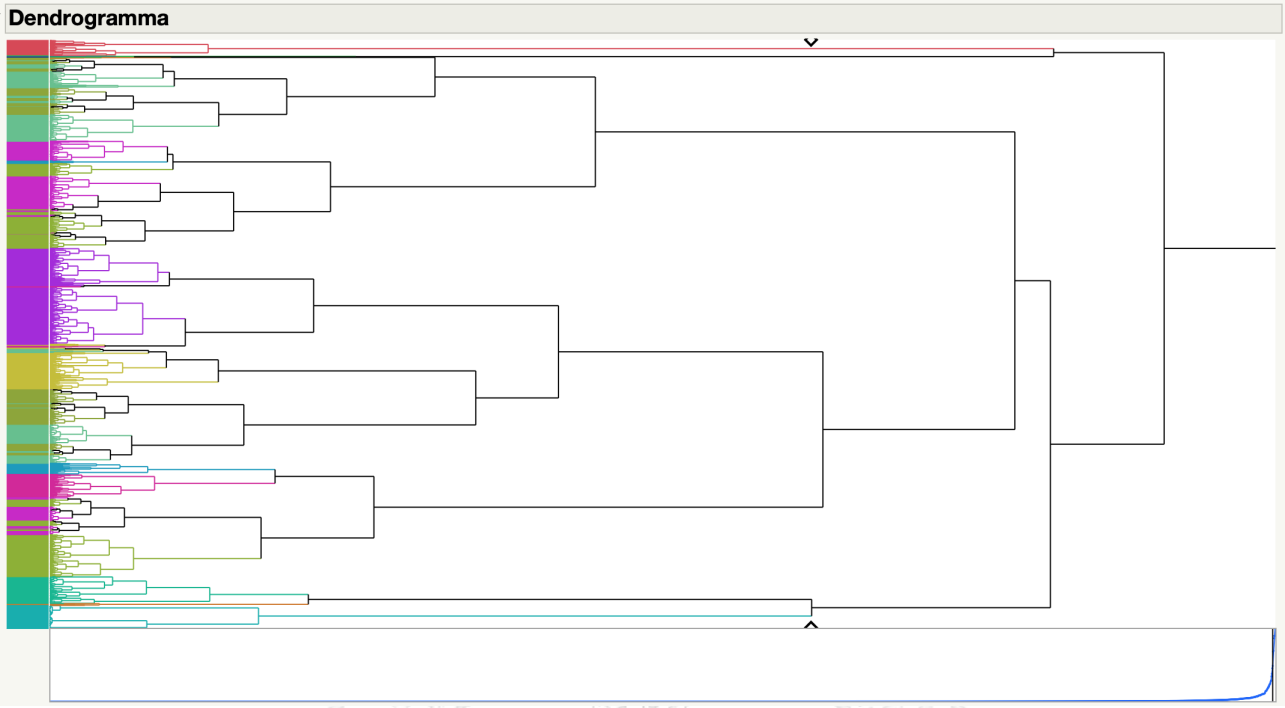


Figura 2.10: Dendrogramma

Inoltre il tool produce anche una tabella contenente, per ogni partizione, il relativo valore della distanza di Ward.

Numero di cluster	Distanza
42	4,53727383
41	4,64824107
40	4,71544410
39	4,73740514
38	4,78214101
37	4,82168691
36	5,11585070
35	5,27519441
34	5,39485020
33	5,53006180
32	5,69106202
31	5,74292374
30	5,83968358
29	5,93161593
28	6,01314292
27	6,07815214
26	6,61466470
25	7,73131312
24	7,81110869
23	8,22772672
22	8,25055421
21	8,97756907
20	9,47509536
19	10,19087495
18	10,31583926
17	10,99907676
16	11,57187362
15	12,62391905
14	12,88568385
13	13,70796307
12	14,94532692
11	15,82795378
10	18,80875084
9	20,80268710
8	24,84198796
7	26,64132735
6	37,20575450
5	37,75534548
4	47,12952556
3	48,86654204
2	49,02494378
1	54,42090668

Figura 2.11: Cronologia di clusterizzazione

Al fine di scegliere il numero di cluster, non avendo informazioni sulla realtà sperimentale, né relative al budget disponibile per condurre gli esperimenti, si è optato per una consistente riduzione del dataset ad un numero di cluster pari a 7; tale valore è stato ottenuto andando ad analizzare i valori delle distanze fornite da JMP (mostrate nella figura 11) e notando che la differenza maggiore tra tali valori si riscontrava nel passaggio da 6 a 7 cluster.

2.3 Calcolo analitico della varianza persa

A questo punto si è proceduto al calcolo preciso della varianza persa a seguito prima della fase di PCA e poi di clusterizzazione. A tal fine, è stato utilizzato il tool di data analysis Knime.

2.3.1 Devianza post-PCA

Per calcolare il valore di tale devianza sono stati dapprima ottenuti i valori di varianza per ogni componente principale, e successivamente sommati tra di loro e moltiplicati per il numero di gradi di libertà. Il valore risultante, che sappiamo spiegare il 96.764% della varianza totale, è di **52148.055.**

2.3.2 Devianza post-clustering

La devianza post-clustering può essere espressa tramite la seguente somma:

Devianza Totale = Devianza Intra-Cluster + Devianza Inter-Cluster

Per calcolare la devianza intra-cluster, sono state isolate le istanze in base al cluster di appartenenza. Per ogni cluster si è proceduto nel seguente modo:

- è stata calcolata la varianza per ogni componente principale (ogni colonna);
- tali valori di varianza sono stati moltiplicati poi per N-1 (con N cardinalità del cluster) per ottenere la devianza;
- i valori di devianza ottenuti sono stati poi sommati per ottenere la devianza complessiva *intra-cluster*.

Per calcolare la devianza inter-cluster, si è dapprima ottenuta da JMP la tabella contenente i centroidi dei cluster individuati; poi:

- di ogni colonna, relativa alle componenti principali di tali punti, rappresentativi dei diversi cluster, è stata ottenuta la varianza;
- tali valori di varianza sono stati poi sommati e moltiplicati per N-1 (con N numero di cluster) per ottenere la devianza *inter-cluster*.

Una volta ottenuti tali valori è stata ottenuta la devianza totale:

$$Dev_{TOT} = Dev_{intra} + Dev_{inter} = 8715.527 + 968.247 \backsimeq 9683.78$$

2.3.3 Conclusione

Con i valori di devianza ottenuti è possibile a questo punto calcolare la percentuale di varianza persa a valle della PCA e del clustering:

Varianza Persa = Devianza Intra-Cluster / Devianza Post PCA

$$Var_{persa}(\%) = \frac{8715.527}{52148.055} \cdot 100 = 16.71\%$$

Quindi, del 96.764% di varianza spiegata dalla PCA, il clustering ne fa perdere il 16.71% (ne spiega quindi l'80.59%).

2.4 Costruzione del workload sintetico

Un possibile workload a questo punto può essere ottenuto con un random sampling dai cluster individuati.

'VmPeak'	'VmSize'	'VmHWM'	'VmRSS'	'VmPTE'	'Threads'	'MemFree'	'Buffers'	'Cached'	'Active'	'Inactive'	'Dirty'	'Writeback'	'AnonPages'	'Mapped'	'Slab'
152272	150216	25172	25108	160	55	5607348	30900	334636	0	141312	301612	5607348	8159224	52	0
170344	170340	31144	31140	200	43	4471580	94884	1259568	0	597900	891332	4471580	8159224	459632	0
170344	170340	31920	31912	200	25	4668808	98624	1117028	0	407620	892240	4668808	8159224	26764	0
178608	174512	36808	36160	208	65	4566668	192424	1117940	0	473252	925612	4566668	8159224	144	0
172392	170340	32696	31340	200	9	4602368	164352	1117452	0	432948	932540	4602368	8159224	28	0
215352	211256	61784	58896	312	161	4529564	204232	1118440	0	525496	908416	4529564	8159224	24	0
215352	206136	61784	53204	312	9	4539096	204664	1118496	0	521232	907432	4539096	8159224	36	0

Figura 2.12: Esempio di workload sintetico

'PageTables'	'Committed_AS'	'NumOfAllocFH'	'proc-fd'	'avgThroughput'	'avgElapsed'	'avgLatency'	'Errors'	Cluster
77364	23772	27764	7196	294240	1530	0	2	1
134504	29956	116216	8024	365692	1530	0	2	2
84208	23740	109296	7208	316832	1530	0	2	3
88500	23844	112872	7212	319412	1020	0	2	4
83740	23844	110316	7204	317048	1530	0	2	5
111240	23848	112956	7320	351388	510	0	2	6
105548	23848	110164	7320	345176	1020	0	2	7

Figura 2.13: Esempio di workload sintetico

Capitolo 3

Capacity Test

3.1 Traccia

3.1.1 Workload Characterization

3.1.2 Capacity Test

3.1.3 Design of experiment

3.2 Soluzione

3.2.1 Introduzione

Il contesto in cui si sono condotte le analisi per questo capitolo è il seguente:

- client:

Marca e modello	Lenovo YOGA 55-14IBD
CPU	Intel Intel Core i3-5005U 2 GHz 2 core fisici
RAM	4 GB GDDR3
Memoria secondaria	500 GB HDD
Sistema operativo	Ubuntu 18.04

- server (SUT):

Marca e modello	Raspberry Pi B+
CPU	ARM ARM710 700 MHz 1 core fisico
RAM	512 MB
Memoria secondaria	8 GB SD
Sistema Operativo	Linux Raspbian

- cavo di rete Ethernet per la comunicazione dei due sistemi.

Per la generazione del workload è stato utilizzato il tool JMeter, in versione 5.0, mentre per il web server è stato utilizzato Apache in versione 2.4.25. Il web server ospita una serie di pagine di puro HTML, classificabili per dimensione:

- 2 pagine small, di 25 kB e 100 kB;
- 2 pagine medium, di 500 kB e 1 MB;
- 2 pagine big, di 2 MB e 5 MB.

3.2.2 Workload Characterization

Il tool per la generazione del workload è stato configurato con 45 thread che effettuano, in modo random, 10 richieste al minuto verso le pagine sopracitate per un tempo di 5 minuti. Attraverso un Listener di tipo Simple Data Writer, sono stati raccolti i parametri di alto livello, in un file .csv.

I parametri di basso livello sono stati raccolti attraverso la direttiva *vmstat* lanciata sul server durante l'esecuzione del test, reindirizzando l'output standard su un file e registrando i parametri ogni secondo.

3.2.2.1 Caratterizzazione dei parametri di alto livello

Per poter effettuare la workload characterization, sono stati richiesti i seguenti parametri di alto livello:

- Thread name, per analizzare quale utente ha effettuato una determinata richiesta;
- Page, ad indicare quale pagina è stata richiesta;
- Bytecount, che rappresenta la dimensione della richiesta.

Si riportano le distribuzioni dei parametri d'interesse:

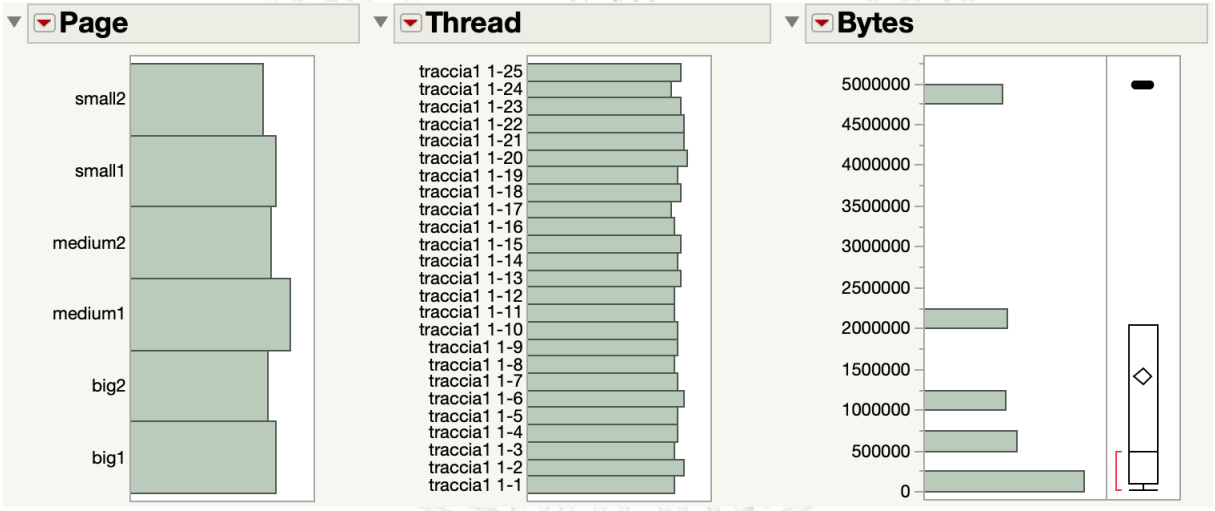


Figura 3.1: Parametri d'interesse

Come si può notare, le richieste sono state eseguite omogeneamente da tutti gli utenti per tutte le pagine. Ciò è conforme con quanto settato in fase di configurazione del tool attraverso il Random Controller.

Risulta interessante notare un altro comportamento del sistema che riguarda la latenza delle richieste. In particolare si è osservato che all'aumentare delle dimensioni delle pagine richieste, la latenza non ha un aumento proporzionale come ci si aspetterebbe:

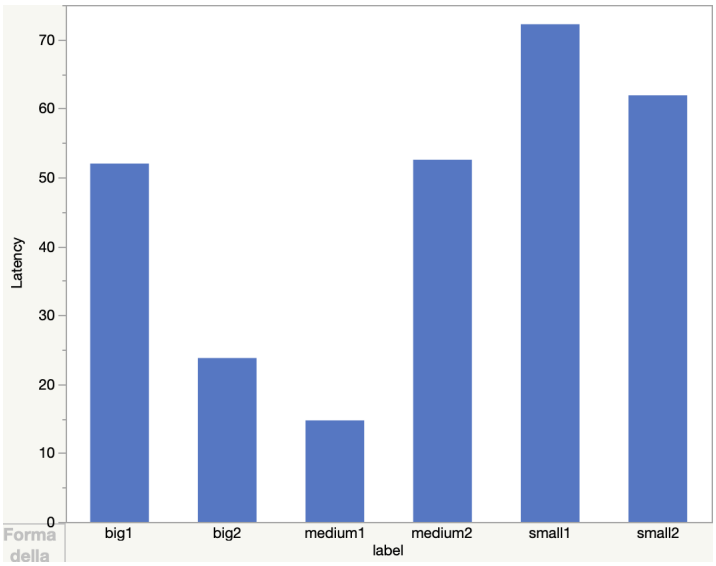


Figura 3.2: Latency media rispetto alla pagina richiesta

Sul grafico è stata plottata la latenza media in relazione alle pagine richieste e, conseguentemente, ai bytes della richiesta. Come si può osservare le pagine small sono quelle caratterizzate da una più alta latenza rispetto alle altre.

Analizzando la matrice di correlazione è possibile notare come ci siano due parametri altamente correlati, ovvero bytes ed elapsed time.

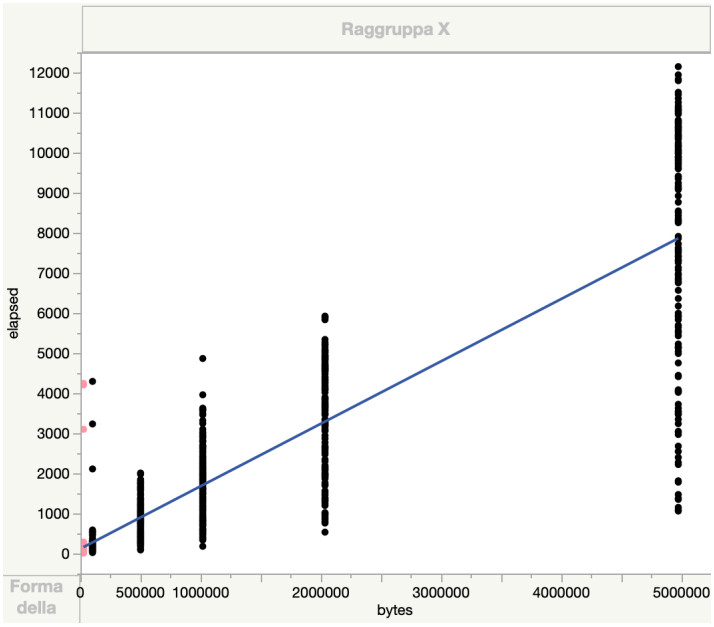


Figura 3.3: Correlazione tra elapsed time e bytes richiesti

In particolare si osserva che, all'aumentare dei bytes della pagina richiesta, l'elapsed time cresce conseguentemente in maniera lineare.

3.2.2.2 Caratterizzazione dei parametri di basso livello

Al fine di caratterizzare i parametri di basso livello, essendo questi valori numerici, si è optato per un'analisi attraverso la PCA e la clusterizzazione. Si è partito dall'analisi delle distribuzioni, non considerando quelle con coefficiente di variazione (COV) nullo.

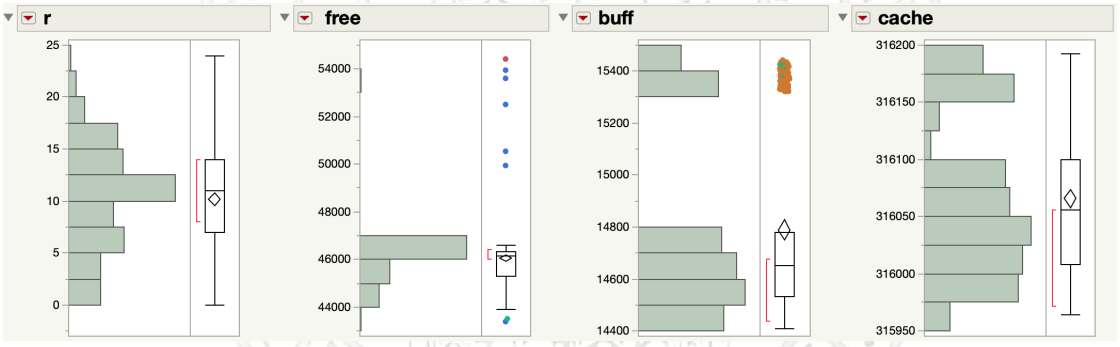


Figura 3.4: Distribuzioni parametri basso livello

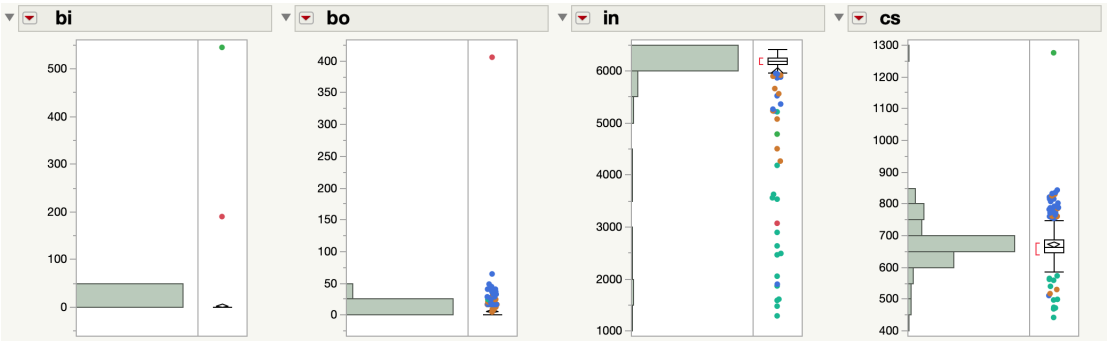


Figura 3.5: Distribuzioni parametri basso livello

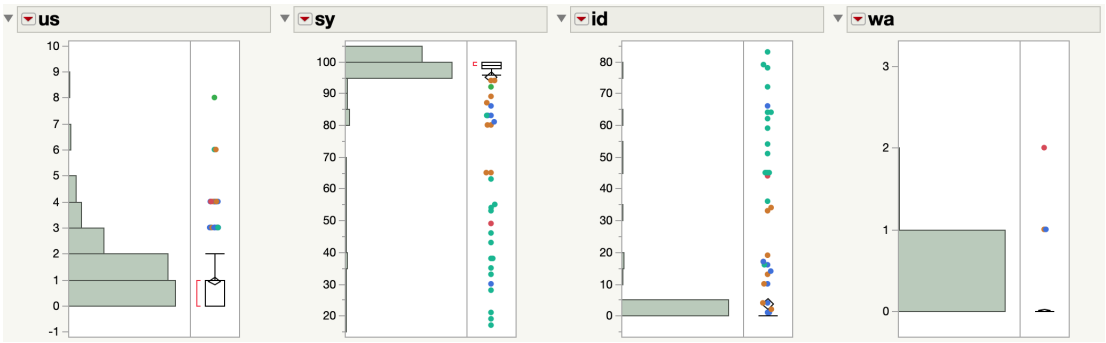


Figura 3.6: Distribuzioni parametri basso livello

Le feature scartate dall'analisi, in quanto tutte nulle, sono le seguenti:

- *b*, numero di processi non interrompibili;
- *swpd*, totale della memoria virtuale utilizzata;
- *si*, totale della memoria swappata dal disco;
- *so*, totale della memoria swappata nel disco;
- *st*, cicli di CPU "rubati" dalla virtual machine, in quanto l'ambiente su cui il web server gira non è virtualizzato ma fisico.

A questo punto si è potuti passare all'analisi delle componenti principali sulle feature selezionate.

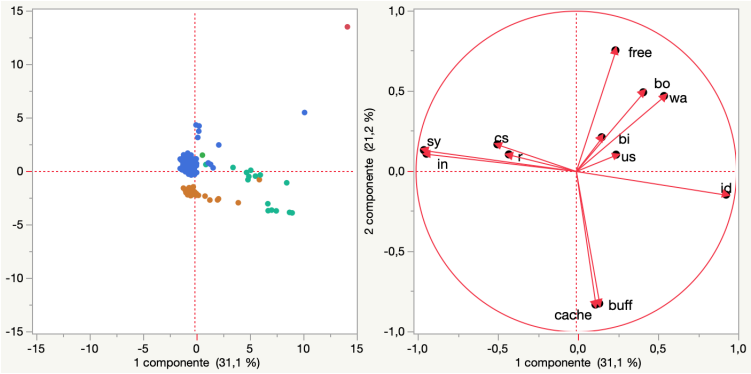


Figura 3.7: Scatter Plot e Loading Plot della PCA

Autovalori

Numero	Autovalore	Percentuale	20 40 60 80	Percentuale cumulativa
1	3,7267	31,056		31,056
2	2,5458	21,215		52,270
3	1,9388	16,157		68,427
4	1,3290	11,075		79,502
5	0,9499	7,916		87,419
6	0,5970	4,975		92,394
7	0,3745	3,121		95,515
8	0,3181	2,651		98,166
9	0,1820	1,517		99,683
10	0,0283	0,236		99,918
11	0,0098	0,082		100,000
12	0,0000	0,000		100,000

Figura 3.8: Autovalori

Osservando gli autovalori ottenuti, con l'obiettivo di conservare almeno il 95% della varianza del dataset di partenza, è stato scelto un numero di componenti principali pari a 7.

A questo punto è stata effettuata la clusterizzazione considerando le componenti principali.

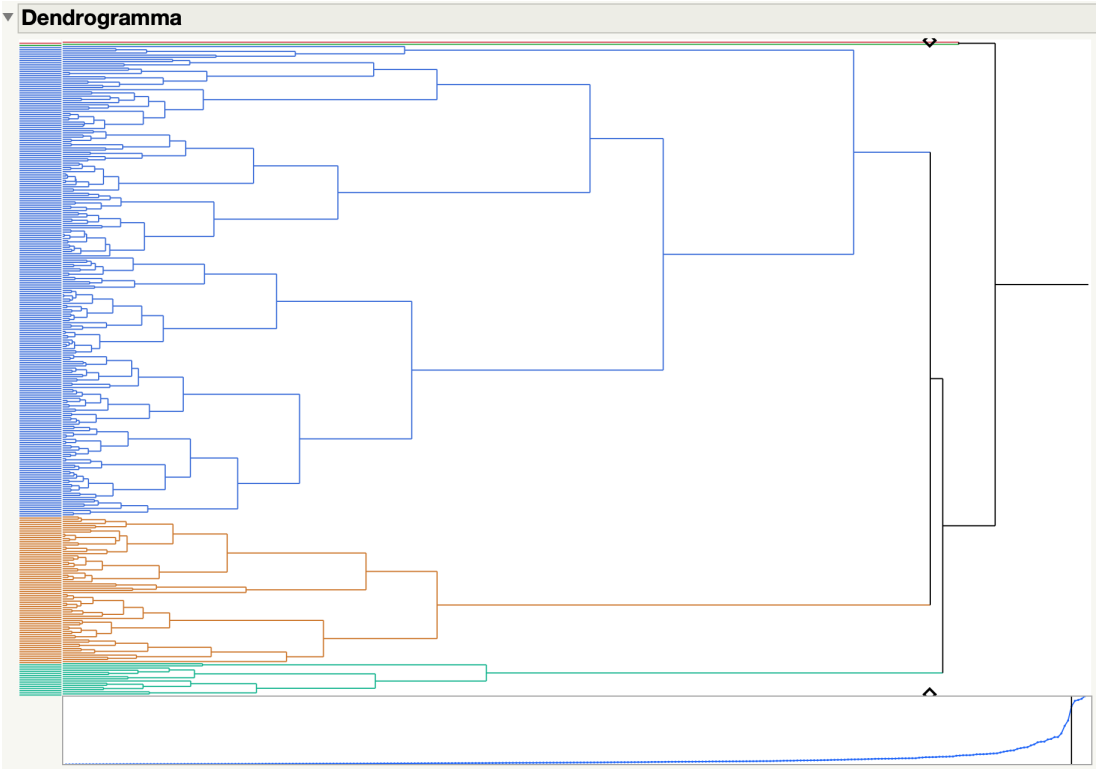


Figura 3.9: Dendrogramma

Cronologia di clusterizzazione

Numero di cluster	Distanza	Leader	Subordinato
16	4,90752284	19	96
15	5,40849848	230	250
14	5,54583665	9	13
13	5,57370761	83	105
12	6,09725842	2	3
11	6,22389260	15	23
10	6,67374480	9	26
9	6,67571938	230	232
8	7,55366107	8	83
7	9,40202050	9	19
6	10,70767703	9	15
5	14,10304189	2	9
4	15,46858839	2	230
3	15,69074419	2	8
2	15,97548224	1	229
1	16,62540674	1	2

Figura 3.10: Cronologia di clusterizzazione

Come nel capitolo 2, non avendo vincoli sul numero di esperimenti, si è applicato come criterio per la scelta del numero di cluster quello di osservare il salto maggiore della distanza all'aumentare del numero di cluster.

A questo punto si è proceduto al calcolo preciso della varianza persa a seguito prima della fase di PCA e poi di clusterizzazione, con lo stesso tool illustrato nel capitolo precedente. La varianza persa è pari a:

$$Var_{persa}(\%) = \frac{1242.276}{3461.454} \cdot 100 = 35.89\%$$

Quindi, del 95.515% di varianza spiegata dalla PCA, il clustering ne fa perdere il 35.89% (ne spiega quindi il 61.23%).

In conclusione, si riporta un esempio di workload sintetico:

r	b	swpd	free	buff	cache	si	so	bi	bo	in	cs
0	0	0	54392	14408	315964	0	0	189	405	3062	591
7	0	0	45180	15320	316148	0	0	544	0	4777	1274
4	0	0	46268	14772	316096	0	0	0	0	6129	686
12	0	0	44948	15336	316152	0	0	0	16	6268	678
0	0	0	45300	15416	316184	0	0	0	0	2044	561

Figura 3.11: Esempio di workload sintetico

us	sy	id	wa	st	Cluster
4	49	44	2	0	1
8	92	0	0	0	2
2	98	0	0	0	3
1	99	0	0	0	4
0	28	72	0	0	5

Figura 3.12: Esempio di workload sintetico

3.2.3 Capacity Test

Sulla base della caratterizzazione del workload eseguita, in questa sezione verranno riportati i risultati del capacity test del sistema descritto. In particolare, il capacity test è stato effettuato considerando un caso il più possibile vicino ad un contesto reale, caratterizzato da un'eterogeneità delle richieste.

Ai fini di aumentare il carico, il capacity test è stato condotto facendo variare il numero di thread, ognuno dei quali settato per fare 10 richieste al minuto; così facendo, è stato possibile far variare il numero di richieste totali inoltrate al server. I risultati del test si riportano in figura:

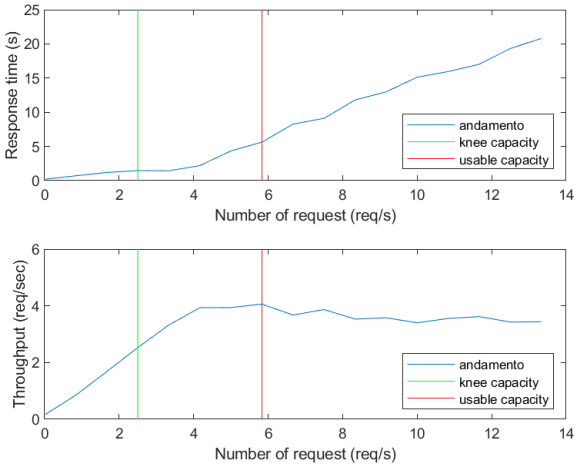


Figura 3.13: Capacity test su pagine miste

La knee capacity risulta essere di 3.33 req/s, mentre la usable capacity di 5.833 req/s.

A questo punto, il capacity test è stato condotto separatamente in base alle dimensioni delle pagine richieste. I risultati sono discussi nelle seguenti sezioni.

3.2.3.1 Capacity test pagine small

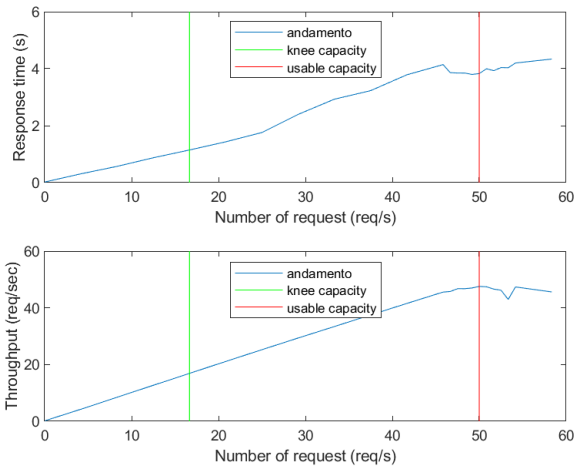


Figura 3.14: Capacity test su pagine small

3.2.3.2 Capacity test pagine medium

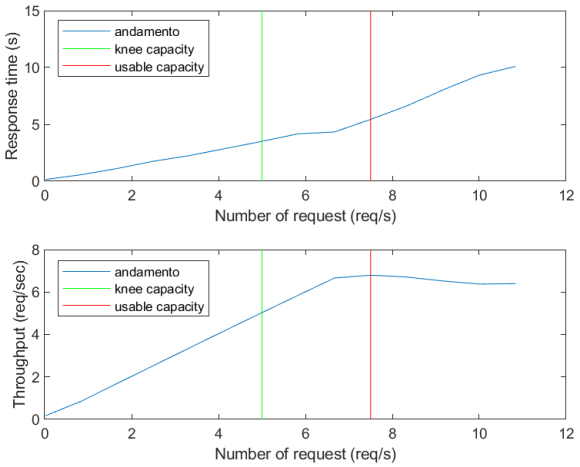


Figura 3.15: Capacity test su pagine medium

3.2.3.3 Capacity test pagine big

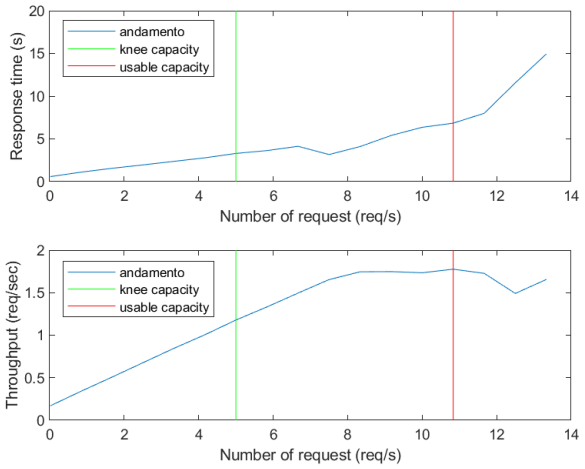


Figura 3.16: Capacity test su pagine big

In sintesi i valori ottenuti, compresi di media, sono riportati in tabella

Capacity test	Knee capacity (req/s)	Usable capacity (req/s)
Misto	2.5	5.83
Small	16.67	50
Medium	5	7.5
Big	5	10.83
Valori medi	7.29	18.54

Tabella 3.1: Valori di knee e usable capacity in sintesi

3.2.4 Design of experiment

I dati di knee capacity e usable capacity medi, recuperati nella sezione precedente sono stati utilizzati per effetuare un design of experiment (DOE), al fine di valutare l'importanza e la significatività dei fattori page type e request rate rispetto alle variazioni del response time.

Capitolo 4

Dependability

4.1 Esercizio 1

4.1.1 Traccia

Trovare la R(t) e l'MTTF per il sistema di cui viene fornito l'RBD. Nel calcolo dell'MTTF, assumere che tutti i componenti siano identici e falliscano randomicamente con failure rate λ.

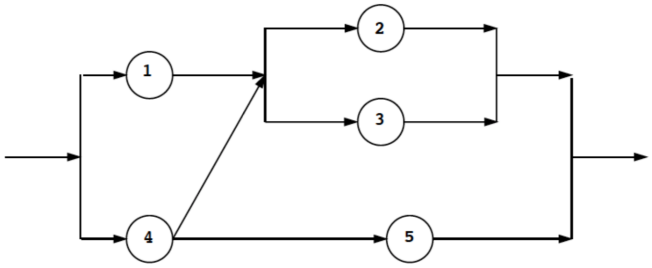


Figura 4.1: RBD

4.1.2 Soluzione

Da un'analisi del diagramma fornito, è possibile notare che i componenti 2 e 3 sono disposti in parallelo; possono essere dunque ridotti ad un unico blocco con reliability $R_{2||3} = 1 - (1 - R_2)(1 - R_3)$.

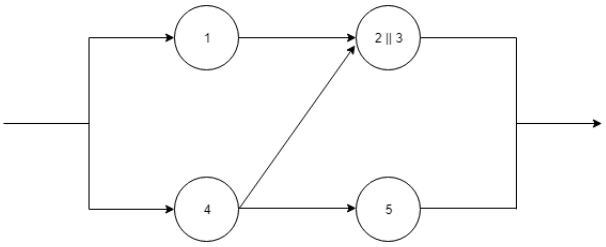


Figura 4.2: Sistema ridotto

Si osservi che non è possibile effettuare ulteriori riduzioni, per cui il diagramma che ne risulta è di tipo non-serie-parallelo. Per questi tipi di diagrammi, effettuiamo l'analisi dei success path; sappiamo infatti che la reliability del sistema risulta essere minore o uguale di quella del parallelo dei success path, che mostriamo di seguito:

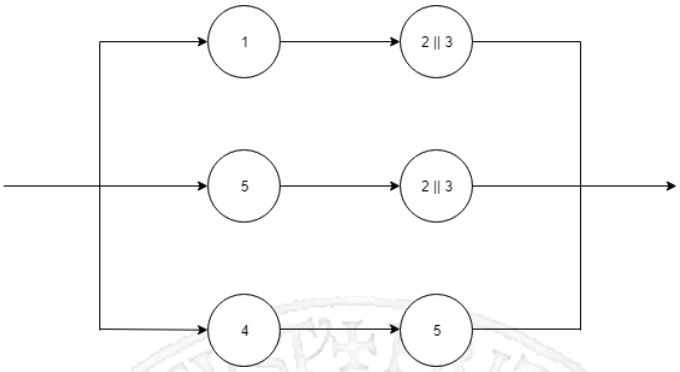


Figura 4.3: Success Path

Il limite superiore che individuiamo è il seguente:

$$R_{SYS} \leq 1 - (1 - R_1 R_{2||3})(1 - R_4 R_{2||3})(1 - R_4 R_5)$$

Per avere un valore preciso di Reliability, facciamo ricorso alla tecnica del conditioning, condizionando appunto il funzionamento del sistema a quello di un nodo. In particolare, si è scelto il nodo 4, in quanto si può notare che nel caso in cui esso funzioni (considerandolo dunque come corto circuito) o meno (considerandolo circuito aperto), è possibile effettuare una riduzione del sistema rispettivamente ad un parallelo ed una serie, che sono di più semplice trattazione.

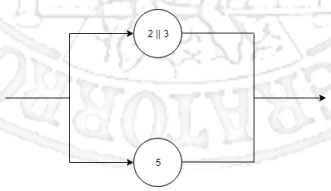


Figura 4.4: Caso in cui 4 funzioni

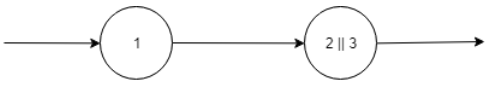


Figura 4.5: Caso in cui 4 non funzioni

Calcoliamo dunque le probabilità condizionate:

$$P(\textit{system works / 4 works}) = 1 - (1 - R_{2||3})(1 - R_5)$$

$$P(\textit{system works / 4 doesn't work}) = R_1 \cdot R_{2||3}$$

Sfruttando il teorema della probabilità totale, la reliability totale del sistema sarà data dalla somma della prima probabilità, moltiplicata per la reliability del componente 4, e la seconda, moltiplicata per l'unreliability del componente 4.

$$R_{SYS} = (1 - (1 - R_{2||3})(1 - R_5)) \cdot R_4 + (R_1 \cdot R_{2||3}) \cdot (1 - R_4)$$

Da traccia, tutti i nodi hanno la stessa reliability; supponiamo sia R.

$$R_{2||3} = 1 - (1 - R_2)(1 - R_3) = 1 - (1 - R)(1 - R) = 1 - (1 - R)^2 = -R^2 + 2R$$

$$R_{SYS} = (1 - (1 - (2R - R^2))(1 - R)) \cdot R + (R \cdot (2R - R^2))(1 - R) = 2R^4 - 6R^3 + 5R^2$$

Ora, sfruttando nuovamente l'ipotesi che tutti i componenti del sistema siano identici, ed assumendo che i fallimenti seguano andamento esponenziale con failure rate pari a λ ($R(t) = e^{\lambda t}$) possiamo calcolare:

$$MTTF_{SYS} = \int_0^\infty [2R^4 - 6R^3 + 5R^2]dt = \int_0^\infty [2e^{4\lambda} - 6e^{3\lambda} + 5e^{2\lambda}]dt = \frac{1}{2\lambda} - \frac{2}{\lambda} + \frac{5}{2\lambda} = \frac{1}{\lambda}$$

4.2 Esercizio 2

4.2.1 Traccia

Vogliamo confrontare due diversi modi di utilizzare la ridodanza per incrementare la reliability di un sistema. Supponiamo che il sistema abbia bisogno di s componenti identici in serie per funzionare in maniera appropriata. Supponiamo inoltre che ci vengano dati m x s componenti. Tra i due schemi, quale fornisce una maggiore reliability? Supposta r la reliability del singolo componente, derivare le espressioni delle reliability delle due configurazioni. Comparare le espressioni per m=3 e s=3.

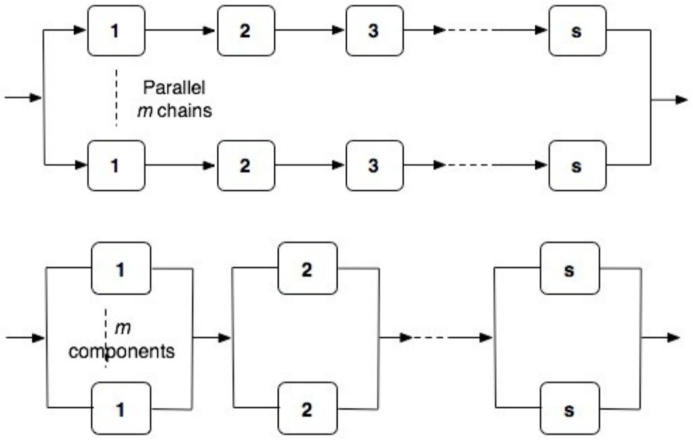


Figura 4.6: Schemi da comparare

4.2.2 Soluzione

L'obiettivo è quello di ricavare le espressioni per i due differenti schemi. Si può notare che il primo è costituito da serie di s componenti disposte su m linee collegate in parallelo. Abbiamo dunque un parallelo di serie, la cui espressione della reliability è la seguente:

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

Il secondo, invece, è composto da s "blocchi" in serie, ognuno dei quali costituito dal parallelo di m componenti. Ci troviamo quindi di fronte ad un sistema serie di paralleli, la cui espressione della reliabiltiy è la seguente:

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

Andando a settare s=m=3, si sono plottate le due funzioni con l'aiuto di MATLAB; il risultato è il seguente.

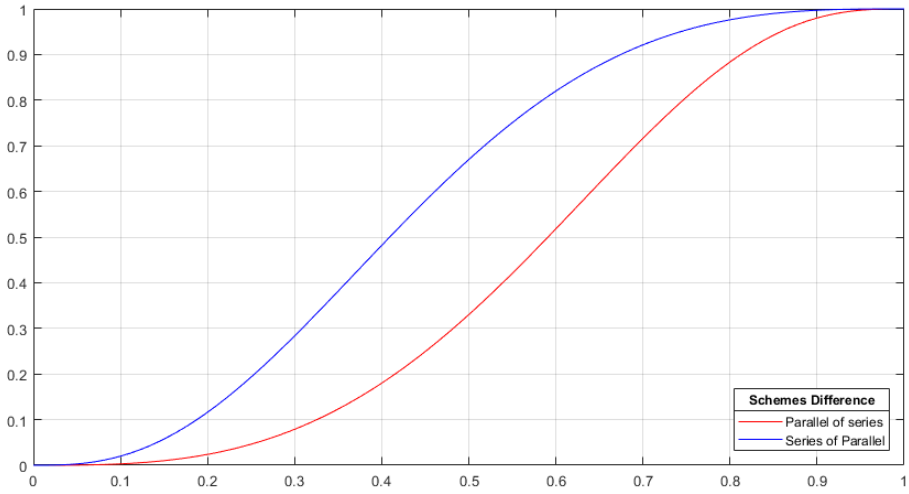


Figura 4.7: Confronto tra i due schemi

È possibile notare che la reliability del secondo schema è migliore.

4.3 Esercizio 3

4.3.1 Traccia

In figura è mostrata l'architettura di una rete di computer in un sistema bancario. L'architettura è chiamata "skip-ring network" and è progettata per permettere ai processori di comunicare anche dopo l'avvenimento di un failure in un nodo. Ad esempio, se il nodo 1 fallisce, il nodo 8 può bypassare il nodo fallito instradando i dati sul link alternativo che collega il nodo 8 con il 2. Assumendo che i link siano perfetti e i nodi abbiano ognuno una reliability R_m, derivare l'espressione per la reliability della rete. Se R_m segue la legge di fallimento esponenziale e il failure rate di ogni nodo è di 0.005 failure all'ora, determinare la reliability del sistema alla fine di un periodo di 48 ore.

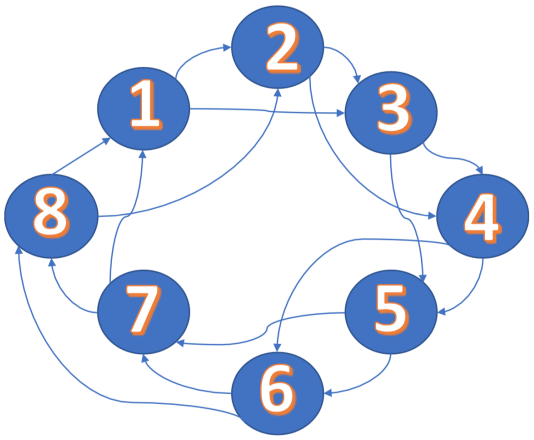


Figura 4.8: Rete del sistema bancario

4.3.2 Soluzione

Per il tipo di rete in esame, ci chiediamo quando il sistema può dichiararsi fallito. La particolarità di tale rete è che un nodo fallito può essere bypassato: il nodo raggiunto tramite il link alternativo dovrà necessariamente funzionare, pena il fallimento dell'intero sistema. Si può quindi dedurre che i fallimenti non devono riguardare nodi adiacenti. Fatta questa assunzione, quanti fallimenti possono avvenire contemporaneamente? Dati 8 nodi, il sistema continua a funzionare con al massimo 4 fallimenti, limite oltre il quale essi saranno per forza di cose adiacenti.

Lo schema è dunque un M-out-of-N System, in particolare un 4-out-of-8. La reliability in questo caso può essere espressa nel seguente modo:

$$R_{48} = \sum_{i=0}^{4} \binom{8}{i} R_m^{8-i}(1-R_m)^i$$

Dal coefficiente binomiale presente nella sommatoria, bisogna però escludere tutte le combinazioni non ammissibili di nodi.

$$R_{48} = \sum_{i=0}^{4} (\binom{8}{i} - e_i) R_m^{8-i} (1 - R_m)^i$$

Dove e_i rappresenta il numero di configurazioni da escludere.

i	e_i	Nota
0	0	Nessun nodo fallisce
1	0	Tutte le configurazioni sono ammissibili in quanto non esistono nodi adiacenti che falliscono
2	8	Bisogna escludere tutte le coppie di nodi adiacenti che falliscono, che sono 8
3	40	Bisogna escludere tutte le configurazioni in cui falliscono 3 nodi, di cui 2 adiacenti più un terzo che ruota (6*8=48 configurazioni). Da queste bisogna escludere però le ripetizioni di coppie di nodi che hanno un nodo in comune (8)
4	68	Le uniche due configurazioni ammissibili sono 1-3-5-7 e 2-4-6-8

Tabella 4.1: Configurazioni

Una volta fatto ciò è possibile calcolare R_{48}

$$R_{48} = R_m^8 + 8 \cdot R_m^7 \cdot (1 - R_m) + (28 - 8) \cdot R_m^6 \cdot (1 - R_m)^2 + (56 - 40) \cdot R_m^5 \cdot (1 - R_m)^3 + (70 - 68) \cdot R_m^4 \cdot (1 - R_m)^4$$

Tenendo conto che che R_m segue una legge di fallimento esponenziale tale che: $R_m = e^{-\lambda t}$, con $\lambda = 0.005$, abbiamo $R_m(48) \backsimeq 0.7866$, e la reliability del sistema in un periodo di 48h risulterà pari a:

$$R_{48} \backsimeq 0.728822$$

4.4 Esercizio 4

4.4.1 Traccia

Un'applicazione richiede in un sistema multiprocessore almeno tre processore debbano essere disponibili con probabilità maggiore del 99%. Il costro di un processore con una reliability dell'80% è di 1000$, e ogni incremento del 10% di reliability costerà 500$. Determinare il numero di processori (n) e la reliability (p) di ogni processore (assumendo che i processori abbiano la stessa reliability) che minimizzano il costo totale del sistema.

4.4.2 Soluzione

Dall'analisi del problema, si deduce che il sistema descritto è di tipo m-out-of-n. In particolare, affinché il sistema funzioni, è necessario il funzionamento di almeno 3 processori. L'obiettivo è di trovare il numero di processori per la configurazione a costo minimo con reliability maggiore di 0.99.

La reliability in questo tipo di sistema si calcola nel seguente modo:

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

Con R_m reliability del singolo processore, uguale per tutti i processori.

Al fine di trovoare la configurazione a costo minimo, è stato individuato il costo di ogni processore al crescere della reliability.

Reliability	Costo ($)
0.8	1000
0.9	1500
1.0	2000

Tabella 4.2: Costi dei processori

Per ogni configurazione possibile sono stati valutati, dato il numero minimo di processori funzionanti (M), l'affidabilità e il costo delle varie configurazioni al variare del numero dei processori del sistema (N). Di queste configurazioni, tra quelle che presentano reliability maggiore del 99%, si sceglie quella a costo minimo. A tale scopo è stato realizzato il seguente script MATLAB.

```
function [R,C,c_min,r_min] = reliabilityFunction(M,N)
R = zeros;
C = zeros;
p=[0.80, 0.88, 0.96];
for j=1:3
    for n=3:1:N
        acc=0;
        for i=0:n-M
            acc = acc + nchoosek(n,i)*(p(j)^(n-i))*((1-p(j))^i);
        end
```

```
11          R(n-2,j) = acc;
12          if(abs(p(j)-0.80)<eps)
13              C(n-2,j) = 1000*n;
14          end
15          if(abs(p(j)-0.88)<eps)
16              C(n-2,j) = 1500*n;
17          end
18          if(abs(p(j)-0.96)<eps)
19              C(n-2,j) = 2000*n;
20          end
21      end
22  end
23  c_min = min(C(R>0.99));
24  r_min = R(C==min(C(R>0.99)));
25  end
```

Codice Componente 4.1: Calcolo reliability e costi

I risultati sono riportati di seguito.

	Reliability singolo processore		
#processori	0.8	0.88	0.96
3	0.51200002	0.68147200	0.88473600
4	0.81919998	0.92680192	0.99090433
5	0.94208002	0.98568112	0.99939781
6	0.98303998	0.99745691	0.99996400
7	0.99532801	0.99957657	0.99999797
8	0.99876863	0.99993271	0.99999988
9	0.99968612	0.99998969	1
10	0.99992210	0.99999845	1

Tabella 4.3: Risultati di reliability

	Costi delle configurazioni		
#processori	0.8	0.88	0.96
3	3000	4500	6000
4	4000	6000	8000
5	5000	7500	10000
6	6000	9000	12000
7	7000	10500	14000
8	8000	12000	16000
9	9000	13500	18000
10	10000	15000	20000

Tabella 4.4: Risultati costi

La soluzione che garantisce il costo minimo e affidabilità maggiore del 99% è quella con numero di processori pari a 7, ognuno con reliability di 0.8. Il sistema complessivo presenta reliability pari a 0.99532801 con costo pari a 7000$.

4.5 Esercizio 5

4.5.1 Traccia

Il sistema mostrato in figura è un sistema di elaborazione per un elicottero. Il sistema ha una ridondanza duale sia per i processori che per le unità di interfacciamento. Vengono utilizzati due bus, ed ogni bus è replicato. La parte interessante del sistema è l'equipaggiamento di navigazione. Il velivolo può essere completamente guidato usato l'"Inertial Navigation System" (INS). Se tale INS fallisce, il velivolo può essere guidato attraverso una combinazione di Doppler e AHRS (Altitude Heading Reference System). Di quest'ultima unità, ne sono presenti 3, delle quali una sola è necessaria. I dati dal Doppler e un AHRS possono essere usati in sostituzione del componente INS se esso fallisce. A causa di altri sensori e strumentazioni, sono richiesti entrambi i bus affinché il sistema funzioni in maniera appropriata, indipendentemente dal sistema di navigazione utilizzato.

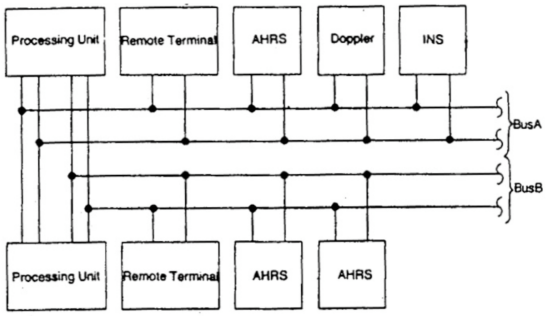


Figura 4.9: Architettura del sistema

1. Disegnare l'RBD del sistema
2. Disegnare il Fault Tree del sistema ed analizzare i minimal cutset
3. Calcolare la reliability per un'ora di volo utilizzando i valori di MTTF in tabella. Assumere che venga applicata la legge di fallimento esponenziale e che la fault coverage sia perfetta

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

Tabella 4.5: MTTF dei componenti

4. Ripetere il punto precedente, ma stavolta incorporare un fattore di coverage per la fault detection e riconfigurazione delle unità di elaborazione. Usando gli stessi dati di fallimento,

determinare il valore approssimativo di fault coverage richiesto per ottenere (alla fine dell'ora) una reliability di 0.99999

4.5.2 Soluzione

4.5.2.1 Punto 1

Dalla descrizione fornita, è possibile dedurre come ciascuna unità funzionale sia necessaria al corretto funzionamento del sistema. L'RBD sarà pertanto composto da una serie di componenti. In particolare, in corrispondenza di componenti replicati, all'interno della serie essi saranno disposti in parallelo tra di loro. Unica eccezione è fatta per il sistema di navigazione, in cui è necessario il funzionamento dell'INS, o in alternativa del Doppler in coppia con un AHRS.

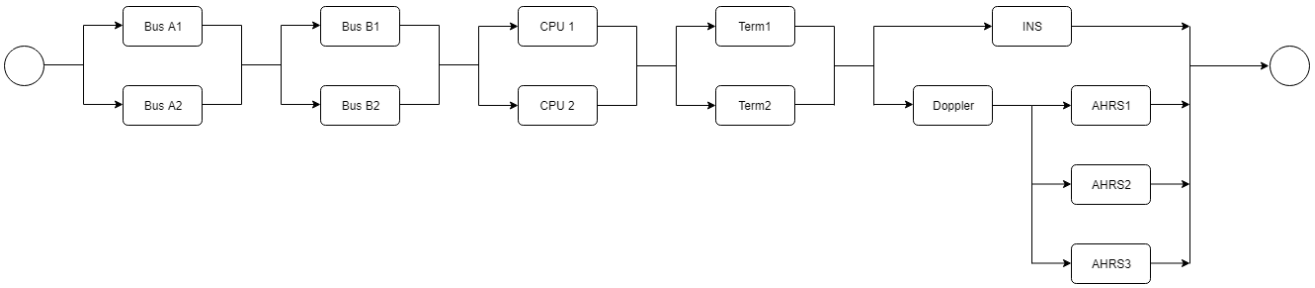


Figura 4.10: RBD del sistema

4.5.2.2 Punto 2

Per costruire il fault tree bisogna considerare i fallimenti del sistema. Si osserva dunque che il sistema fallisce in presenza di una delle seguenti condizioni:

- entrambi i bus di tipo A falliscono;
- entrambi i bus di tipo B falliscono;
- entrambe le CPU falliscono;
- entrambi i terminali falliscono;
- entrambi i sistemi di navigazione falliscono: ciò implica il fallimento dell'INS e o del Doppler o di tutti e 3 gli AHRS.

Date queste condizioni, è possibile ricavare la formula del fallimento:
$(BusA1 \wedge BusA2) \vee (BusB1 \wedge BusB2) \vee (CPU1 \wedge CPU2) \vee (Term1 \wedge Term2) \vee (INS \wedge (Doppler \vee (AHRS1 \wedge AHRS2 \wedge AHRS3)$
Da questa espressione booleana è possibile ricavare il fault tree.

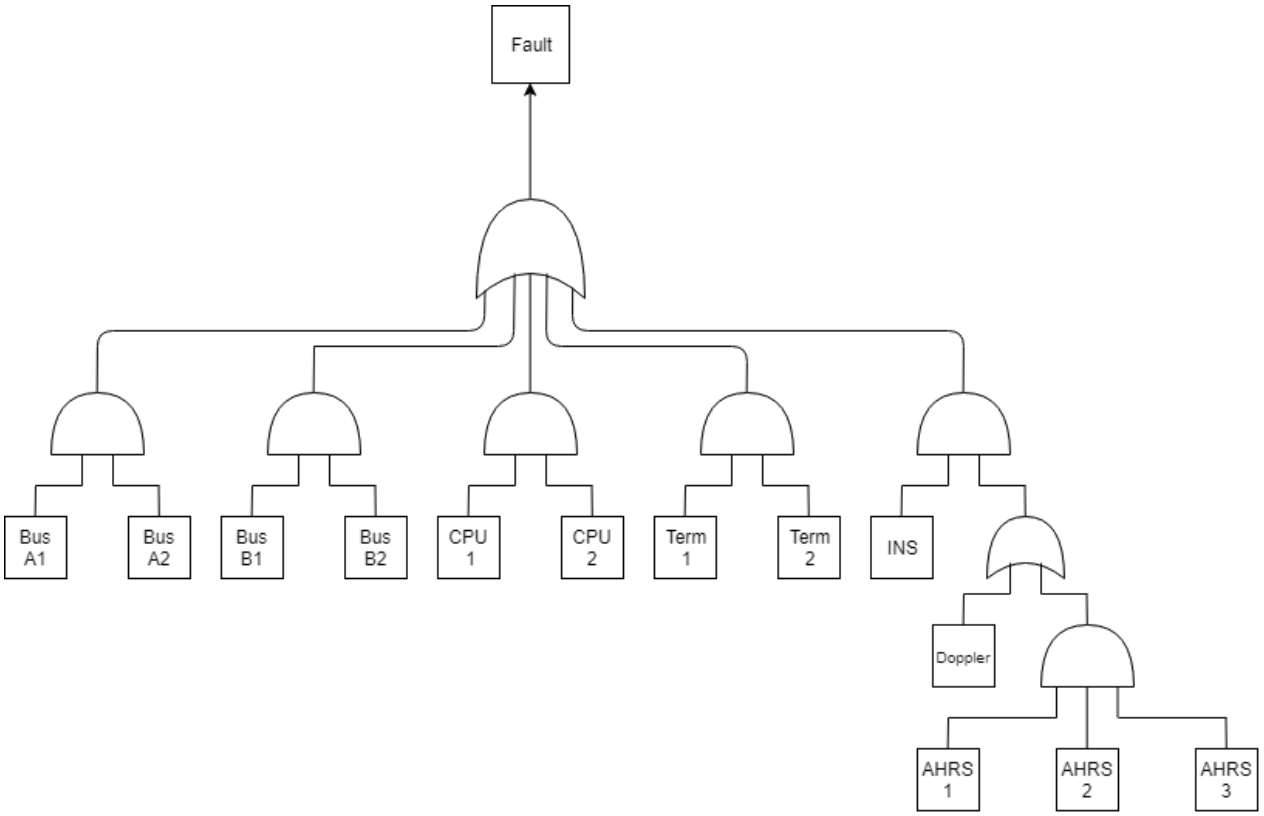


Figura 4.11: Fault Tree del sistema

Analizzando la figura si individuano i seguenti minimal cutset:

- {Bus A1, Bus A2}
- {Bus B1, Bus B2}
- {CPU 1, CPU 2}
- {Term 1, Term 2}
- {INS, Doppler}
- {INS, AHRS 1, AHRS 2, AHRS 3}

0.0.2.3 Punto 3

Si va adesso ad analizzare la reliability del sistema dopo un'ora di volo. È possibile calcolarE tale valore a partire dalle reliability dei singoli componenti; si inizia quindi a trovare il parametro λ relativo ad ogni componente a partire dalla tabella con i diversi MTTF.

Equipement	MTTF (hr)	λ
Processing Unit	5000	0.0002
Remote Terminal	2500	0.0004
AHRS	1000	0.001
INS	1000	0.001
Doppler	300	0.003
Bus	10000	0.0001

Tabella 4.6: MTTF dei componenti

A questo punto è possibile esprimere le diverse reliability dei componenti del sistema come:

$$R_{BUSAEQ} = 1 - (1 - R_{BUS})^2$$

$$R_{BUSBEQ} = 1 - (1 - R_{BUS})^2$$

Le quali rappresentano le reliability dei due sistemi formati dai paralleli dei Bus A e B.

$$R_{CPUEQ} = 1 - (1 - R_{CPU})^2$$

$$R_{TermEQ} = 1 - (1 - R_{BUS})^2$$

$$R_{AHRSEQ} = 1 - (1 - R_{AHRS})^3$$

Rappresentano gli altri sottosistemi formati dalla replicazioni di componenti uguali in parallelo (CPU, Terminal, AHRS).

$$R_{ALT} = R_{Doppler} R_{AHRSEQ}$$

$$R_{CS} = 1 - (1 - R_{INS})(1 - R_{ALT})$$

Le precedenti due reliability rappresentano la serie tra il Doppler e il parallelo dei tre componenti AHRS, quindi il sistema di controllo alternativo, e il parallelo di tale sisema con il sistema INS, in maniera tale da ottenere la reliability totale del sottosistema dedicato al controllo. A questo punto è possibile calcolare la reliability totale del sistema tramite la serie dei diversi sottosistemi sopra analizzati:

$$R_{sys} = R_{BUSAEQ} R_{BUSBEQ} R_{CPUEQ} R_{TermEQ} R_{CS}$$

Il risultato è stato ottenuto tramite il seguente script MATLAB:

```
format long

ycpu = 0.0002;
yterm = 0.0004;
yAHRS = 0.001;
yins = 0.001;
ydoppler = 0.003;
ybus = 0.0001;

t = 1;


    Rbus = exp(-(ybus.*t));
    Rcpu = exp(-(ycpu.*t));
    Rterm =exp(-(yterm.*t));
    R_AHRS = exp(-(yAHRS.*t));
    Rdoppler = exp(-(ydoppler.*t));
    Rins = exp(-(yins.*t));

    RbusA = 1-(1-Rbus).^2;
    RbusB = 1-(1-Rbus).^2;
    Rcpu_eq = 1-(1-Rcpu).^2;
    Rterm_eq = 1-(1-Rterm).^2;
    Ralt = Rdoppler.*(1-(1-R_AHRS).^3);
    R_cs = 1-(1-Rins).*(1-Ralt);
    Rsys = RbusA*RbusB*Rcpu_eq*Rterm_eq*R_cs;
    Rsys
```

Codice Componente 4.2: Reliabilty totale del sistema

Si ottiene quindi il valore della reliability totale del sistema $R_{sys} = 0.999996786066414$

4.5.2.3 Punto 4

Si supponga a questo punto di introdurre un circuito di detection per la rilevazione del fallimento delle CPU. La probabilità di tale circuito di rilevare un fallimento è indicata con il parametro c:

$$R_{sys} = R_1 + C(1 - R_1)R_2$$

Quindi nel caso del sottosistema formato dalle due CPU in parallelo avremo che:

$$C = \frac{(R_{detection} - R_{CPU})}{R_{CPU}(1 - R_{CPU})}$$

Si consideri adesso la reliability del sistema non includendo le CPU:

$$R_{sysNOCPU} = R_{BUSAEQ}R_{BUSBEQ}R_{TermEQ}R_{CS}$$

A tal proposito, si vuole trovare la probabilità c che ci permetta di avere una reliability del sistema almeno pari a 0.99999; per farlo scriviamo la relazione imponendo tale vincolo:

$$0.99999 = R_{detection} R_{sysNOCPU}$$

$$R_{detection} = \frac{0.99999}{R_{sysNOCPU}}$$

Una volta trovato tale valore lo si sostituisce all'interno della formula che esprime il valore di C.

Il procedimento illustrato è stato implementato nel seguente script MATLAB:

```
format long

ycpu = 0.0002;
yterm = 0.0004;
yAHRS = 0.001;
yins = 0.001;
ydoppler = 0.003;
ybus = 0.0001;

t = 1;


    Rbus = exp(-(ybus.*t));
    Rcpu = exp(-(ycpu.*t));
    Rterm =exp(-(yterm.*t));
    R_AHRS = exp(-(yAHRS.*t));
    Rdoppler = exp(-(ydoppler.*t));
    Rins = exp(-(yins.*t));

    RbusA = 1-(1-Rbus).^2;
    RbusB = 1-(1-Rbus).^2;
    Rcpu_eq = 1-(1-Rcpu).^2;
    Rterm_eq = 1-(1-Rterm).^2;
    Ralt = Rdoppler.*(1-(1-R_AHRS).^3);
    R_cs = 1-(1-Rins).*(1-Ralt);

    Rsys_noCPU = RbusA*RbusB*Rterm_eq*R_cs;
    Rsys_noCPU

    Rcpu_detection = 0.99999/Rsys_noCPU;
    C = (Rcpu_detection - Rcpu)/(Rcpu*(1-Rcpu));
    C
```

Codice Componente 4.3: Reliabilty totale del sistema

Il risultato è che il valore minimo per il parametro c deve essere: $C \backsimeq 0.96606$

Capitolo 5

FFDA

5.1 Traccia

1. Condurre l'analisi dei file di log MercuryErrorLog.txt e BGLErrorLog.txt per indirizzare i seguenti punti:

   - plot del conteggio di tuple per ogni CWIN;
   - raggruppamento delle entries per il CWIN scelto;
   - distribuzioni di reliability empiriche;
   - fit delle reliability empiriche (con i modelli esponenziale, iperesponenziale, weibull);
   - analisi dei modelli ottenuti tramite il K-S test.

   Comparare i risultati attraverso i sistemi.

2. Mercury e BG/L: selezionare i 5 nodi più "inclini" ad errori e determinare il parametro CWIN per ogni nodo:

   - il valore di CWIN è uguale per ogni nodo?
   - esistono colli di bottiglia per la dependability? (es. nodi con un alto numero di tuple se paragonati ad altri);
   - plottare la reliabilty a livello di nodo per i nodi che hanno un grande *interarrival* (>30);

3. Mercury: per ogni categoria di errore (ad esclusione di OTH) determinare CWIN, numero di tuple e modello di reliability:

   - qual è la categoria con il maggior numero di tuple?
   - quale categoria è più/meno affidabile?

   BG/L: analizzare le categorie J18-U01 e J18-U11 (conteggio di tuple e reliability).

4. Mercury e BG/L: individuare i nodi simili a livello funzionale (es. tg-c in Mercury o 2 nodi IO in BG/L, ecc) selezionati dalla top 5 dei nodi più inclini ad errore:

   - i nodi presentano numero di tuple o parametri di reliability simili?

- Mercury: estrarre le categorie di errore e i nodi che più contribuiscono a tali categorie. Cosa si può notare?
- BG/L: quali sono i rack/nodi più inclini ad errore?

5.2 Soluzione

5.2.1 Punto 1

Al fine di scegliere l'ampiezza della finestra temporale per effettuare l'analisi dei file di log forniti, è stato eseguito lo script *tupleCount_func_CWINpy.sh* che fornisce il numero di tuple per vari tentativi di ampiezza della finestra di coalescenza. I risultati ottenuti sono riportati sui seguenti grafici.

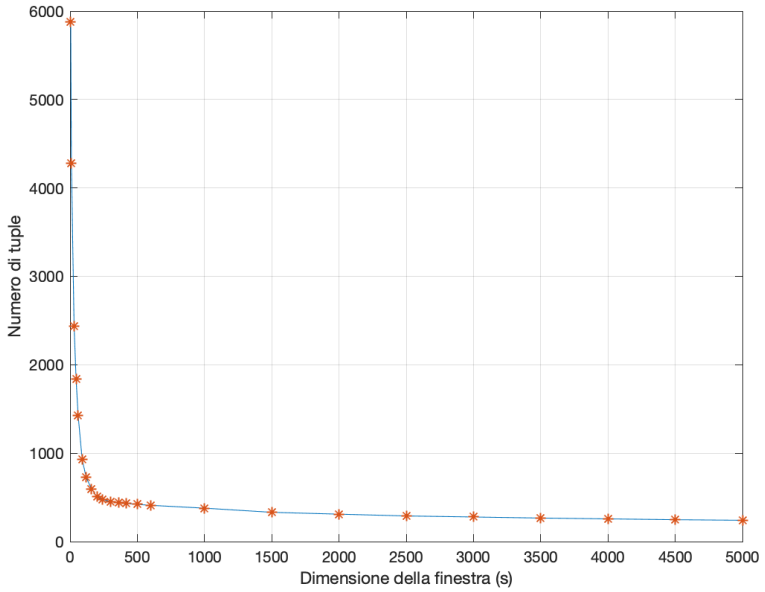


Figura 5.1: Conteggio tuple MercuryErrorLog

Per quanto riguarda il file di log *MercuryErrorLog.txt*, si può osservare che la curva presenta un ginocchio; la scelta dell'ampiezza della finestra è guidata dalla necessità di catturare il maggior numero di tuple che siano relative allo stesso fault con la minore ampiezza della finestra temporale possibile. Dunque si è optato di scegliere un valore della finestra, prossimo al ginocchio, di 360 secondi che cattura 440 tuple.

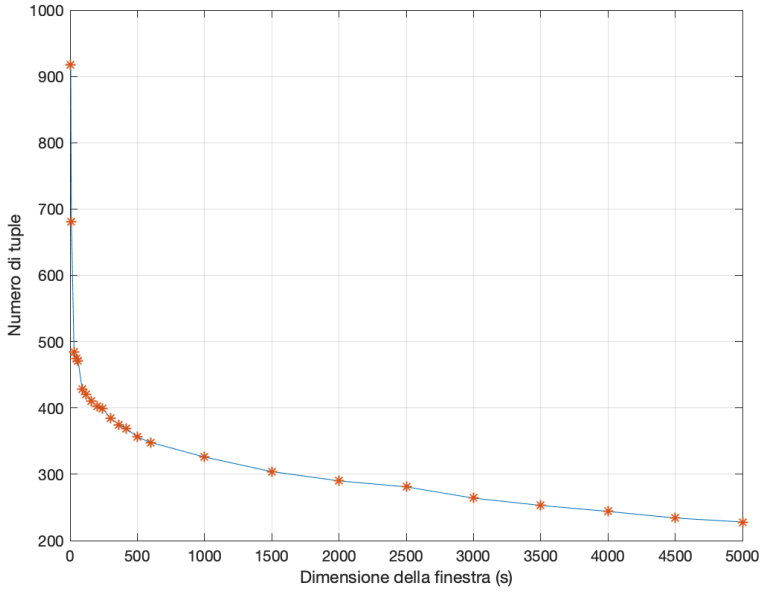


Figura 5.2: Conteggio tuple BGL

Nel caso del file *BGLErrorLog.txt*, è possibile osservare come il ginocchio della curva sia meno definito; dunque si è optato di scegliere un valore della finestra che non fosse troppo elevato, in quanto è sempre preferibile avere troncamenti piuttosto che collisioni. Il valore scelto è di 420 secondi che cattura 369 tuple.

Dopo aver scelto il valore di ampiezza per la finestra di coalescenza, viene effettuato l'effettivo raggruppamento, lanciando lo script *tupling_with_CWIN.sh* con parametro il log per cui raggruppare le tuple e il valore di ampiezza scelto. Dei vari output dello script, assume particolare importanza ai fini dell'analisi il file *interarrivals.txt*, che contiene i diversi tempi che intercorrono tra una tupla e la successiva. Tali tempi rappresentano proprio il TTF "empirico" (in quanto ottenuto esclusivamente a partire da delle osservazioni). A questo punto, si possono dunque ottenere le distribuzioni delle reliability empiriche tramite MATLAB. Tale script, oltre a fornire la distribuzione di probabilità empirica (plot in alto a sinistra), calcola anche il fit per i modelli esponenziale, iperesponenziale e weibull.

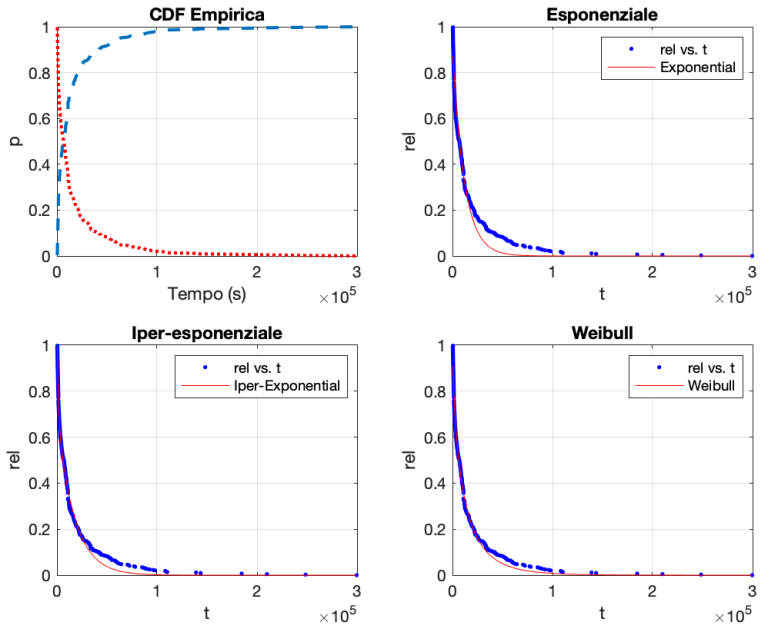


Figura 5.3: Fit della reliability Mercury

Al fine di valutare la bontà dei risultati ottenuti, si è effettuato il test di Kolmogorov-Smirnov. Per effettuare il test si è utilizzata la funzione MATLAB kstest2 i cui risultati sono mostrati nella tabella sottostante. Si verifica subito che nel caso dell'esponenziale l'ipotesi nulla è rigettata (H = 1). Questo ci consente di dire che la nostra distribuzione non può essere descritta da un'esponenziale semplice. L'ipotesi nulla del test non è invece rigettata per i modelli dell'iperesponenziale e per la Weibull (H=0); per scegliere tra questi due modelli si vanno quindi a valutare altri parametri quali p-value, SSE, R-square. Questi favoriscono tutti l'iperesponenziale rispetto alla Weibull, infatti il valore del p-value della Weibull è più vicino alla soglia (0.05) per il rigetto dell'ipotesi nulla. Per quanto riguarda la devianza dell'errore di regressione e l'R-square, avremo anche in questo caso valori migliori per l'iperesponenziale rispetto alla Weibull. Si può dunque concludere che il modello iperesponenziale è, tra quelli proposti, il migliore per rappresentare la distribuzione.

Modello	Funzione	Parametri	Intervalli di confidenza al 95%	H	P	Goodness of fit
Esponenziale	$a \cdot e^{bx}$	$a = 0.9011$ $b = -8.484e-05$	$a \in (0.891, 0.9112)$ $b \in (-8.746e-05, -8.221e-05)$	1	0.002	SSE: 1.081 R-square: 0.9693 Adjusted R-square: 0.9693 RMSE: 0.05117
Iperesponenziale	$a \cdot e^{bx} + c \cdot e^{dx}$	$a = 0.4125$ $b = -0.0008358$ $c = -0.0008358$ $d = -5.883e-05$	$a \in (0.3978, 0.4272)$ $b \in (-0.0008995, -0.0007721)$ $c \in (0.6976, 0.7205)$ $d \in (0.6976, 0.7205)$	0	0.71206	SSE: 0.1417 R-square: 0.996 Adjusted R-square: 0.996 RMSE: 0.01857
Weibull	$e^{-(lx)^a}$	$a = 0.7066$ $l = 9.427e-05$	$a \in (0.6952, 0.718)$ $l \in (9.274e-05, 9.581e-05)$	0	0.0833	SSE: 0.3687 R-square: 0.9895 Adjusted R-square: 0.9895 RMSE: 0.02988

Tabella 5.1: Risultati fitting Mercury

Dai risultati, è possibile notare che l'unico modello che rigetta l'ipotesi nulla è quello esponenziale. La scelta del miglior fit ricade dunque sui restanti due modelli. L'iperesponenziale ha un p-value di 0.71206

5.2.2 Punto 2

Per adempiere alla seconda richiesta, la prima cosa da fare è stata quella di individuare i nodi a cui corrisponde il maggior numero di entries. Ciò è stato ottenuto tramite lo script *logstatistics.sh*, che suddivide le entries totali del log file sia per categoria di errore sia per nodo. Tale script è stato lanciato per entrambi i log file e sono stati estratti i 5 nodi con il maggior numero di occorrenze.

Nodo	Entries
tg-c401	62340
tg-master	4098
tg-c572	4030
tg-s044	3224
tg-c238	1273

Tabella 5.2: Entries per i nodi di MercuryErrorLog

Nodo	Entries
R71-M0-N4	1716
R12-M0-N0	1563
R63-M0-N2	976
R03-M1-NF	960
R63-M0-N0	791

Tabella 5.3: Entries per i nodi di BGLErrorLog

Una volta individuati tali nodi si è poi proceduto a filtrare i file di log forniti, selezionando da questi solo le entries relative ai nodi precedentemente estratti. A tale scopo è stato utilizzato lo script *filtering.sh*.

A questo punto, bisogna determinare, per questi nodi, l'ampiezza della finestra di coalescenza con relativo conteggio delle tuple, proprio come fatto nell'esercizio precedente.

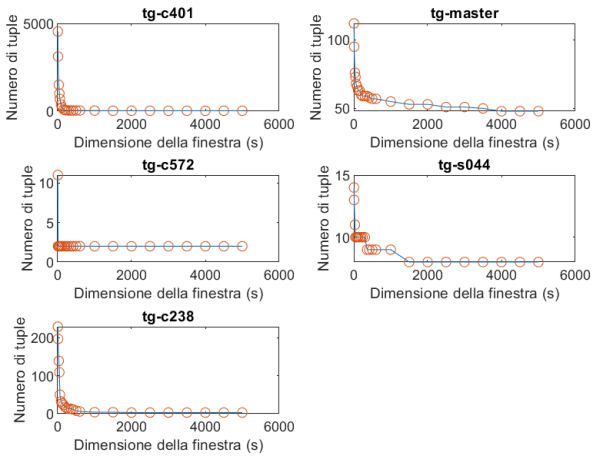


Figura 5.4: Count tuple nodi Mercury

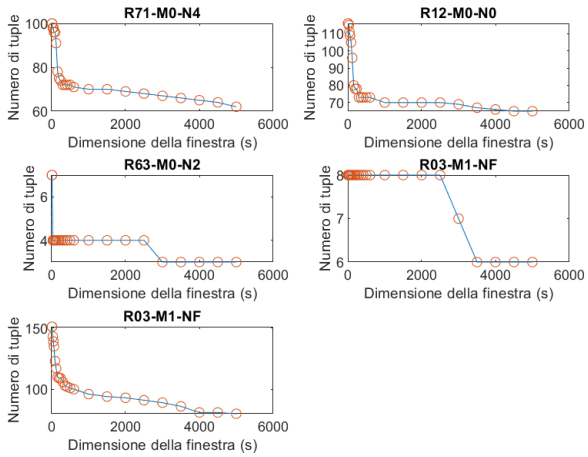


Figura 5.5: Count tuple nodi BGL

A partire dai grafici ottenuti, molto diversi tra di loro, è stata fatta una scelta del parametro CWIN ad hoc per ogni nodo.

Nodo	CWIN Scelto	#tuple
tg-c401	240	48
tg-master	240	59
tg-c572	30	2
tg-s044	1000	9
tg-c238	600	6

Tabella 5.4: CWIN per i nodi del log Mercury

Nodo	CWIN Scelto	#tuple
R71-M0-N4	600	71
R12-M0-N0	600	73
R63-M0-N2	2500	4
R03-M1-NF	3500	6
R63-M0-N0	1000	96

Tabella 5.5: CWIN per i nodi del logBGL

Si può osservare che esistono per entrambi i log file dei nodi particolarmente critici per la dependability:

1. per il file Mercury, i nodi tg-c401 e tg-master presentano un numero di tuple molto più elevato rispetto agli altri;

2. allo stesso modo, per il file BGL, i nodi R71-M0-N4, R12-M0-N0 e R63-M0-N0 presentano un numero di tuple altrettanto elevato, costituendo di fatto dei colli di bottiglia per la dependability.

Per i nodi che presentano un numero di tuple maggiore di 30 si vuole plottare la reliability. A tal fine, come nell'esercizio precedente, è stato utilizzato lo *script tupling_with_CWIN.sh* che prende in ingresso il log file e il valore scelto per la finestra.

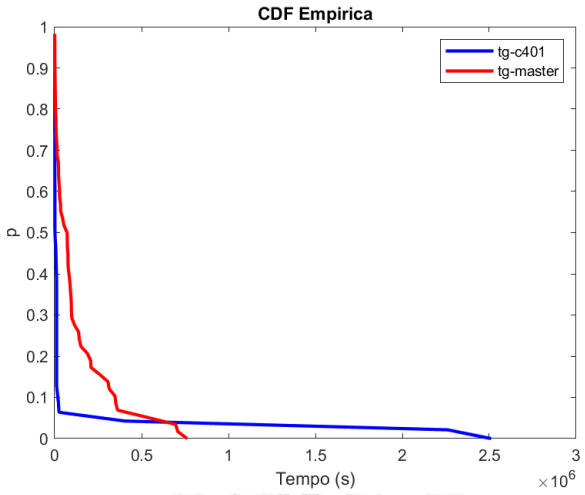


Figura 5.6: Reliability nodi Mercury

Per quanto riguarda il file Mercury, si evince come i due nodi individuati come i più critici abbiano una bassa reliability: ma in particolare è la reliability del nodo master che crolla a picco, il che conferma la deduzione fatta in precedenza riguardo al suo essere un collo di bottiglia per la dependability del sistema.

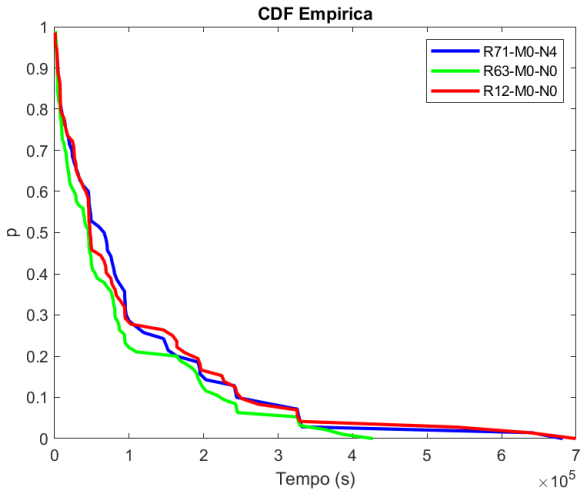


Figura 5.7: Reliability nodi BGL

Per il file BGL, i tre nodi hanno reliability sperimentali molto simili tra loro, nonostante ci sia una curva (relativa al nodo R63-M0-N0) che presenta una pendenza più ripida e raggiunge il fallimento prima delle altre.

5.2.3 Punto 3

Sulla falsa riga dell'esercizio precedente, si è condotta la stessa analisi con la differenza che sono state considerate le entries raggruppate per categoria di errore.

Categoria	Entries
DEV	57248
I-O	12819
MEM	5547
NET	3702
PRO	1504

Tabella 5.6: Entries per categorie di MercuryErrorLog

Categoria	Entries
J18-U01	50055
J18-U11	49932

Tabella 5.7: Entries per categorie di BGLErrorLog

Sempre con lo stesso approccio, si sono filtrati i file di log secondo le categorie scelte, sia di Mercury che di BGL. A questo punto si è passati alla scelta della finestra di coalescenza.

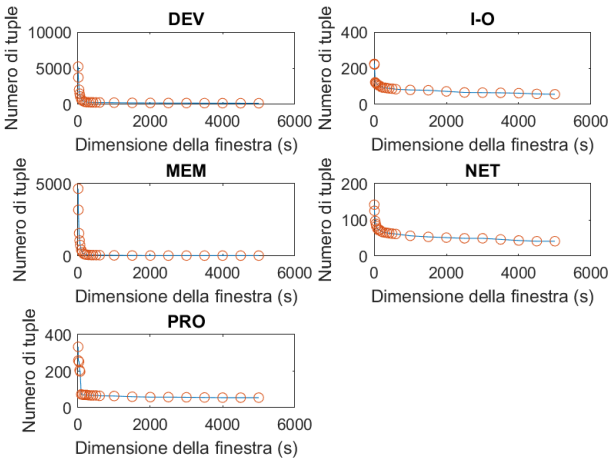


Figura 5.8: Count tuple categorie Mercury

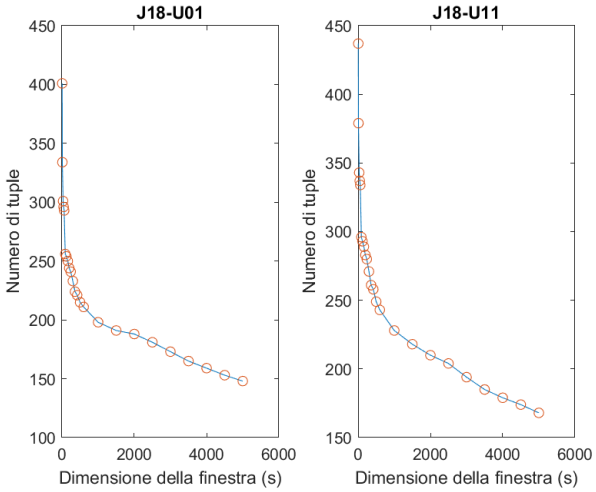


Figura 5.9: Count tuple categorie BGL

A partire dai grafici ottenuti, sono stati scelti, per tutte le categorie considerate, i valori di CWIN subito dopo il ginocchio della curva.

Categoria	CWIN Scelto	#tuple
DEV	600	225
I-O	600	84
MEM	500	62
NET	600	61
PRO	600	65

Tabella 5.8: CWIN per categorie del log Mercury

A prima vista, la categoria DEV sembrerebbe quella più critica per la dependability, in quanto con un valore di CWIN pressocché uguale a tutti gli altri presenta un numero di tuple molto maggiore.

Categoria	CWIN Scelto	#tuple
J18-U01	2000	188
J18-U11	2000	210

Tabella 5.9: CWIN per categorie del log BGL

La scelta del valore di CWIN per le due categorie di errore del file BGL è la stessa, in quanto i grafici che mostrano il numero di tuple al variare dei tentativi di dimensione della finestra hanno lo stesso andamento. Anche i numeri di tuple risultani sono molto simili.

Anche in questo caso sono state plottate le CDF empiriche delle reliability delle diverse categorie per entrambi i log-file.

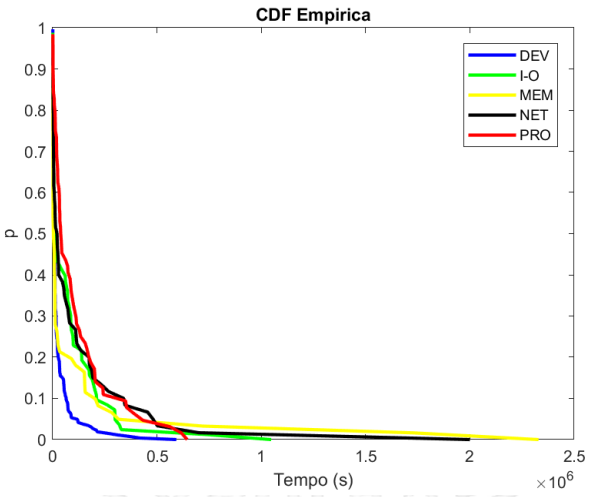


Figura 5.10: Reliability categorie Mercury

L'analisi del grafico mostra come, per il file Mercury, la categoria DEV sia più problematica per la reliability. Le categorie più affidabili risultano NET e PRO.

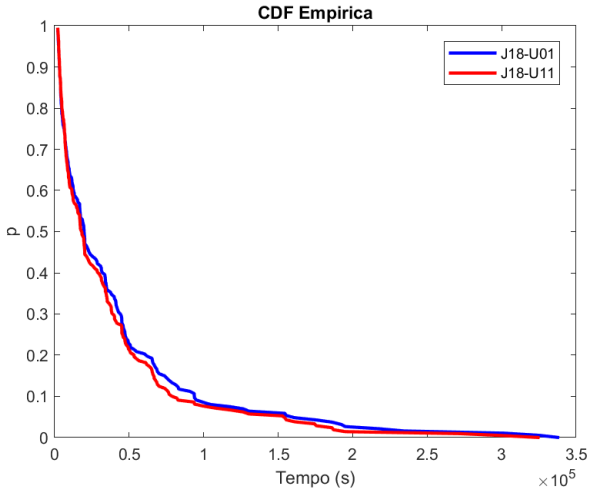


Figura 5.11: Reliability categorie BGL

Per il file BGL, le distribuzioni di reliability confermano la somiglianza delle due categorie d'errore già notata durante il calcolo dell'ampiezza ottima di CWIN e pertanto non si individua una categoria d'errore maggiormente affidabile rispetto all'altra.

5.2.4 Punto 4

5.2.4.1 Mercury

Per quanto riguarda i nodi che più contribuiscono a ciascuna categoria di errore, si riportano in tabella (per ogni categoria considerata) il numero di entries nel log e, per ogni categoria, i cinque nodi con il più alto numero di entries. Nella cella di ogni nodo, si riporta anche la sua percentuale di contribuzione alla categoria.

Categoria	#entries	Nodo 1	Nodo 2	Nodo 3	Nodo 4	Nodo 5
DEV	57248	tg-c401 - 89%	tg-c572 - 6%	tg-c238 - 2%	tg-c242 - 2%	/
MEM	12819	tg-c401 - 90%	tg-c572 - 7%	tg-c238 - ~1%	/	/
I-O	5547	tg-s044 - 58%	tg-master - 8%	tg-login3 - 7%	tg-s038 - 4%	tg-c550 - 4%
NET	3702	tg-master - 98%	/	/	/	/
PRO	1504	tg-c648 - 41%	tg-c324 - 16%	tg-c284 - 12%	tg-c451 - 11%	tg-c447 - 9%

Tabella 5.10: Nodi che più contribuiscono alle categorie di Mercury

Si può notare come le categorie che presentano più errori siano DEV e MEM. Inoltre, gli errori avvengono negli stessi nodi con pressocchè la stessa percentuale, quindi si può immaginare che vi sia una correlazione tra le due categorie di errore e che possano essere assunte come relative agli stessi fault. Un'ulteriore particolarità è che per la categoria NET il nodo tg-master rappresenta la quasi totalità degli errori. Inoltre, la categoria I-O è l'unica a riguardare nodi di tipo tg-s e login, contrariamente alle altre, relative principalmente a nodi di tipo tg-c.

5.2.4.2 BGL

Per analizzare quali fossero i rack più inclini ad errore, il file di log BGL è stato filtrato in modo tale da ottenere i rack tramite la seguente direttiva bash:

```
cat BGLErrorLog.txt | awk '{print $2}' | sed -e
's/\(-M[0-9]-N.\)*$//g' >> BGLErrorLog_filterByRacks.txt
```

Tale file è stato poi importato in JMP, dove sono state analizzate le occorrenze di ciascun rack:

Figura 5.12: Conteggio rack BGLErrorLog

Visivamente, si può notare come i rack più critici siano quelli la cui occorrenza è superiore al 2% del conteggio totale delle entries. In particolare, il rack R63 è quello che presenta da solo il 6% degli errori totali, cosa che dall'analisi iniziale non emergeva in quanto fatta considerando i singoli nodi e non l'intero rack.