|  |
| --- |
|  |

|  |
| --- |
| **IMPIANTI DI ELABORAZIONE** |
| Prof. Domenico Cotroneo, Corso di studi Ingegneria Informatica LM, Università degli studi Federico II di Napoli |
|  |
| **Caliendo Giuseppe M63/639**  **Gentile Maurizio M63/648** |
|  |
|  |
| **A.A 2016/2017** |
|  |

Sommario

[1. Sieve 2](#_Toc475948899)

[1. Descrizione dell’algoritmo e dei sistemi testati 2](#_Toc475948900)

[2. Analisi e considerazioni preliminari 6](#_Toc475948901)

[3. Analisi statistica dei dati 7](#_Toc475948902)

[4. Confronto finale 11](#_Toc475948903)

[2. Dataset Reduction 13](#_Toc475948904)

[3. Web Server 24](#_Toc475948905)

[1. Descrizione del problema 24](#_Toc475948906)

[2. Workload Characterization 24](#_Toc475948907)

[2.1 WL Parametri App\_level 25](#_Toc475948908)

[2.2 WL Parametri Low\_level 31](#_Toc475948909)

[3. Capacity Test 34](#_Toc475948910)

[4. Design of Experiments 37](#_Toc475948911)

[5. Performance Degradation Analysis 41](#_Toc475948912)

[4. Esercizi Reliability 47](#_Toc475948913)

[1. Exercise 1 47](#_Toc475948914)

[2. Exercise 2 51](#_Toc475948915)

[3. Exercise 3 53](#_Toc475948916)

[4. Exercise 4 58](#_Toc475948917)

[5. Field Failure Data Analysis 60](#_Toc475948918)

# Sieve

## Descrizione dell’algoritmo e dei sistemi testati

L’algoritmo di Sieve di Eratostene è un algoritmo atto al calcolo dei numeri primi presenti all’interno di una sequenza di interi positivi.

La sua importanza risiede nel fatto che tale algoritmo risulta essere molto utilizzato come test Workload nell’ambito di sistemi computer.

Il termine giusto per descrivere tale algoritmo è infatti “applicazione di Benchmark” e l’utilizzo che se ne fa è appunto quello di benchmarking di sistemi diversi. Esso presenta infatti proprietà quali la riproducibilità, scalabilità e la portabilità che fanno i modo che, ripetuto su sistemi differenti, possa fornire una valutazione più o meno oggettiva di tali sistemi in base al valore del **tempo di esecuzione** e permettere dunque anche la loro comparazione.

L’algoritmo, data la sequenza di valori in ingresso, di dimensione fissata N, opera tramite un ciclo con un indice j che va da 1 fino a ed elimina dall’intera sequenza tutti i multipli di j.

In questa sede si effettuerà un’analisi dei tempi di esecuzione dell’algoritmo facendo variare vari fattori.

L’architettura di esecuzione sarà unica e riguarderà un notebook Samsung NP-RV520 con processore Intel Core i3-2310M da 2.10 GHz e 4Gb di Memoria RAM.

Questo elaboratore ospita 2 sistemi operativi partizionati: Linux Debian 8.2 Jessie a 64 bit e Windows 7.

Il Sieve sarà utilizzato per testare sostanzialmente **il sistema operativo Debian Jessie** e lo stesso sistema **Debian ospitato** però **da macchina virtuale WMWare eseguita su Windows7**, in modo tale da valutare le performance e le differenze tra l’operare con Debian effettivo e Debian simulato da un programma di virtual emuling potente come WMWare.

Allo stesso tempo, i sistemi saranno testati con il Sieve scritto in due linguaggi di programmazione differenti (e quindi compilato da due compilatori differenti) e cioè **Java** e **C++**.

Un ulteriore parametro che sarà fatto variare sarà **N**, inteso come dimensione dell’array di interi sul quale il Sieve deve operare. Saranno valutati valori di N pari rispettivamente a **10.000, 100.000 e 1.000.000**.

In definitiva abbiamo 2 sistemi da testare, 2 linguaggi di programmazione e 3 valori di N per un totale di 12 combinazioni differenti.

L’obbiettivo di tale elaborato è quello di cercare di caratterizzare statisticamente i tempi di esecuzione per ognuna delle seguenti combinazioni, prelevando campioni di dati ottenuti eseguendo più volte il programma chiamante la funzione di Sieve e prelevando i tempi tramite le funzioni *getTimeOfDay* in C++ e *System.nanoTime* in Java.

Effettuata questa descrizione statistica, tramite indici appropriati e intervalli di confidenza, si procederà infine a effettuare un’analisi del confronto tra i tempi in base ai parametri testati.

Di seguito si lasciano le implementazioni C++ e Java per l’algoritmo Sieve di Eratostene utilizzate in tutte le prove descritte nei paragrafi successivi.

void SieveOfEratosthenes(int n)

{

// Create a boolean array "prime[0..n]" and initialize

// all entries it as true. A value in prime[i] will

// finally be false if i is Not a prime, else true.

//bool prime[n+1];

bool\* prime= (bool\*)(malloc((n+1)\*sizeof(bool)));

memset(prime, true, sizeof(prime));

for (int p=2; p\*p<=n; p++)

{

// If prime[p] is not changed, then it is a prime

if (prime[p] == true)

{

// Update all multiples of p

for (int i=p\*2; i<=n; i += p)

prime[i] = false;

}

}

/\*// Print all prime numbers

for (int p=2; p<=n; p++)

if (prime[p])

cout << p << " ";\*/

free(prime);

}

*Implementazione Sieve C++*

*Implementazione Sieve Java*

**public** **static** **void** sieveOfEratosthenes(**int** n) {

// Create a boolean array "prime[0..n]" and initialize

// all entries it as true. A value in prime[i] will

// finally be false if i is Not a prime, else true.

**boolean** prime[] = **new** **boolean**[n+1];

**for**(**int** i=0;i<n;i++)

prime[i] = **true**;

**for**(**int** p = 2; p\*p <=n; p++) {

// If prime[p] is not changed, then it is a prime

**if**(prime[p] == **true**) {

// Update all multiples of p

**for**(**int** i = p\*2; i <= n; i += p)

prime[i] = **false**;

}

}

/\*// Print all prime numbers

for(int i = 2; i <= n; i++) {

if(prime[i] == true)

System.out.print(i + " ");

} \*/

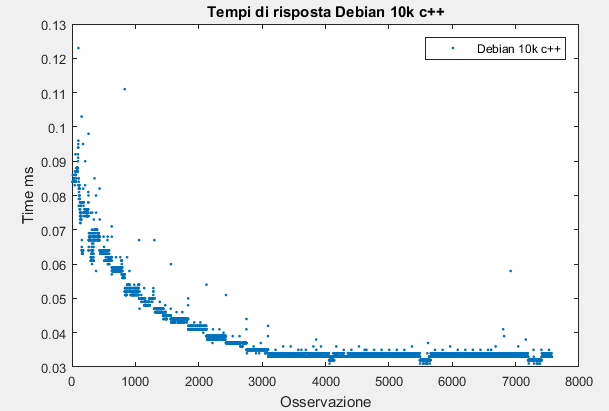
}

## Analisi e considerazioni preliminari

Effettuare tutto lo studio proposto non è stato facile. A valle di tutto è bene inserire alcune considerazioni e assunzioni preliminari che sono state fatte, onde evitare fraintendimenti.

In primo luogo abbiamo eseguito un paio di combinazioni in C++ su Debian prelevando campioni di 10.000 valori all’ interno di un unico ciclo soltanto per cominciare a vedere la natura dei dati e per ravvisare se si potesse presentare qualche comportamento anomalo.

In effetti abbiamo notato in tale campione, come si può vedere dal grafico che mostra l’andamento dei valori, la presenza di un **trend** esponenziale negativo con valori alti in una fase di “transitorio” iniziale che poi si stabilizzano in una serie di valori non troppo distanti tra di loro.



*Figura 1 Trend esponenziale*

Abbiamo analizzato questo problema e da qui abbiamo capito che tale trend dipendesse dal tempo da cui il programma era in esecuzione, per cui, dopo essere stato caricato in memoria e avere attraversato una fase “lenta” di esecuzione per un breve periodo, potesse poi eseguire nel pieno delle possibilità offerte dall’ambiente di esecuzione.

Da qui abbiamo pensato che fosse interessante testare i sistemi non considerando appunto questa fase di transitorio iniziale, in quanto risultasse qualcosa di indipendente dal Sieve. A tal fine si è inserita una funzione di “perditempo” (realizzata tramite banali cicli for di attesa) all’interno di tutti i programmi per **prendere i valori dei campioni soltanto dopo un certo tempo di esecuzione del programma.**

Questo ha definitivamente eliminato il trend e ovviamente si è estesa tale opera a tutte le combinazioni (anche Java) per rendere uniforme ed equa l’analisi predisposta.

Un’ulteriore assunzione prima di passare all’analisi dei dati è stata quella di effettuare le esecuzioni in Java disabilitando il JIT Compiler tramite l’opzione di esecuzione “-Djava.compiler=NONE”. Questo perché si è notato, tramite esecuzioni di prova preliminari che la presenza di tale compilatore influenzava le prestazioni, limitando in particolar modo le prime iterazioni.

## Analisi statistica dei dati

Per ognuna delle possibili combinazioni, si è scelto di prelevare un campione preliminare di 20 elementi.

Ogni campione è stato costruito eseguendo più volte il programma, per fare in modo da avere osservazioni che risultassero indipendenti fra di loro, e accodando i valori su un file di testo.

Lo scopo del campione preliminare è quello di utilizzarlo per effettuare un calcolo del numero di osservazioni richieste per avere un intervallo di confidenza del 90% (con un fattore di accuratezza r del 10%).

Essendo la grandezza del nostro campione minore di 30, dalla formula inversa dell’intervallo di confidenza per campioni piccoli, otteniamo questa formula per il calcolo di n:

Dove il valore è il quantile di una distribuzione t con N-1 gradi di libertà. Essendo il nostro α pari a 0.1, il corrispondente valore tabellato di è pari a 1.729.

Con s invece si indica la deviazione standard del campione e con la media campionaria.

Nelle seguenti tabelle tutti i valori di , s ed n (opportunamente calcolato) per ognuna delle combinazioni.

La prima tabella sono i valori nel caso di Debian fisico e la seconda nel caso di Debian virtuale.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | C | | | JAVA | | |
| N\_Sieve | **Media** | **Deviazione standard** | **n** | **Media** | **Deviazione standard** | **N** |
| 10.000 | 0,0332 | 0,001908 | 0.9877938 | 0,456146 | 0,001194 | 0.0020480 |
| 100.000 | 0,32275 | 0,013034 | 0.4875327 | 4,861413 | 0,007735 | 0.0007567 |
| 1.000.000 | 3,68115 | 0,078157 | 0.1347573 | 51,53435 | 0,270363 | 0.0082279 |

*Tabella 1: Calcolo grandezza campione per Debian reale*

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | C | | | JAVA | | |
| N\_Sieve | **Media** | **Deviazione standard** | **n** | **Media** | **Deviazione standard** | **N** |
| 10.000 | 0,03305 | 0,001572 | 0.676283 | 0,440491 | 0,015529 | 0.37155 |
| 100.000 | 0,31855 | 0,012521 | 0.46189 | 4,685377 | 0,048096 | 0.0315 |
| 1.000.000 | 3,6183 | 0,125755 | 0.361105 | 51,36604 | 2,190293 | 0.54355 |

*Tabella 2: Calcolo grandezza campione per Debian virtuale*

Come è possibile notare, valori di n sono tutti molto piccoli (approssimabili ovviamente per eccesso ad 1) per cui si può dire che basterebbe prendere campioni di una sola osservazione per avere intervalli di confidenza che nel 90% dei casi racchiudano la media della popolazione.

Poiché 1 è un valore molto basso e poiché non è molto oneroso da un punto di vista di costo, prenderemo campioni di grandezza pari a 5 per scrivere i nostri intervalli di confidenza.

Presi i campioni di grandezza 5, abbiamo valutato le medie e i coefficienti di variazione.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | C | | JAVA | |
| N\_Sieve | **Media** | **Coefficiente di variazione** | **Media** | **Coefficiente di variazione** |
| 10.000 | 0,0322 | 0,040492 | 0,456062 | 0,003777 |
| 100.000 | 0,3256 | 0,044166 | 4,86656 | 0,002637 |
| 1.000.000 | 3,6738 | 0,017045 | 51,47475 | 0,001436 |

*Tabella 3: Medie e coefficienti di variazione per campioni di grandezza 5, Debian reale*

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | C | | JAVA | |
| N\_Sieve | **Media** | **Coefficiente di variazione** | **Media** | **Coefficiente di variazione** |
| 10.000 | 0,032 | 0,022097 | 0,437551 | 0,02748 |
| 100.000 | 0,3212 | 0,039773 | 4,671606 | 0,007281 |
| 1.000.000 | 3,6938 | 0,02527 | 50,74513 | 0,003337 |

*Tabella 4: Medie e coefficienti di variazione per campioni di grandezza 5, Debian virtuale*

Dai valori dei coefficienti di variazione, notiamo che, essendo questi molto bassi, **la media è un ottimo candidato per la rappresentazione dei tempi di risposta dei sistemi.**

A questo punto siamo passati al calcolo degli intervalli di confidenza per le varie combinazioni. Abbiamo in particolare utilizzato la formula nel caso di campione piccolo, ricordando che dal teorema del limite centrale, se il campione è relativamente piccolo (minore di 30), la distribuzione campionaria ha in genere una forma di tipo t-distribution.

La formula in particolare è la seguente dove i valori che compaiono sono già stati spiegati precedentemente

Applicandola alle varie combinazioni (eseguendo la formula con un’ apposita function Matlab) otteniamo i seguenti intervalli di confidenza:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| N\_Sieve | DEBIAN | | DEBIAN VIRTUALE | |
|  | C++ | JAVA | C++ | JAVA |
| N=10.000 | [0.030956841056019 0.033443158943981] | [0.454419500541494 0.457704099458506] | [0.031325802402852 0.032674197597148] | [0.426087024893179 0.449015775106821] |
| N=100.000 | [0.311888736869274 0.339311263130726] | [4.854326151346897 4.878793848653102] | [0.309019579672277 0.333380420327723 | [4.639176003598704 4.704035996401296] |
| N=1.000.000 | [3.614094863556307 3.733505136443693] | [51.404273570759045 51.545223229240946] | [3.604802341958005 3.782797658041996] | [50.583683446357412 50.906568953642591] |

*Tabella 5: Intervalli di confidenza*

L’ultimo passo della nostra analisi statistica è stato quello di effettuare dei **Test sulla popolazione**, al fine di valutare se ci fossero campioni e quindi combinazioni statisticamente differenti per effettuare possibili confronti.

Tramite una semplice **analisi visiva**, abbiamo visto che per ogni N di Sieve e per ogni sistema operativo, le differenze tra i linguaggi di programmazione erano statisticamente significative. In particolare, possiamo notare che gli intervalli di confidenza sono **completamente disgiunti** (addirittura un altro ordine di grandezza) quindi ha senso effettuare un confronto per linguaggio di programmazione.

L’analisi più interessante che si può fare è valutare invece, se al variare del sistema testato (Debian reale o virtuale) si possa parlare di differenza significativamente statistica e quindi confrontare i valori.

Per valutare ciò, si è effettuato per ogni coppia del tipo “Debian\_Nx\_linguaggioy – VirtualDebian\_Nx\_linguaggioy” **un test zero mean sulle differenze per osservazioni accoppiate.**

Nello specifico, per ognuna di queste coppie, si sono fatte le differenze fra le osservazioni di posto analogo e si sono calcolati al solito modo, degli intervalli di confidenza per i campioni differenza.

Qualora un intervallo contenga lo 0, vuol dire che la differenza non è statisticamente significativa.

Di seguito tutti gli intervalli calcolati

|  |  |
| --- | --- |
| CONFRONTO | C.I. PER LA DIFFERENZA |
| Deb10k-cpp / Virtual10k-cpp | [-0.001366693639484; 0.001766693639484] |
| Deb100k-cpp / Virtual100k-cpp | [-0.017752290514527; 0.026552290514527] |
| Deb1M-cpp / Virtual1M-cpp | [-0.153063105732581; 0.113063105732581] |
| Deb10k-java / Virtual10k-java | [0.006029885318366; 0.030990914681634] |
| Deb100k-java / Virtual100k-java | [0.163801068108536; 0.226106931891464] |
| Deb1M-java / Virtual1M-java | [0.537700138872030; 0.921544261127970] |

*Tabella 6: test sulla differenza per sistema operativo*

Come si può notare, i primi 3 intervalli (quelli relativi al linguaggio C++) contengono lo 0, mentre quelli relativi a Java no. Questo ci porta a dire che possiamo confrontare sì i due sistemi, ma solo relativamente al caso Java, mentre relativamente a C++ non possiamo dire nulla.

## Confronto finale

Lo scopo finale del benchmarking di sistemi è quello di osservare nello svolgimento di compiti simili (in tal caso l’esecuzione dell’algoritmo Sieve), a patto che questi abbiano specifiche proprietà (come descritto nel primo paragrafo), le differenze notabili tra i due e cercare di stabilire un “vincitore” cioè chi è più performante in tal caso.

Nel nostro caso specifico il confronto riguarda due differenti esecuzioni di uno stesso sistema operativo su uno stesso elaboratore: una esecuzione è diretta, un’altra è emulata su un altro sistema operativo preesistente.

In base alle considerazioni effettuate nel paragrafo precedente, possiamo pervenire ai seguenti risultati:

* I campioni sono significativamente differenti al variare del linguaggio di programmazione. Pertanto, indipendentemente da se il sistema sia quello effettivo o se sia emulato, **il linguaggio C++ è molto più performante rispetto al linguaggio Java.**
* Non possiamo permetterci di valutare la differenza fra il sistema operativo reale e quello emulato tramite Hypervisor, nel caso in cui il linguaggio di programmazione sia C++. La differenza non è statisticamente significativa.
* Possiamo valutare la differenza tra sistema reale e sistema emulato qualora il linguaggio di programmazione sia Java. **Sorprendentemente, il sistema Debian emulato su Windows si comporta leggermente meglio per le esecuzioni in Java rispetto al caso in cui non sia emulato.**

# 2. Dataset Reduction

1. Descrizione del problema

Il problema che si pone in questo elaborato è quello dell’analisi multivariata di un set di dati.

Si ha a disposizione un dataset di 3000 dati (3000 righe), descritti tramite una serie di 24 attributi o features (colonne).

L’obbiettivo è quello di andare a ridurre considerevolmente questo dataset applicando note tecniche di data reduction al fine di ottenere un dataset più snello che sia comunque rappresentativo della popolazione da cui è stato prelevato il campione di partenza.

La **rappresentatività** non potrà mai essere la stessa ma applicando bene le tecniche e facendo le opportune considerazioni sarà possibile mantenerne un’alta percentuale.

In particolare ci concentriamo sull’idea di mantenere più invariata possibile la **variabilità** dei dati espressa in termini di varianza o devianza a seconda dei casi, che sono due indici di dispersione.

La varianza del dataset è espressa come dove è la media dei valori, mentre la devianza è semplicemente .

In altre parole andremo cioè a selezionare un numero n esiguo di righe dalle 3000 di partenza, in modo tale che le n righe selezionate mantengano quanto più possibile la variabilità dei 3000 dati di partenza.

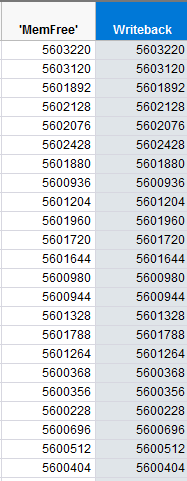
Le tecniche che utilizzeremo al fine di attuare il nostro scopo, saranno principalmente due: PCA e Clustering che andranno combinate e che analizzeremo dettagliatamente in seguito.

I software utilizzati per l’elaborato sono JMP, per l’applicazione delle tecniche sopracitate, e Matlab per le considerazioni numeriche sui risultati ottenuti.

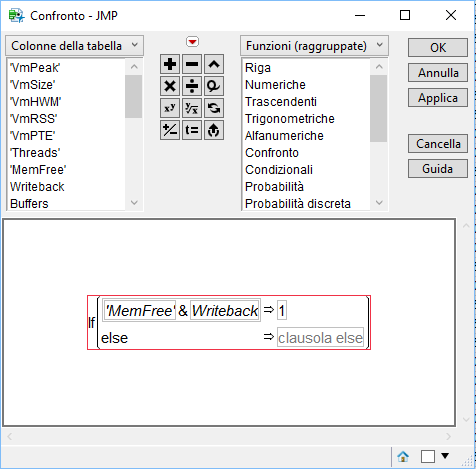
1. Trattamento preliminare dei dati

Prima di andare ad applicare le tecniche sopra accennate, si è andata a effettuare un’analisi qualitativa e statistica dei dati, mostrati in formato tabellare tramite il software JMP. Quest’analisi è stata molto utile e ha permesso l’eliminazione di alcune colonne dai nostri dati, senza alterarne la natura.

In particolare, come si può vedere dalla figura le due colonne **MemFree** e **Writeback** forniscono praticamente lo stesso valore per ogni riga di dato.



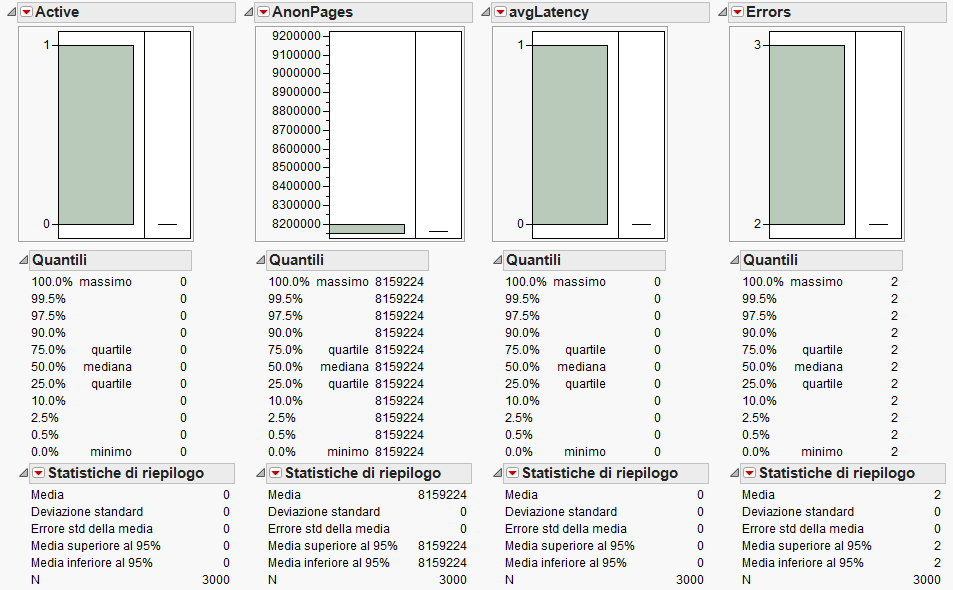
In particolare, per verificare se effettivamente tali colonne risultavano identiche abbiamo utilizzato la seguente formula condizionale in JMP.



*Figura 2 Confronto colonne*

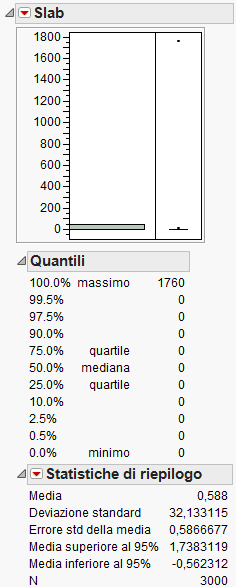
Il risultato è stato una colonna aventi di valori pari ad 1, per cui la nostra ipotesi è giusta e quindi è ininfluente in termini di variabilità dei dati mantenerle entrambe e si è optato per l’eliminazione della colonna **Writeback**.

Successivamente, da un’analisi degli istogrammi relativi ad ogni colonna, abbiamo visto che alcune colonne presentavano varianza nulla, in quanto i valori si ripetevano costantemente. In particolare le colonne **AvgLatency, Errors, Active e Anonpages** come si può vedere dalle immagini.



Anche queste sono state eliminate per il loro contributo nullo alla varabilità delle tuple.

Infine abbiamo notato la presenza di un ulteriore colonna che ha varianza diversa da zero ma che contribuisce davvero poco nella discriminazione dei dati e questa è la colonna ‘**Slab**’. L’eliminazione di tale colonna è dovuto dal fatto che presenta 2999 valori uguali e solo uno differente da tutti gli altri.



Al termine di questa prima analisi preliminare, siamo dunque rimasti con 18 colonne a fronte delle 24 di partenza, per cui sono stati eliminati 6\*3000= 18.000 valori numerici dai nostri dati senza che sia stata intaccata la loro variabilità complessiva.

1. PCA

La Principal Component Analysis è una tecnica per la semplificazione dei dati usata nell’ambito della statistica multivariata. Essa agisce sulle variabili dei dati (nel nostro caso le **colonne** della tabella). L’output di questa tecnica è una **trasformazione lineare** delle variabili. Questa trasformazione proietta le variabili in un nuovo sistema cartesiano in cui la prima variabile spiega la maggior parte della **varianza** dei dati di partenza, la seconda in minore quantità e così via.

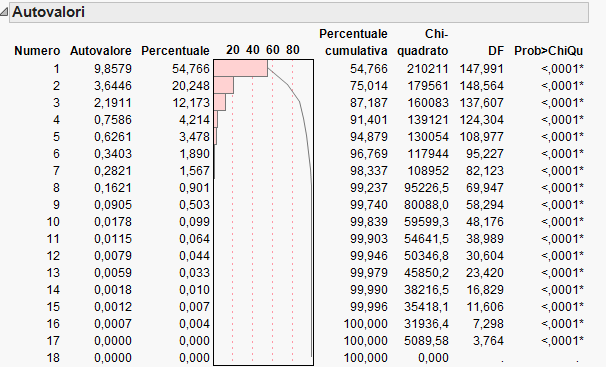
La trasformazione dello spazio apporta ovviamente anche una mutazione dei valori del dataset tramite la formula

Dove sono i valori espressi nello spazio delle componenti principali, sono i coefficienti di trasformazione dello spazio e i valori di partenza.

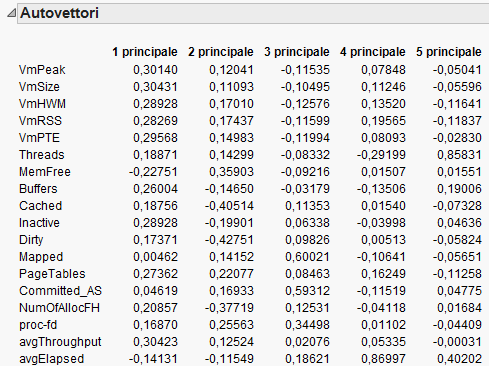
I coefficienti possono essere ottenuti utilizzando o la matrice di correlazione (quando i dati sono dipendenti linearmente) o la matrice di covarianza (per dipendenza quadratica) .

In definitiva passiamo da uno spazio di N variabili ad un altro di altrettante variabili, ordinate però in modo tale che le prime M con M<N spieghino un’alta percentuale della varianza dei dati di partenza. Applichiamo quindi tale tecnica al nostro caso in esame.

Il software JMP è in grado di effettuare l’analisi delle componenti principali senza particolari sforzi. Impostando gli opportuni settaggi (Matrice di trasformazione, Metodo di stima) abbiamo calcolato le nuove variabili e notato tramite un’ analisi degli autovalori che ne bastavano 5 (a fronte delle 18 di partenza) per spiegare il 94,879% della varianza.



Infine abbiamo voluto effettuare per completezza un’ulteriore analisi su tali componenti, andando a verificare quali degli attributi di partenza hanno maggior peso nelle componenti selezionate.



Di seguito vi è mostrato un riassunto dei pesi positivi e negativi maggiori:

|  |  |  |
| --- | --- | --- |
| COMPONENTI PRINCIPALI | PESI MAGGIORI NEGATIVI | PESI MAGGIORI POSITIVI |
| PRIMA COMPONENTE | 1. MemFree = -0.22751 2. AvgElapsed = -0.14141 | 1. VmSize = 0.30431 2. AvgThrougput = 0.30423 |
| SECONDA COMPONENTE | 1. Dirty = -0.42751 2. Cached = -0.40514 | 1. MemFree = 0.35903 2. Proc-fd = 0.25563 |
| TERZA  COMPONENTE | 1. VhHWM = -0.12576 2. VmPTE = -0.11994 | 1. Mapped = 0.60021 2. Commited\_AS = 0.59312 |
| QUARTA COMPONENTE | 1. Threads = -0.29199 2. Buffers = -0.13506 | 1. AvgElapsed = 0.86997 2. VmRSS = 0.19565 |
| QUINTA COMPONENTE | 1. VmRSS = -0.11837 2. VmHWM = -0.11641 | 1. Threads = 0.85831 2. AvgElapsed = 0.40202 |

1. Clustering

La tecnica della PCA sopra analizzata con l’eliminazione delle componenti meno significative comporta solo una riduzione del dataset in termini di colonne. La tecnica del Clustering invece, è specifica per la riduzione del dataset in termini di righe. L’idea di base di questa tecnica è quella, definita una metrica di distanza, di accorpare tra di loro punti (i dati) a distanza piccola in sottoinsiemi di punti denominati cluster.

Date N righe di partenza si generano un numero M di cluster con M idealmente molto minore di N. Ognuno degli M cluster verrà rappresentato tramite un solo punto, che può essere uno qualsiasi dei suoi punti interni oppure il centroide del clustering espresso tramite la media dei punti interni componente per componente.

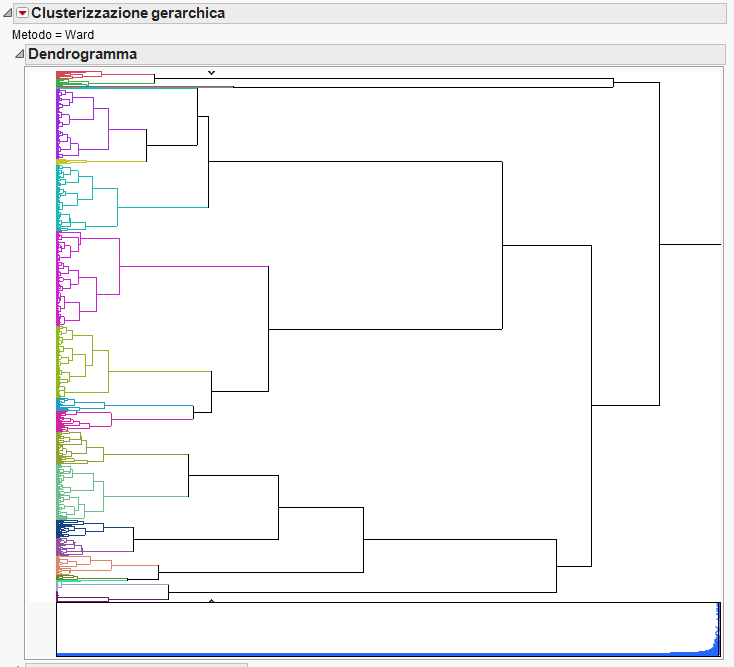
Il risultato finale è quindi quello di ridurre il dataset da N righe a M righe di dati.

Esistono varie tecniche di clustering. Principalmente queste possono essere distinte in tecniche gerarchiche, che tentano di accorpare nodi costruendo degli alberi, in cui la radice è un solo cluster e i nodi sono singoli punti e tecniche non gerarchiche (come k-means) che iniziano con una scelta arbitraria di k cluster e iniziano a muovere i nodi verso questi k.

Si è scelto di utilizzare, tramite il supporto di JMP , **la tecnica gerarchica di Ward**, che è quella che in genere si comporta meglio. Essa è una tecnica agglomerativa che unisce ad ogni iterazione una coppia di cluster tale da minimizzare l’incremento della devianza intracluster.

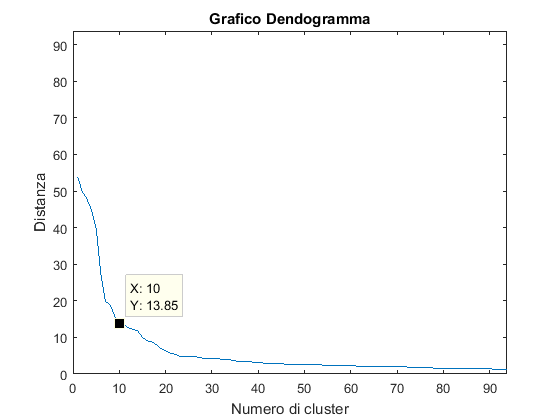
Abbiamo applicato il clustering sulle 5 componenti generate a valle dell’operazione di PCA descritta nel paragrafo precedente.

L’output offerto da JMP è stato il seguente dendogramma che è appunto l’albero che viene fuori dall’operazione di clustering gerarchico applicata.



Il dendogramma può fornire una mano su un passo molto importante che è quello della scelta del numero di cluster. Su JMP è possibile spostare la linea verticale sul dendogramma per vedere come vengono raggruppati i valori al variare del numero di cluster.

Per valutare un numero di cluster appropriato si è operato su due fronti. Abbiamo preso i valori della cronologia di clusterizzazione, che per ogni numero di cluster danno il valore corrispondente di distanza e li abbiamo riportati su un grafico.



Come si può vedere, la variabilità della distanza diminuisce per i valori di numero di cluster successivi al ginocchio del grafico. Questo implica che un valore efficiente di N in termini di costo potesse essere un valore intorno a 10.

Una volta scelto il numero di cluster, i valori vengono raggruppati tornando nel loro spazio originale (che è quello che ha senso fisico) e non considerandoli nello spazio generato dall’operazione di PCA.

Viene generata una colonna che indica per ogni tupla il cluster d’appartenenza e si possono ordinare i valori in base all’identificativo dei cluster a cui appartengono. Per ogni cluster, basta scegliere o il centroide dei suoi valori raggruppati oppure uno qualsiasi di essi (la variabilità persa è sempre la stessa). Ad esempio la nostra scelta è stata quella di prendere un valore casuale. A questo punto il processo si può dire concluso perché abbiamo estratto da N tuple di dati M tuple con M<N e con M pari al numero di cluster scelto.

A valle dell’operazione di clustering, in genere si esegue anche un trattamento specifico per gli **outliers**, cioè quei cluster ottenuti che hanno molti pochi dati rispetto agli altri. In molte applicazioni questi outlier possono essere eliminati, in altre no. Questo dipende dalla significatività di quei dati. Può essere infatti che essi rappresentino un comportamento specifico del sistema di cui è importante tener traccia. Ad esempio, i cluster 2 e 3 erano outliers in quanto presentavano pochissimi dati (dell’ordine delle unità). Osservando tali valori si è visto che avessero valori molto differenti dagli altri in alcuni campi (ad esempio i camp AvgElapsed e PageTables). Tuttavia per mancanza di informazioni sull’importanza di tale caso (cioè quando il tempo di risposta e il numero di page tables sono elevati) abbiamo deciso di non eliminare questi cluster

Descritta la tecnica del clustering nel suo complesso, possiamo tornare indietro su un passo molto importante che è quello della scelta del numero di cluster.

A conferma del discorso precedente sulla distanza, abbiamo ricercato un indice più specifico che ci indicasse la bontà di scegliere un certo numero di cluster rispetto ad un altro.

Per il clustering vale la seguente relazione:

Dove la devianza è quell’indice di dispersione descritto nel primo paragrafo (noi la considereremo come somma delle devianze dei valori colonna per colonna).

Abbiamo quindi deciso di utilizzare, come discriminante della bontà della scelta del numero di cluster, il fattore .

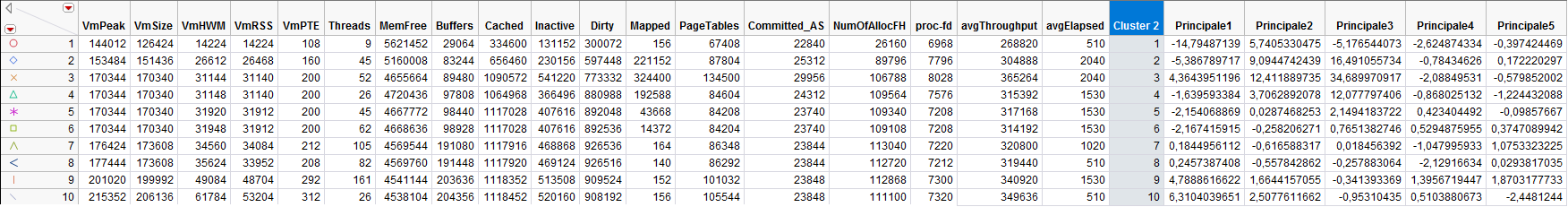
Questo fattore indica la percentuale di devianza persa a seguito dell’operazione di clustering. Più è alto dunque, più devianza si perde.

Ovviamente questo fattore diminuisce all’aumentare del numero di cluster. Abbiamo realizzato un script Matlab che calcolasse questo valore e ripetendo varie volte l’operazione di clustering abbiamo ottenuto i seguenti risultati.

|  |  |
| --- | --- |
| NUMERO DI CLUSTER | PERCENTUALE DI DEVIANZA PERSA |
| 3 | 0.072051537881512 |
| 5 | **0.034481005990008** |
| 6 | 0.034363445803687 |
| 7 | **0.032104789356779** |
| 10 | **0.025901347668673** |
| 15 | 0.**020382263047450** |
| 30 | **0.016815619892323** |

Ricollegandoci al discorso precedente, si può vedere che con il valore di 10 cluster precedentemente scelto si perde solamente il 2% circa di devianza e quindi si optato defitivamente per tale valore.

In ultima istanza, riportiamo i valori scelti dal dataset, predendone uno casuale da ognuno dei 10 cluster. Con tali valori rappresentiamo dunque il nostro dataset.



# 3. Web Server

## Descrizione del problema

Nel seguente capitolo si pone in esame il problema di valutare le prestazioni di un Web Server che viene sottoposto ad un carico stressante di richieste. In particolare la valutazione avviene tramite quattro passaggi chiave:

* **Workload characterization:** si genera un WL tramite richieste di JMeter, lo si caratterizza statisticamente e si sviluppa a partire da esso un modello riutilizzabile e ripetibile.
* **Capacity test:** analisi dei tempi di risposta e del throughput del sistema sottoposto a carico crescente per valutare le prestazioni limite.
* **Design of Experiments:** Progettazione di un case di esperimenti e valutazione dell’influenza dei vari fattori controllabili sulle variabili di risposta del sistema.
* **Aging:** Valutazione delle prestazioni del sistema sul lungo periodo e analisi dei tempi di decadimento.

Come Web Server si è analizzato il Web Server Apache v. 2.4.25, installato su macchina Samsung NP-RV520 con processore Intel Core i3-2310M da 2.10 GHz e 4Gb di Memoria RAM ospitante il sistema operativo Debian Jessie 8.2 a 64 bit. Le richieste sono state generate grazie all’applicativo Apache JMeter installato su macchina HP Pavilion g6 con processore Intel Pentium CPU B960 da 2.2 GHz e 4 GB di Memoria RAM ospitante il sistema operativo Microsoft Windows 10 a 64 bit. La connessione tra i due sistemi avveniva tramite tecnologia Ethernet.

## Workload Characterization

L’obbiettivo della Workload Characterization è quello di caratterizzare il sistema in esame, tramite opportune tecniche di analisi statistica e generarne un modello ripetibile. Relativamente al nostro Web Server è stata fatta una distinzione tra due tipologie di parametri prelevati, e in particolare:

* Parametri App-Level: sono tutti i parametri sensibili per l’utente client come il tempo di riposta, la latenza, il tasso di successo ecc. collezionabili tramite il tool JMeter e gli opportuni settaggi.
* Parametri Low\_Level: sono i parametri di basso livello che caratterizzano il Server; collezionati tramite il comando *vmstat* eseguito da terminale Linux, possono essere ad esempio la memoria libera, la cache, le interrupt per secondo ecc.

Per la WL Characterization è stato effettuato uno stress test della durata di 5 minuti. Tramite JMeter sono stati settati tutti **i parametri controllabili** di questo test. In particolare tramite il *Thread Group configurations* è stato imposto un **numero di thread** pari a 75. I thread rappresentano i diversi client che si ripartiscono le richieste da effettuare al server. Il numero scelto non è casuale; abbiamo cercato un numero di thread quanto più alto possibile (a fronte delle pagine scelte di cui parleremo di seguito) per rendere più veritiera la simulazione e rendere sensibili le richieste (essendo la connessione Ethernet molto veloce); un numero più alto tuttavia provocava un crash software dell’applicazione Jmeter che non riusciva bene a ripartire le richieste ai vari threads.

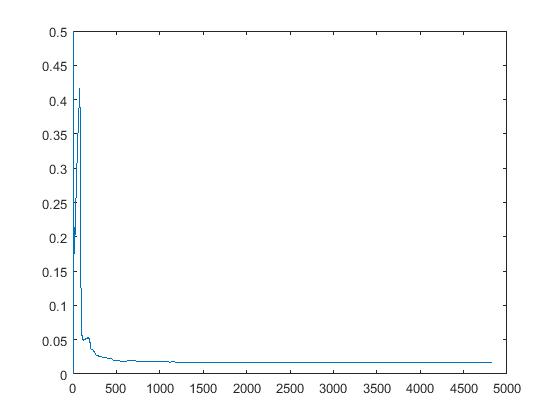
Per quanto riguarda la **tipologia di richieste**, abbiamo impostato 5 richieste http tramite gli opportuni Sampler *http request*. Ognuna di queste richieste si differenzia dalle altre per dimensione del payload ma anche per tipologia dei dati trasmessi (testo e/o immagine). Nella tabella seguente le 5 pagine utilizzate per la WL Characterization.

|  |  |  |
| --- | --- | --- |
| PAGINA | DIMENSIONE | TIPO DI DATO PREVALENTE |
| Manuale.html | 2.5 Kb | Testo |
| Tom&Jerry.jpg | 181 Kb | Immagine |
| YouTube.html | 530 Kb | Testo |
| Facebook.html | 934 Kb | Testo |
| Hoho.png | 2.1 Mb | Immagine |

In ultima istanza abbiamo imposto tramite il *Constant Throughput Timer* il carico di richieste da imporre specificato in richieste al minuto. Abbiamo scelto di selezionare un numero elevato (60.000) per spingere il test al massimo, anche se alla fine con le pagine selezionate non venivano generate più di 1000 richieste al minuto.

## 2.1 WL Parametri App\_level

I parametri App Level sono stati prelevati tramite il listener di JMeter *Simple Data Writer*. Nei 5 minuti di test sono stati prelevati 4821 valori. Come prima analisi abbiamo eliminato le prime 273 istanze e le ultime 118. Le prime perché dal grafico del throghput realizzato tramite ambiente Matlab (come visibile dall’immagine), c’è un picco elevato iniziale a nostro parere poco caratterizzante del dataset e dovuto ad una fase di assest di Jmeter.



Le ultime sono state eliminate essendo quelle relative alla fase di OFF di Jmeter (il pulsante è stato premuto manualmente allo scatto dei 5 minuti) in cui molte richieste sono fallite perché ancora in esecuzione. Sono rimaste infine un totale di 4430 tuple.

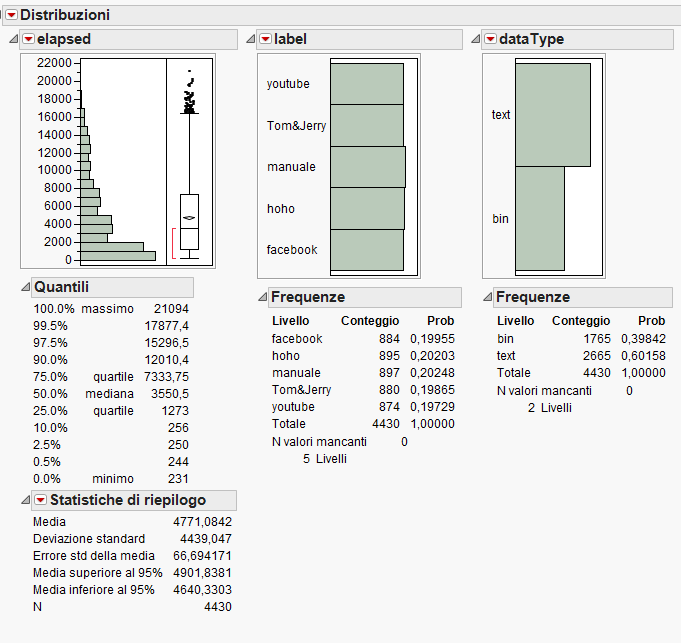
Tramite il tool JMP, è stata fatta un’ analisi visiva delle distribuzioni (istogrammi) di tutti i parametri App\_Level prelevati.

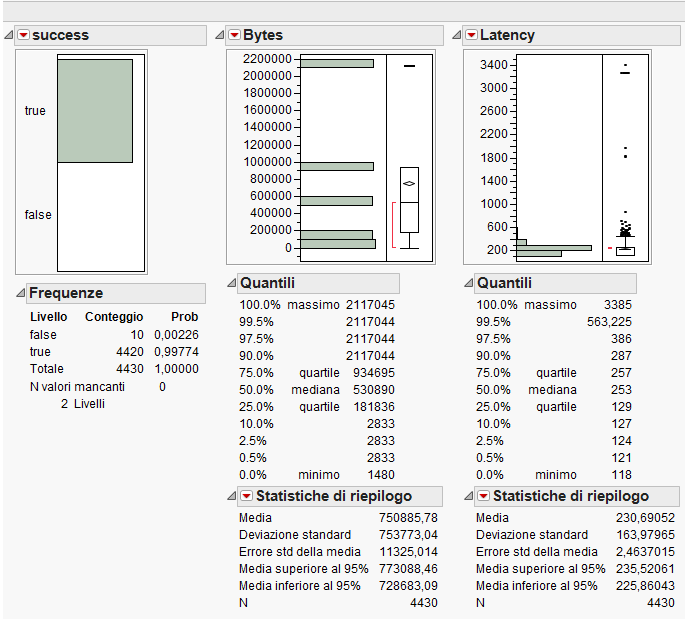
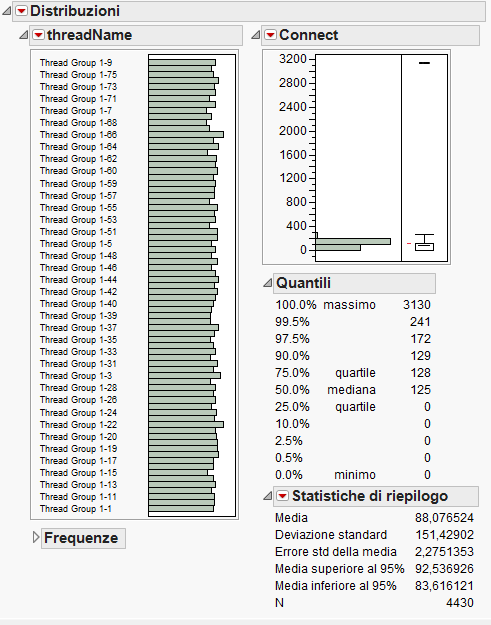
Da qua si è deciso quali colonne lasciare e quali no in base a se avessero potuto aver senso o meno nell’analisi statistica.

Di seguito un elenco di tutte le colonne mantenute con una breve descrizione del loro significato.

|  |  |
| --- | --- |
| PARAMETRO | SIGNIFICATO |
| Elapsed time | Tempo di risposta relativo alla richiesta |
| Label | Etichetta indicatrice della pagina richiesta |
| Thread name | Quale thread ha effettuato quella richiesta |
| Data Type | Tipo di dato (testo o bin) |
| Successo | Esito della richiesta (true o false) |
| Latency | Tempo che intercorre tra l’invio della richiesta e la sua ricezione |
| Bytes | Bytes inviati per quella richiesta |
| Connect | Tempo necessario per instaurare la connessione per quella richiesta |

Per questi fattori mostriamo nella figura di seguito l’andamento delle distribuzioni, risultato dall’analisi con JMP.

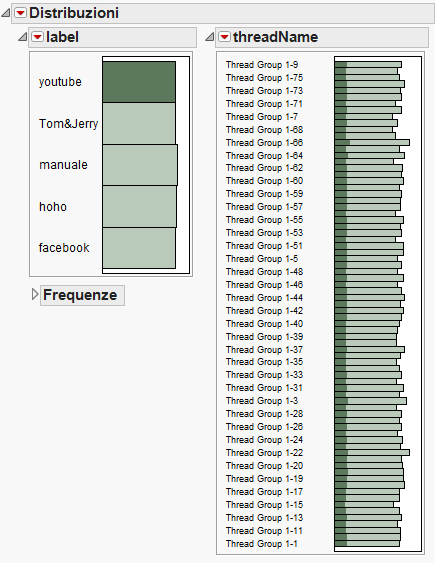




I tre parametri che ha senso caratterizzare statisticamente sono Elapsed Time, Latency e Connect. Come si può vedere dall’immagine, con i valori ottenuti possiamo dire che, analizzando il coefficiente di variazione (varianza su media), questo risulta essere molto basso, per cui la media indicata in figura risulta essere un buon candidato per rappresentare i tempi di Elapsed, latency e connect. Inoltre per questi tre fattori ci interessa più che altro la totalità di interesse, non tanto i singoli valori, per cui la media è un valido estimatore.

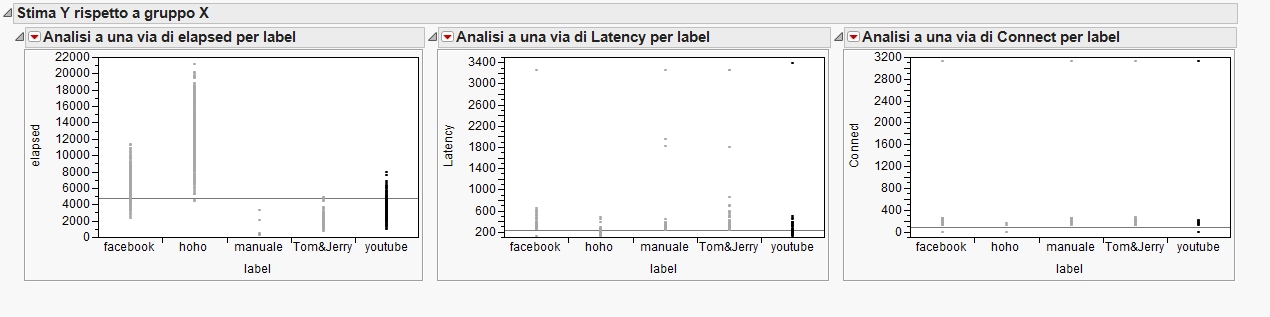
Un’altra cosa che si può analizzare dall’immagine è il parametro Success, che è quasi sempre “true”; solamente 10 valori su 4430 sono “false” il che vuol dire che la connessione è molto affidabile e raramente una richiesta termina con un fallimento. Questo ovviamente perché abbiamo utilizzato una connessione Ethernet che dal punto di vista fisico è molto affidabile; inoltre le richieste seguono il protocollo applicativo http che è affidabile e disponibile in quando si basa sul protocollo di trasporto TCP. Con la velocità a disposizione difficilmente possono infatti verificarsi problemi di timeout.

Per quanto riguarda Thread name, possiamo vedere che le pagine sono distribuite equamente tra i thread( cosa forzata da JMeter) , come si può vedere anche dal grafico seguente



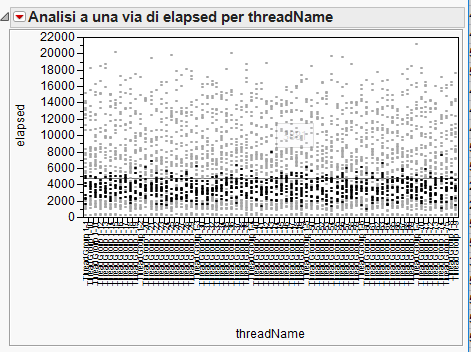
In ultima istanza, abbiamo deciso di effettuare un’analisi a coppie, per vedere un po’ come variavano i parametri (in particolare elapsed, latency e connect) in funzione degli altri fattori.

Questi primi 3 grafici mostrano elapsed, latency e connect in funzione del tipo di pagina.

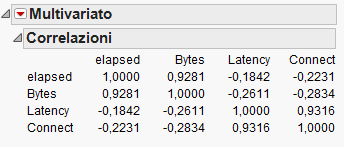


Da qui si vede come, giustamente, elapsed sia maggiore per pagine grandi, mentre latency e connect non dipendono dalla pagina ed è abbastanza casuale (ad esempio la pagina hoho ha mediamentente latenza più bassa della pagina facebook e tom&jerry nonostante sia molto più grande in termini di dimensioni).

Quest’altro grafico mostra invece elapsed in funzione dei thread. Da qui vediamo che più o meno tutti i thread mantengono gli stessi valori di elapsed time.



Infine si è voluto valutare se ci fosse una certa correlazione tra bytes, elapsed, connect e latency. Sfruttando l’analisi multivariata di JMP si è generata la seguente matrice di correlazioni:



Si può notare come bytes ed elapsed siano abbastanza correlati (questo perché più grande è la pagina, più alto sarà il tempo di risposta) ed anche latency con connect (il tempo di latenza dipende fortemente dal tempo di connessione).

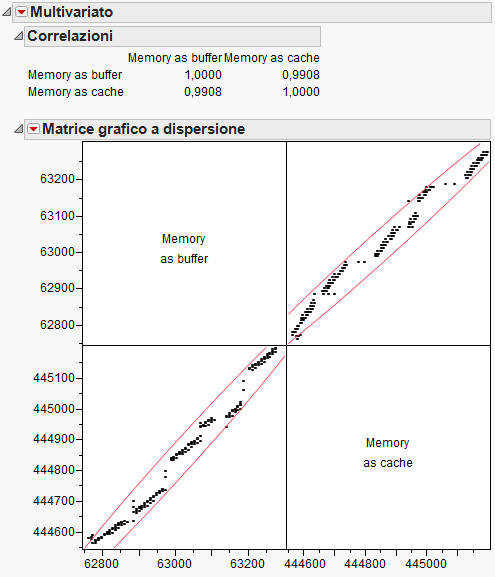
## WL Parametri Low\_level

Passiamo ora all’analisi Low level, e cioè a quella dei parametri di basso livello prelevati sul server tramite l’utility *vmstat*. Questi sono stati prelevati mentre veniva eseguito il test di 5 minuti sopra accennato. Il prelievo è stato impostato in modo tale da prelevare un campione di osservazione della frequenza di una ogni secondo. Alla fine ci siamo trovati 306 valori di cui alcuni sono stati scartati. In particolare abbiamo eliminato i primi 10+1 valori: il primo perché perdiamo un secondo avviando vmstat prima di JMeter e altri 10 perché abbiamo scartato i primi 273 valori dai parametri AppLevel che in termini di secondi corrispondono proprio a 10. Abbiamo poi eliminato anche gli ultimi 8 perché sono fuori dall’intervallo di osservazione (si può notare dal numero di interrupt per secondo che scende drasticamente da 9500 a 110).

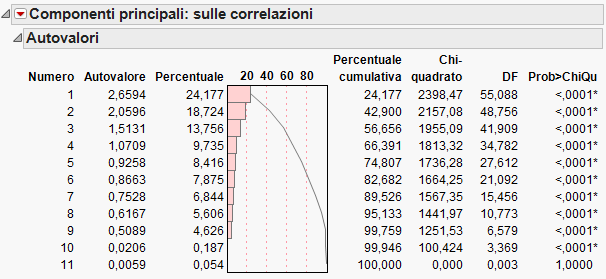
Come per l’analisi AppLevel , abbiamo dato un’occhiata alle distribuzioni per capire quali parametri considerare e quali no. Eliminando tutti quelli che ci sembrava inadeguato considerare (ad esempio memory swapped che era sempre 0) sono rimaste le seguenti colonne descritte in tabella

|  |  |
| --- | --- |
| PARAMETRO | SIGNIFICATO |
| Runnable process | Il numero di processi in attesa per la fase di esecuzione. |
| Process in sleep | Il numero di processi in stato di sleep |
| Free Memory | Memoria RAM libera |
| Memory as cache | Memoria RAM utilizzata come cache |
| Memory as buffer | Memoria RAM utilizzata come buffer |
| Interrupt per second | Numero di interrupt per secondi |
| Context switch per second | Numero di context switch per secondi |
| Time running no-kernel code | il tempo trascorso per l'esecuzione di codice non-kernel |
| Time running kernel code | il tempo trascorso per l'esecuzione di codice kernel |
| Time idle | Tempo trascorso per inattività. |
| Time wait IO | Il tempo trascorso in attesa di IO |

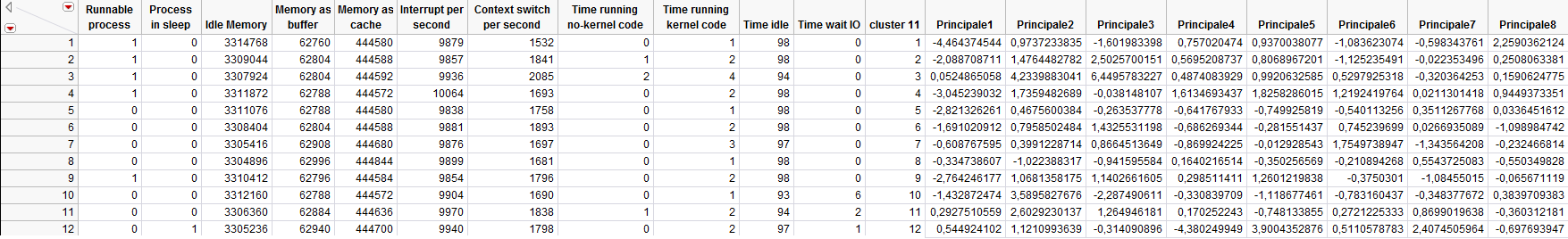
Per analizzare questa distribuzione, abbiamo deciso di agire in maniera differente effettuando tecniche di analisi multivariata come PCA e Clustering. In particolare abbiamo eseguito una PCA in quanto i valori che abbiamo raccolto hanno un alto grado di correlazione fra di essi, come possiamo notare ad esempio dalla matrice di correlazione tra memory as cache e memory as buffer.



Dopo aver effettuato la tecnica della PCA, abbiamo analizzato gli autovalori e quindi in particolare, la varianza spiegata da ogni componente, e scelto quanti fra questi prendere. Abbiamo preso le prime otto, poiché spiegano il 95,133 % della varianza totale.



Fatto ciò abbiamo ricorso alla tecnica del clustering, per effettuare una riduzione delle righe del dataset applicandola sulle componenti principali generate precedentemente. Abbiamo notato che un numero appropriato di cluster fosse pari a 12, data la forma del dendogramma. In particolare, con tale numero di cluster si è persa una percentuale di devianza pari a 30% totale. Di seguito i valori selezionati per ogni cluster.



## Capacity Test

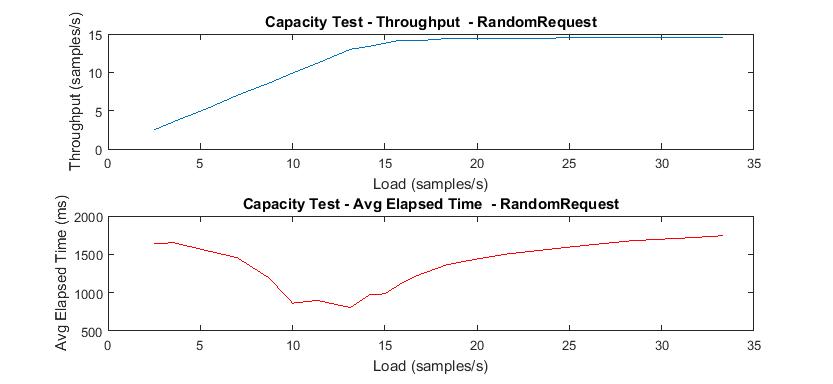
Il capacity test è un test che viene fatto per valutare i parametri di limite del sistema, cioè quei parametri oltre i quali il sistema non è in grado di andare. È molto utile per eseguire tuning di performance.

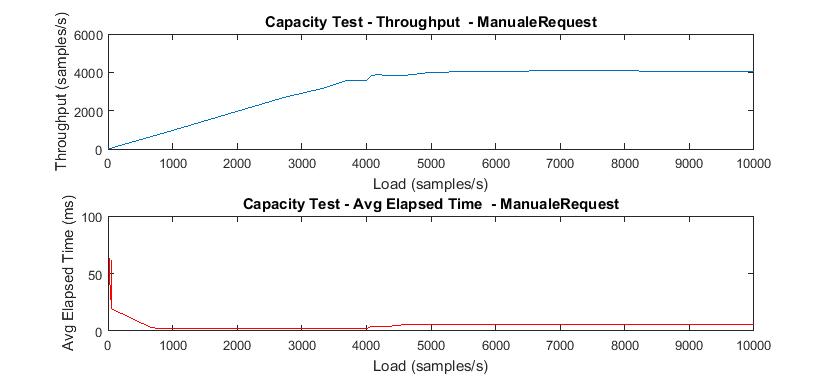
Per questo test, abbiamo testato 4 tipi di pagine di grandezza variabile (manuale, tom&jerry, facebook e hoho, già presentate nella WL characterization).

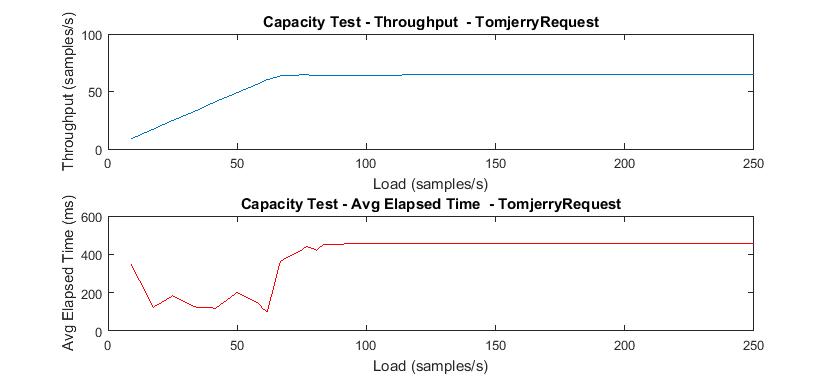
Il primo test è stato fatto considerando tutte le pagine insieme con le richieste presentate in maniera random tramite il *Random Controller* di JMeter.

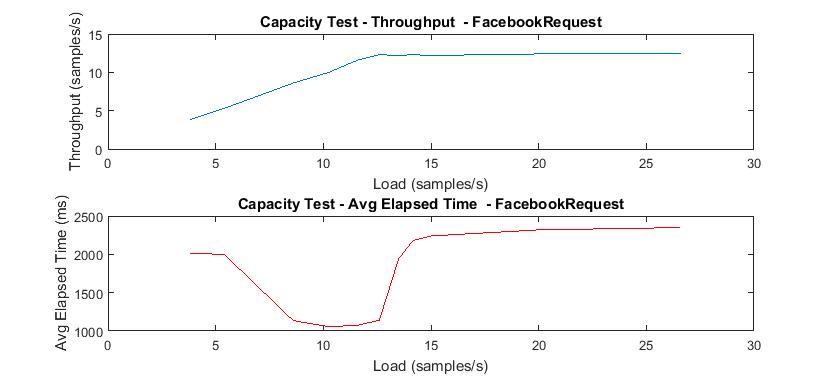
Poi abbiamo considerato le pagine singolarmente effettuando un capacity test opportuno per ogni pagina. In ognuno dei casi sono stati effettuati vari test (in numero variabile) con JMeter della durata di 1 minuto ciascuno e impostati con **carico crescente** tramite il *Constant throughput Timer*. Per ogni esecuzione sono stati manualmente prelevati i valori del throughput medio e dell’elapsed time medio forniti dal *Summary Report* di JMeter.

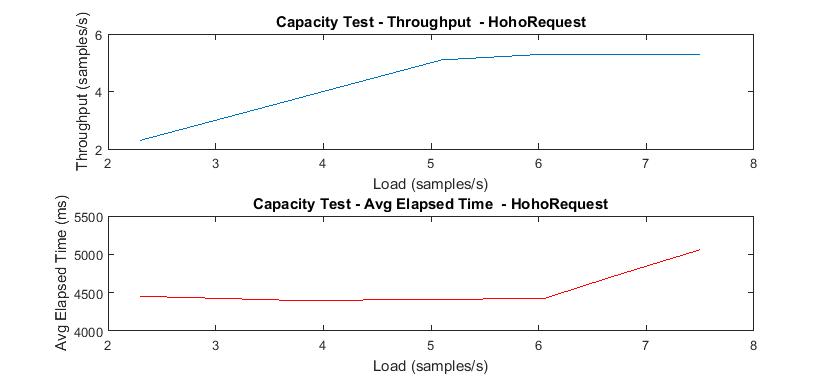
Tutto questo al fine di valutare per ogni caso (random e singole pagine) i grafici del throughput e dell’elapsed time in funzione del carico con l’obiettivo di osservare da questi grafici i valori di **ginocchio**, cioè il punto in cui si ha una drastica diminuzione dell’aumento di throughput (aumenta ma lo fa meno rapidamente) e di **usable capacity** (Capacità massimo di utilizzo, oltre la quale il throughput decresce o al massimo rimane costante)**.** Aver effettuato prima il caso random, è stato utile per velocizzare il prelievo dei valori per le singole pagine, perché tramite la grandezza di queste potevamo già predirre dove si potessero trovare i valori di carico corrispondenti a Knee e usable capacity, effettuando delle semplici proporzioni fra tali valori e la grandezza delle pagine.











Nella tabella riassuntiva qui riportata, possiamo infine osservare i valori di Knee e usable capacity trovati per ognuno dei casi testati

|  |  |  |
| --- | --- | --- |
| TEST | KNEE | USABLE CAPACITY |
| Random | 13 samples/sec | 14,5 samples/sec |
| Manuale.html | 3552,2 samples/sec | 4062,3 samples/sec |
| Tom&Jerry.jpg | 63,5 samples/sec | 64,9 samples/sec |
| Facebook.html | 12,3 samples/sec | 12,4 samples/sec |
| Hoho.png | 5,1 samples/sec | 5,3 samples/sec |

## Design of Experiments

L’obiettivo del design of experiment è quello di progettare un set di esperimenti per valutare una variabile detta **variabile di risposta** facendo variare opportuni **fattori controllabili** manipolabili da colui che effettua gli esperimenti. Di seguito poi si possono applicare varie tecniche statistiche per valutare quale dei fattori ha un maggior impatto sulla variabile di risposta in termini di **importanza e significatività.**

Nel caso in esame, come variabile di risposta si è considerato **l’average elapsed time** del nostro Web Server.

Per quanto riguarda i fattori, ne sono stati scelti 2 e in particolare il **carico imposto e il tipo di pagina.**

Per il fattore tipo di pagina abbiamo considerato 4 livelli ciascuno relativo a una delle pagine testate singolarmente nel capacity test (manuale, tome&jerry, facebook e hoho).

Per il fattore carico abbiamo considerato 2 livelli: LOW (che indica il 25 % della usable capacity relativa alla pagina in esame, calcolata nel capacity test precedente) e HIGH che è invece il 100% di tale valore.

Tutto questo per un totale di 8 possibili combinazioni.

Tramite il software JMP abbiamo impostato il piano personalizzato per il DOE. Abbiamo inserito i due fattori specificando per entrambi il loro essere fattori categorici e specificando il numero di livelli. Abbiamo eliminato le interazioni tra i fattori e impostato a 10 il numero di ripetizioni per ogni combinazione con un totale di 80 esecuzioni ( **Two factor-Full Factorial Design con replicazione**).

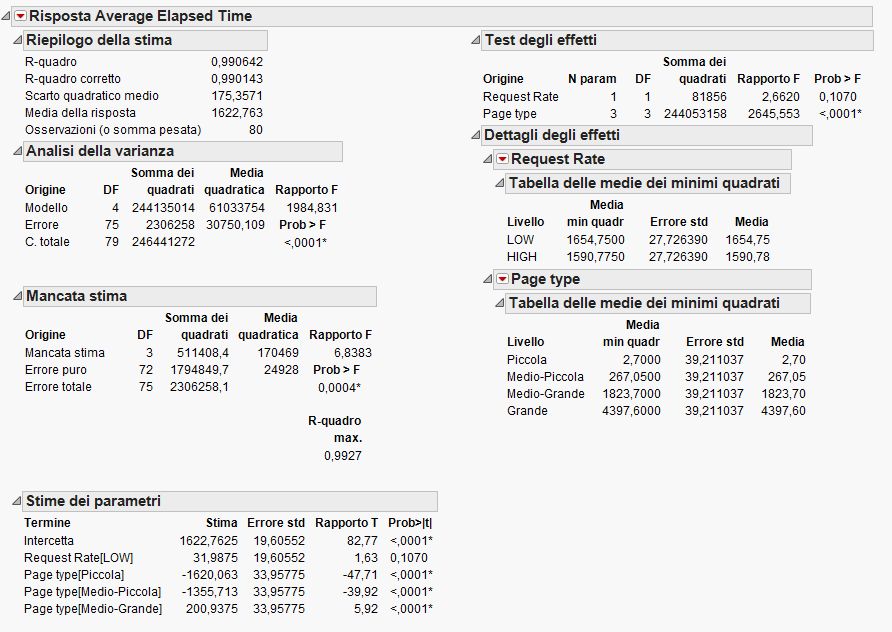
Una volta generata la tabella, l’abbiamo riempita con i valori ottenuti dai vari test. Ogni test (ogni ripetizione) ha avuto la durata di 1 minuto e si è collezionato per ognuna di esse il valore di average elapsed time fornito da JMeter.

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| CARICO = LOW | | | | | | | | | | |
| Manuale.html | 1 | 2 | 1 | 1 | 2 | 2 | 1 | 1 | 1 | 1 |
| Tom&Jerry.jpg | 298 | 149 | 192 | 181 | 288 | 307 | 214 | 260 | 248 | 243 |
| Facebook.html | 2155 | 1921 | 1834 | 2006 | 2020 | 1968 | 1951 | 2037 | 2017 | 2022 |
| Hoho.png | 4513 | 4027 | 4331 | 4565 | 4319 | 4665 | 4359 | 4249 | 4278 | 4560 |

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| CARICO = HIGH | | | | | | | | | | |
| Manuale.html | 4 | 4 | 4 | 4 | 4 | 4 | 4 | 4 | 5 | 4 |
| Tom&Jerry.jpg | 232 | 268 | 296 | 325 | 306 | 313 | 301 | 313 | 304 | 303 |
| Facebook.html | 1300 | 1115 | 1960 | 1747 | 1899 | 1528 | 1621 | 1949 | 1911 | 1513 |
| Hoho.png | 3723 | 4378 | 4637 | 4477 | 4513 | 4446 | 4666 | 4417 | 4439 | 4390 |

Come già detto precedentemente, l’obbiettivo del DOE è quello di valutare l’impatto dei fattori sulla variabile di risposta. In particolare possiamo valutare l’importanza e la significatività dei fattori.

L’**importanza** di un fattore indica la percentuale di variazione espressa sottoforma di sum of squares spiegata del quel fattore rispetto a quella totale. Indica cioè quanto contribuisce quel fattore a rendere tale la variabilità dei dati in nostro possesso. Tramite la funzione *Stima modello* di JMP è possibile ottenere facilmente la tabella delle Sum of squares e valutare facilmente questi rapporti.

.

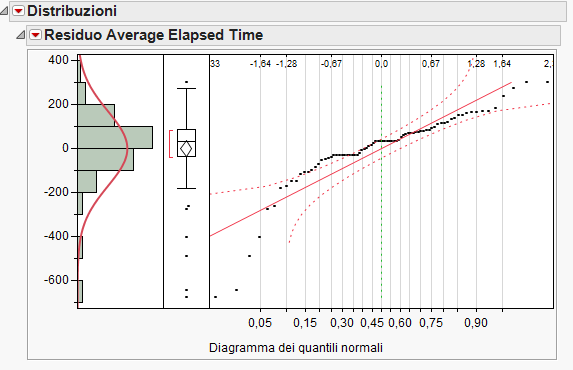
Come si può facilmente notare, la stragrande maggioranza della SS totale viene spiegata dal fattore “tipo di pagina” (precisamente il 0,9996%). Inoltre si può notare che la percentuale dell’errore per 80 esperimenti è pari a 0,00935 %.

La **significatività** di un fattore sulla variabile di risposta indica invece se la sua variazione spiegata è significativamente più alta di quella dell’errore. Importanza non implica necessariamente significatività. Valutare la significatività è importante per capire se i fattori sono più o meno statisticamente significativi sulla variabile di risposta senza che gli errori sperimentali possano influenzarla di molto.

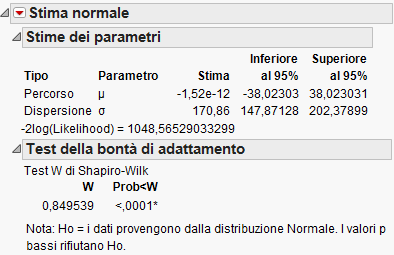
Per valutare questo parametro si esegue il test della varianza ANOVA.

Questo test deve essere eseguito in modi diversi, a seconda dei nostri dati. In particolare dobbiamo valutare la **normalità dei residui** e poi eventualmente l’**homoschedasticità dei dati.**

Quindi consideriamo i residui cioè le differenze tra ogni valore e il valore medio di tutti i valori (ovviamente un valore medio per ogni specifica combinazione). Studiare la normalità vuol dire valutare se la distribuzione dei residui è assimilabile ad una normale. Per vedere questo si può effettuare dapprima un’analisi visiva tramite il plot quantile-quantile disegnato con JMP.

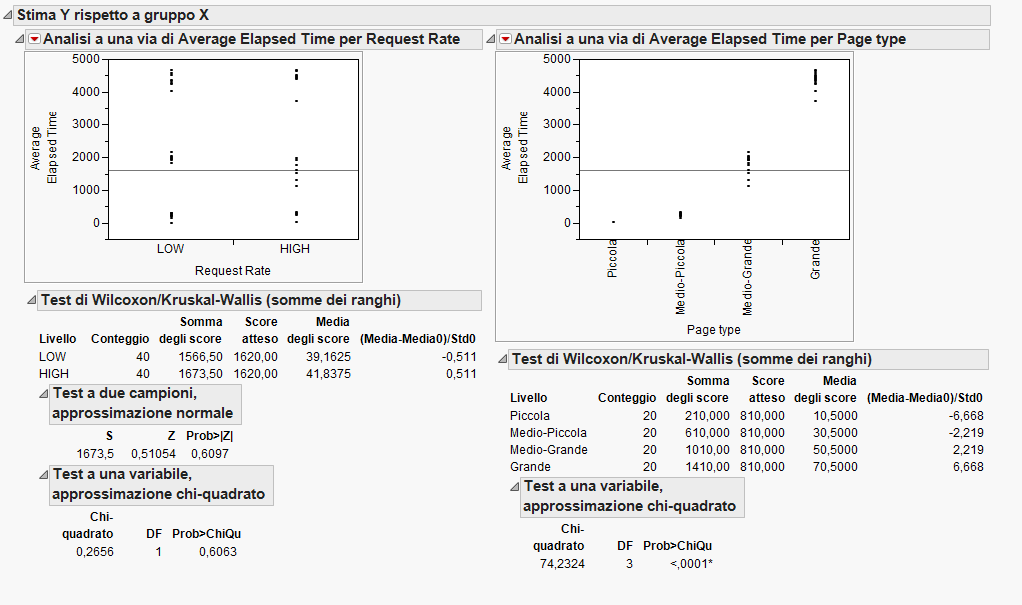


In genere possiamo dire che è normale quando la distribuzione rientra nella banda rossa del plot indicata. Nel nostro caso ci sono alcuni valori che vanno al di fuori di questa banda; potremmo ignorarli e considerare normale ugualmente la distribuzione. Per una maggiore confidenza eseguiamo però il test di Shapiro-Wilk con JMP.



Il test indica un valore di p<0.0001; questo ci conferma quindi la non normalità. Per questo la decisione finale è quella di procedere con un test **ANOVA non parametrico** senza studiare l’homoschedasticità, perché il risultato di non normalità dei residui basta a definire questa scelta. Se avessimo dovuto studiarla avremmo potuto facilmente valutarla con il test di Levene fornito da JMP.

Quindi si procede con il test ANOVA non parametrico di **Wilcoxon-Kruskall Willis,** sempre valutato con JMP



L’esito del test di Wilcoxon per entrambi i fattori indica che l’ipotesi che “un fattore x non influenzi l’esito dell’esperimento” è **rigettata** per il caso del fattore page-type, mentre risulta **non rigettata** per il fattore request rate. A conferma di ciò basti notare dalla precedente come il tempo di risposta medio cambia a seconda del tipo di pagina, mentre risulta lo stesso per entrambi i livelli di request rate.

## Performance Degradation Analysis

L’obiettivo della performance degradation analysis è quello di valutare il comportamento di un sistema sul lungo periodo, analizzando il degrado delle performance dal punto di vista di una o più variabili e ottendendo una stima dei tempi che può impiegare a raggiungere talune condizioni sensibili.

In particolare vogliamo effettuare un’analisi di questo tipo sempre relativamente al nostro Web Server.

Abbiamo impostato JMeter con la configurazione Random selezionata anche nel capacity test; abbiamo impostato il carico alla sua usable capacity e lasciato eseguire per 7 ore e 30 minuti.

I parametri che abbiamo collezionato e di cui si voleva effettuare la performance degradation analysis sono i parametri relativi alla memoria del server.

Con il comando Linux *vmstat 30 >valori\_aging.csv* abbiamo prelevato i valori di nostro interesse (quelli relativi alla **memoria**, quella **libera** in particolare e in aggiunta quella usata come cache e come buffer), con una frequenza di un’osservazione ogni 30 secondi; i valori della memoria swapped non sono stati considerati perché sempre nulli.

Al termine del prelievo dei valori abbiamo ottenuto un set di 903 valori.

Il primo passo è stato quello di effettuare un fitting dei dati per memoria libera, cache e buffer, realizzato in ambiente Matlab.







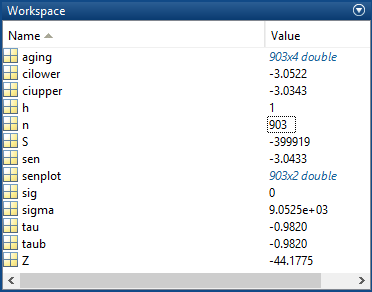
Quello che ci interessa in particolare in questa analisi è valutare la degradazione della memoria libera. Come si vede dal grafico infatti (memoria libera in funzione dei secondi trascorsi) i valori seguono un trend lineare di decadimento ben definito.

Per valutare effettivamente l’esistenza di questo trend, eseguiamo il test non parametrico di Mann-Kendall tramite il comando Matlab

***[taub tau h sig Z S sigma sen n senplot cilower ciupper] = ktaub( [x y] a,1)***

dove il primo parametro di ingresso indica i valori dell’asse delle x (30, 60, 90 sec …), il secondo il valore assunto dalla funzione in tali punti, il terzo il grado di confidenza e l’ultimo indica se plottare lo slope oppure no.

Otteniamo i seguenti valori in output riguardo l’andamento della memoria libera:



Di seguito il grafico fornito da tale funzione:



Dal grafico si può notare come lo slope (coefficiente angolare della retta pendente di colore rosso), segue perfettamente i dati.

A conferma di ciò analizziamo i valori forniti dal test. In particolare due sono i valori significativi: *h*, che ci indica l’esito del test; essendo pari a 1 vuol dire che il test ha avuto successo ed effettivamente **i dati seguono un trend** lineare. Il parametro *sen* invece ci indica il coefficiente di angolazione dello retta di regressione.

Questo parametro è molto importante, perché possiamo usarlo per valutare il TTE (time to exhaustion) della nostra memoria libera; vogliamo valutare cioè quanto è il tempo necessario (ammesso che questo trend venga sempre seguito) affinchè la memoria libera diventi 0.

Per fare ciò basta semplicemente scrivere l’equazione della retta, considerando il valore iniziale della retta e il coefficiente angolare fornito da Ktaub.

Per ricostruire la retta si è utilizzato il seguente script in matlab:

vv = median(aging(:,1));

middata = aging(round(length(aging(:,1))/2),4);

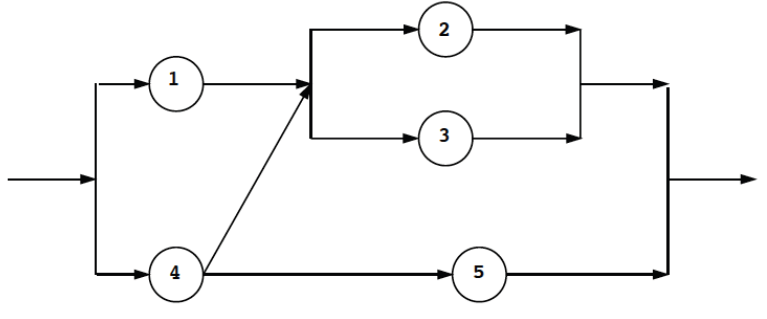
rect = vv + sen\*(aging(:,4)-middata);

Data quindi l’equazione della retta , abbiamo che, imponendo y=0, il TTE è pari a 1067608 secondi, cioè al’incirca 296.5 h.

# Esercizi Reliability

## Exercise 1

Calcolare la reliability R(t) e l’MTTF del sistema il cui diagramma di reliability è mostrato in figura. Nel calcolare l’MTTF assumere che tutti i componenti siano uguali e che falliscano randomicamente con failure rate pari a .



II sistema a blocchi di cui dobbiamo valutare la reliability si trova in una forma non series-parallel, cioè non è possibile calcolare la R(t) considerando i componenti come sole combinazioni di serie e paralleli. Per risolvere il problema dunque, conviene rifarsi alla definizione di reliability come probabilità che il sistema non fallisca in [0,t] e applicare opportune leggi probabilistiche. In particolare possiamo applicare la regola di Bayes.

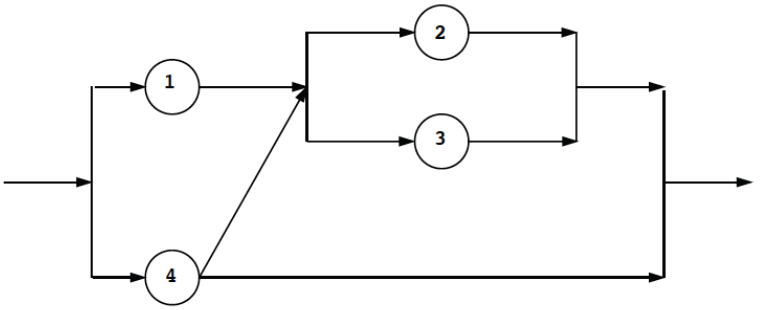
Grazie a questa regola, considerato uno qualsiasi dei blocchi, considerato che tutti i blocchi hanno la stessa reliability R e considerato che falliscano in maniera indipendente l’uno dall’ altro, si può scrivere la reliability del sistema tramite la seguente espressione:

Dove la prima probabilità è la probabilità che il sistema non fallisca nell’intervallo [0,t] considerato che il blocco scelto sia funzionante, mentre la seconda è la probabilità che il sistema non fallisca nell’intervallo [0,t] considerato che il blocco scelto fallisca in tale intervallo di tempo.

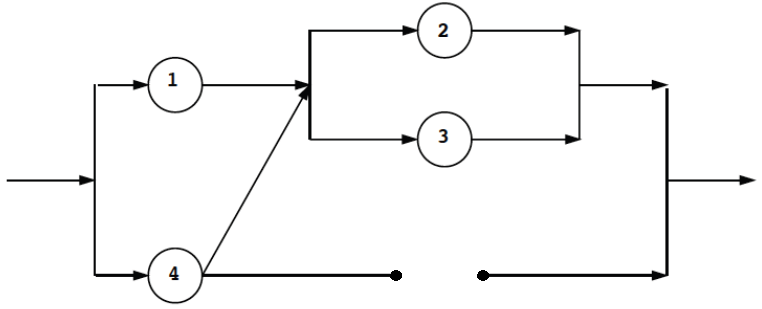
Abbiamo scelto come blocco su cui applicare tale relazione, il blocco 5 che ci è sembrato quello più indicato per semplificare la forma delle reti risultanti.

A questo punto rimane soltanto di valutare le due probabilità appena descritte che sono fondamentalmente la reliability del sistema considerato che il blocco 5 sia sempre funzionante e la reliability del sistema considerato che il blocco 5 fallisca sicuramente.

Per fare ciò conviene rappresentare il sistema nelle due configurazioni sostituendo il blocco 5 con un collegamento sicuro nel caso in cui si presuppone che funzioni sempre, e troncando il suo collegamento nel caso si presupponga il contrario.



*Figura 3 Sistema considerato che il blocco 5 funzioni*



*Figura 4 Sistema considerato che il blocco 5 non funzioni*

Calcoliamo dapprima la reliability del sistema nel caso il blocco 5 funzioni.

Come si può vedere dalla figura, possiamo vedere il sistema come in una configurazione che consiste del parallelo tra il blocco 4 e il blocco comprensivo della serie tra 1 e il parallelo 2//3. Questo in quanto il collegamento centrale diviene ininfluente perché il sistema funziona solo se funziona 4 oppure se funziona il blocco serie di 1 e 2//3.

Detto questo, il calcolo diventa semplice conoscendo le relazioni per cui la reliability di due blocchi in serie è il prodotto delle due reliability, mentre quella di due blocchi in parallelo è 1 meno il prodotto delle loro due un-reliability.

=

= =

= = =

= = .

Passiamo ora al calcolo della reliability del sistema nel caso si consideri il blocco 5 non funzionante. In tal caso, come notiamo sempre dalla figura, la configurazione è ancora più semplice e consiste semplicemente della serie tra il parallelo 1//4 e il parallelo 2//3. Pertanto si ha:

=

= .

A questo punto possiamo tornare alla relazione iniziale per esprimere la reliability totale del sistema:

=

=

.

Per valutare il MTTF del sistema, per definizione si effettua un integrale tra 0 e +∞ della R(t) del sistema.

Definito il tasso di fallimento λ del singolo blocco, si suppone implicitamente che l’ andamento della reliability del singolo blocco sia un esponenziale negativo e segua la legge .

La reliability del sistema complessivo in funzione del tempo sarà quindi:

.

Come detto il MTTF lo calcoliamo effettuando . Da cui:

=

=

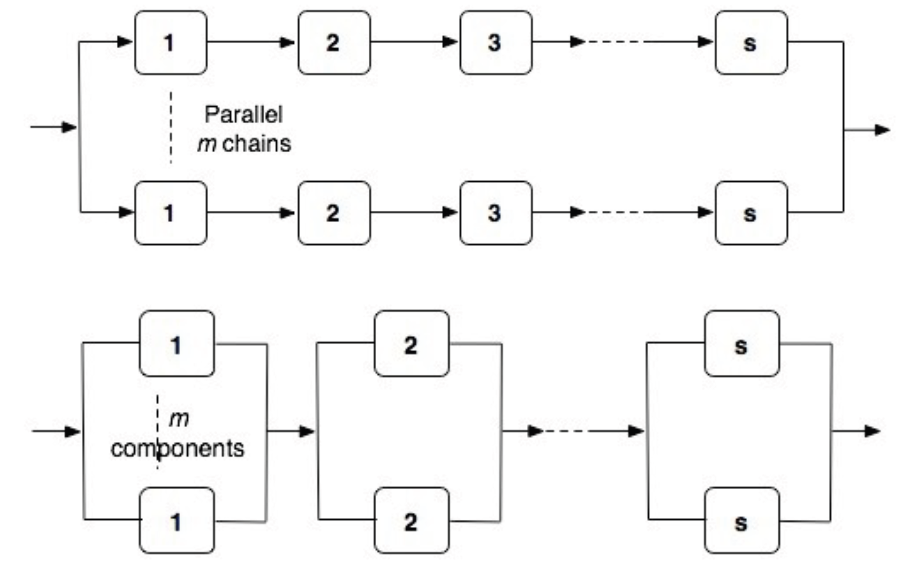
Risulta quindi che con la configurazione a blocchi indicata, il MTTF del sistema complessivo è uguale al MTTF di un singolo blocco.

Il seguente grafico Matlab, mostra come si comporta la reliability del sistema nel tempo a seconda di differenti valori di λ



## Exercise 2

Si vogliono mettere a confronto due schemi che cercano di aumentare la reliability di un sistema usando ridondanza. Supponiamo che il sistema abbia bisogno di s componenti identici in serie per le proprie operazioni. Supponiamo anche che siano dati (m \* s) componenti. Quale dei due schemi mostrati in figura fornirà una reliability maggiore? Considerando che la reliability di un singolo componente sia r, deriva le espressioni per le reliability delle due configurazioni. Confronta le due espressioni per m= 2 e s= 3.



Osservando gli schemi, si può subito visivamente notare che la seconda configurazione sia la migliore per qualunque m ed s. Avendo bisogno che funzionino s componenti consecutivi, con la prima configurazione avremmo che deve per forza funzionare una **completa** delle m serie a disposizione. Con la seconda configurazione invece, si hanno m paralleli e bisogna che per ogni parallelo funzioni **uno qualsiasi** degli s componenti che lo compongono. Per cui senza dubbio le probabilità che il sistema fallisca nel secondo caso sono minori sempre di quelle del primo caso.

Andiamo quindi ora a ricavare le espressioni per le reliability dei due sistemi. Dalle relazioni che regolano le reliability di sistemi con componenti in serie e in parallelo si hanno i seguenti risultati:

Essendo che le reliability dei singoli componenti sono tutte uguali e tutte pari ad , possiamo semplificare come segue:

L’esercizio richiede poi in particolare di confrontare le due espressioni per m=2 e s=3.

A tal fine abbiamo considerato che la reliability dei blocchi segua un andamento esponenziale negativo nel tempo con una legge .

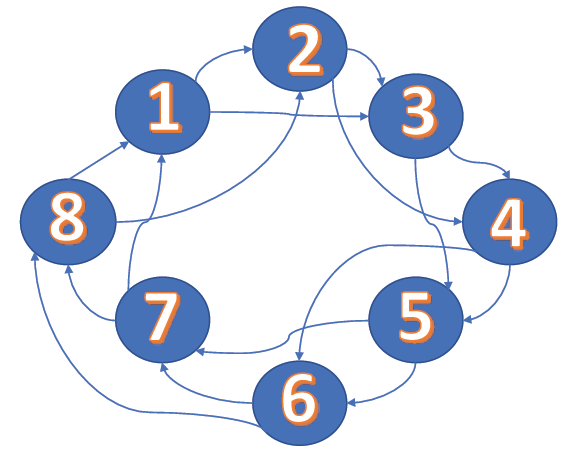
Riportando in ambiente Matlab i valori di ee fissando un certo λ per l’espressione di , abbiamo plottato, con i valori ottenuti da una function scritta ad-hoc, le due reliability in funzione del tempo, ottenendo i risultati mostrati in figura.



Come si può notare, la reliability del secondo sistema risultata sempre superiore al variare del tempo.

## Exercise 3

L’architettura di una rete di computer in un sistema bancario è mostrata in figura. L’architettura è chiamata rete a skip-ring ed è progettata per permettere ai processori di comunicare anche dopo che si sono presentati dei fallimenti di alcuni nodi. Ad esempio, se fallisce il nodo 1, il nodo 8 può bypassare il nodo fallito istradando i dati su un link alternativo che va dal nodo 8 al nodo 2. Assumendo che i collegamenti siano perfetti e che ogni nodo abbia una reliability , deriva un’espressione per la reliability della rete. Se obbedisce alla legge di fallimento esponenziale e il failure rate di ogni nodo è 0.0005 fallimenti all’ora, determina la reliability del sistema al termine di un periodo di 24 ore.



Come è possibile notare dalla descrizione e dall’immagine, in pratica il sistema fallisce in un intervallo di tempo [0,T], qualora ci siano almeno due nodi consecutivi che sono falliti in questo stesso arco di tempo.

In tale intervallo di tempo può fallire un numero di nodi che va da 0 ad 8 e per ognuno di questi casi c’è un certo numero di configurazioni nodi falliti-nodi non falliti da considerare.

L’obbiettivo è quello di valutare la reliability dell’intera rete, cioè la probabilità che il sistema funzioni in [0,T]. Per fare ciò, tenendo conto di quanto detto, applichiamo il teorema della probabilità totale per cui la probabilità che il sistema funzioni (A è l’evento sistema funzionante) è data dalla somma di tutte le .

è l’evento che indica che nodi su 8 non falliscono e quindi è la probabilità che il sistema funzioni dato che funzionano nodi su 8.

è l’evento che indica che il sistema non fallisce in [0,T]

è l’evento che indica che nodi su 8 non sono falliti.

Cominciamo con il valutare le varie . Essendo i nodi indipendenti, possiamo valutare la probabilità che si verifichi l’evento come la probabilità che nodi non falliscano per la probabilità che nodi falliscano; il tutto moltiplicato per il numero di configurazioni possibili in cui si verifica tale evento. Ad esempio la probabilità che funzionino 7 nodi su 8 è il prodotto tra che è la probabilità che funzionino 7 nodi, che è la probabilità che 1 nodo non funzioni e 8, che sono le possibili configurazioni della rete in cui 1 nodo non funziona e gli altri 7 sì.

Nello specifico, il fattore “numero di configurazioni con nodi funzionanti su è pari alla binomiale .

Per cui possiamo formalizzare le nel modo in cui segue:

.

Il prossimo step, è quello di valutare le , cioè le probabilità che il sistema funzioni dati nodi funzionanti su 8.

La generica è data dalla probabilità che dati nodi falliti, non ci siano coppie di nodi falliti consecutivi. Per valutarla, si sono analizzate per ogni , tutte le configurazioni con nodi falliti e si sono contate quante di queste **non** presentassero almeno due nodi falliti consecutivi.

La probabilità è il rapporto tra il numero di queste occorrenze fratto il numero di configurazioni possibili con nodi funzionanti (cioè la binomiale di cui sopra). Ad esempio la probabilità che il sistema funzioni dato che funzionino 4 nodi su 8 è perché con 4 nodi falliti ci sono solo due configurazioni in cui il sistema può funzionare è cioè quella con i nodi 1-3-5-7 che funzionano o quella con i nodi 2-4- 6-8 che funzionano, a fronte delle 70 combinazioni possibili con 4 nodi funzionanti e 4 non (.

Nella seguente tabella, tutte le , valutate contando manualmente i casi in cui ci fossero due nodi falliti consecutivi nelle configurazioni possibili (contando ovviamente come nodi consecutivi anche 8 e 1).

|  |  |
| --- | --- |
| NUMERO NODI FUNZIONANTI SU 8 |  |
| 0 | 0 |
| 1 | 0 |
| 2 | 0 |
| 3 | 0 |
| 4 |  |
| 5 |  |
| 6 |  |
| 7 | 1 |
| 8 | 1 |

Da qui possiamo finalmente esplicitare la forma finale della reliability del sistema:

Andando a sostituire i valori corretti si ha:

\*

.

L’ultimo punto richiesto dall’esercizio è quello di valutare, supponendo che segua una legge esponenziale e che il tasso di fallimento λ di un nodo è 0.0005/h la reliability del sistema dopo 24 ore.

Stando a quanto detto sopra, la legge che regola la reliability del singolo blocco è .

Avendo noi un’espressione per la reliability del sistema complessivo, e cioè , possiamo sostituire a la sua legge esponenziale in essa.

Riportando tutto in ambiente Matlab, tramite la seguente function, abbiamo valutato i valori della reliability al variare del tempo, secondo un intervallo prefissato. Ovviamente il tempo è inteso in ore.

function [ y] =es3( l,t )

y=0;

p=[0 0 0 0 2/70 16/56 20/28 1 1];

R=exp(-1\*l.\*t);

for i=0:8

y=y+p(i+1)\*nchoosek(8,i).\*R.^i.\*(1-R).^(8-i);

end

end

Tali valori sono stati poi appositamente plottati.



Cercando sul grafico il valore per t= 24, si ottiene che la reliability del sistema dopo 24 ore è pari a quasi 3 nines (0.9989).

## Exercise 4

Un’ applicazione richiede che almeno 3 processori di un sistema multiprocessore siano disponibili con più del 99% di probabilità. Il costo di un processore con l’80% di reliability è 1500$ e ogni incremento del 20% in reliability costa 1000$. Determinare il numero di processori n e la reliability p di ogni processore (assumere che tutti i processori abbiano la stessa reliability) che minimizzi il costo totale del sistema.

La configurazione espressa nel problema di cui sopra (cioè che almeno 3 processori su N debbano funzionare affinchè il sistema funzioni) può essere vista come un sistema 3 out-of-N, per cui la reliability del sistema assume una forma ben definita:

Dove è la reliability del singolo processore.

Ora l’obbiettivo del problema è quello di trovare (decidendo se effettuare incrementi o meno) in modo da minimizzare il costo totale del sistema e da garantire una reliability di almeno 2 nines.

Per prima cosa si è notato che per i singoli processori è possibile avere due soli valori di reliability: o 0.8 al costo di 1500$ o 0.96 al costo di 2500$ (cioè 0.8 al costo di 1500 più il 20% di 0.8 e cioè 0.16 al costo di 1000). Valori di reliability superiori ad 1 non sono ammessi.

Inoltre abbiamo da rispettare un vincolo che ci impone di valutare la reliability totale considerando che tutti gli N processori abbiano la stessa reliability.

Per cui il problema si riduce a valutare il valore minimo intero di N che ci da di 0.99 nel caso in cui = 0.8 e il valore minimo intero di N che ci da di 0.99 nel caso in cui = 0.96

A tal fine abbiamo realizzato la seguente function Matlab (basata sulla formula di cui sopra), che presi in ingresso ed N, fornisce il valore di

function [ y] =es4( r,n )

y=0;

for i=0:(n-3)

y=y+nchoosek(n,i)\*(r^(n-i))\*(1-r)^i;

end

end

Eseguendo più volte tale function, facendo variare N da 3 in poi ed tra 0.8 e 0.96 abbiamo ottenuto i seguenti valori di

|  |  |  |
| --- | --- | --- |
| N | R= 0.8 | R=0.96 |
| 3 | 0.512 | 0.884736 |
| 4 | 0.8192 | 0.99090432 |
| 5 | 0.94208 |  |
| 6 | 0.98304 |  |
| 7 | 0.995328 |  |

Dalla tabella notiamo subito che per 0.8 il minimo valore di N che ci restituisce un di almeno 0.99 è 7, mentre per 0.96 è 4.

Poiché noi vogliamo minimizzare il costo, valutiamo quale fra le due configurazioni sia la più economica:

La configurazione più economica che fornisce una reliability totale di almeno due nines è quella che prevede 4 processori ognuno equipaggiato con 1 incremento di reliability del 20%.

# Field Failure Data Analysis

1. Introduzione

In questo elaborato si vuole eseguire un’analisi di alcuni log di errori fatali di grandi sistemi di datacenter. In particolare ci sono stati forniti due file di log di dimensione molto elevata (prelevati in archi di tempo di notevole durata) ottenuti attreverso operazioni di logging presso due sistemi distruibuiti complessi : **Mercury** e **BG/L.**

Nello specifico si vogliono applicare i passi tipici della field failure data analysis per ottenere una stima della reliability di tali sistemi (stima empirica e fitting attraverso funzioni note) , cercare di valutare quali siano i nodi più critici e provare ad accorpare quegli errori che sono da imputare ad uno stesso fault di partenza.

I metodi applicati non ci permetteranno di ottenere informazioni certe, in quanto si basano su delle euristiche non precise e indipendenti dai problemi in esame, ma ci permettono di iniziare a effettuare una prima analisi discriminatoria per questi dati di grandi dimensioni.

La tecnica della FFDA si basa in particolare su tre passi fondamentali:

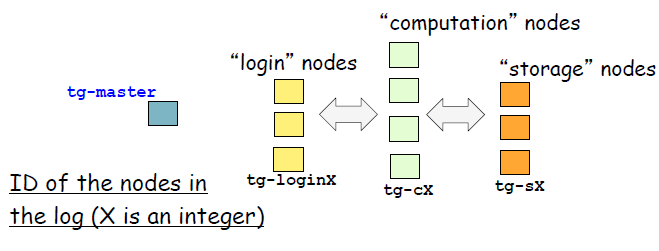
-**filtering:** ci permette di prelevare solo alcuni dati di interesse dai dati di partenza, attraverso l’utilizzo di parole chiave definite nel nostro dizionario di dominio.

- **manipulation:** cerca di raggruppare errori provenienti da uno stesso fault in un insieme detto tupla, attraverso il metodo della finestra di coalescenza, che illustreremo in seguito.

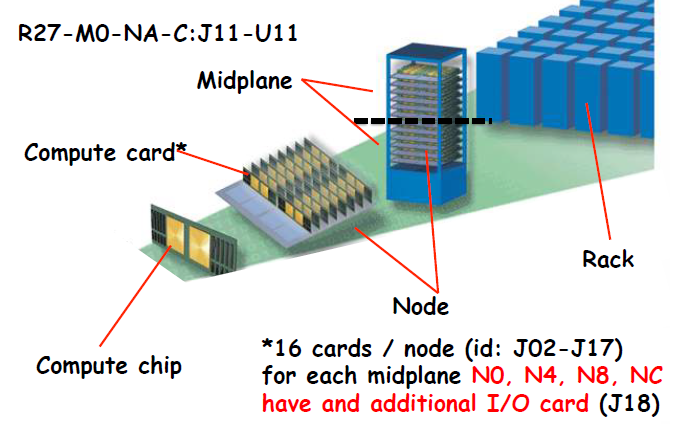
-**analysis:** a partire dai risultati ottenuti cerca di stimare l’andamento della reliability del sistema e altri parametri di interesse.

Per applicare al meglio questa tecnica, ci risulta utile analizzare un po’ l’anatomia dei sistemi che andremo a studiare, in particolare per le fasi di filtering.

Il sistema Mercury consiste di nodi di elaborazione IBM e ha un’architettura a tre livelli più un nodo di gestione (tg-master). Tutti i nodi eseguono un sistema operativo Red-Hat 9.0



Il file di log di Mercury consiste di 80,854 entries di errori fatali. Ognuna di esse segue un formato prestabilito: timestamp + nodo d’origine + sottosistema d’origine + messaggio testuale. Le tipologie di errore si possono riassumere nelle seguenti categorie: DEV, MEM, NET, I-O, PRO.

Per quanto riguarda BG/L la struttura è la seguente:  


In pratica il sistema è composto da una serie di rack ( R) ; ogni rack ha due midplane (M0 ed M1); su ogni midplane ci sono un certo numero di nodi (N) e ogni nodo ha 16 schede (J). Per ogni midplane i nodi N0, N4, N8 e NC hanno una card aggiuntiva per l’IO (J18).

Il file di log di BG/L in esame ha 125,624 entries di errori fatali. Ogni errore ha il f ormato: timestamp + nodo d’origine + card di origine + messaggio di testo.

1. Analisi delle reliability dei due sistemi

In questa sezione andremo a cercare di individuare l’andamento della reliability dei due sistemi in base ai timestamp dei vari errori forniti nei file di log in esame.

Prima di tutto bisogna manipolare i dati per cercare, come già anticipato sopra, di riunire in varie **tuple** errori che provengano da uno stesso fault d’origine. Il metodo utilizzato per compiere questa opera è un metodo basato sulla distanza temporale: si cerca di individuare un intervallo di tempo detto **finestra di coalescenza** (**CWIN**) che sia tale da raggruppare insieme dati di errore vicini. In particolare dato un errore , se la differenza di timestamp tra ed risualta minore o uguale di CWIN allora i due errori fanno parte della stessa tupla. Appena si incontra un errore che è a distanza temporale minore di CWIN dal precedente, allora li inizia una nuova tupla.

Risulta evidente come tale metodologia non sia esatta. Con una data CWIN, supposto anche che sia quella ottima che ci consente di raggruppare al meglio gli errori, ci saranno sempre possibilità di raggruppare assieme errori provenienti da fault differenti perché magari avvenuti in momenti simili (fenomeno della collisione) oppure di raggruppare in tuple differenti errori che hanno la stessa radice (fenomeno del troncamento).

Esiste però una metodologia empirica per valutare la CWIN più idonea possibile con i mezzi a nostra disposizione. Si va a creare un grafico che per una serie di valori di CWIN sull’asse delle ascisse, mappa sulle y i corrispettivi valori del numero di tuple (cioè con una CWIN di grandezza si avrebbero tuple). Questo grafico in genere ha sempre un andamento che va a decrescere velocemente all’aumentare dei valori CWIN per poi stabilizzarsi (o decrescere lentamente) dopo un certo punto detto ginocchio della curva. Il valore migliore di CWIN che noi possiamo selezionare per tuplare al meglio i nostri dati è un valore che si trova appena dopo tale ginocchio.

Una volta scelta la CWIN più idonea si vanno a tuplare i dati e in base ai valori delle durate temporali tra la fine di una tupla e l’inizio della successiva (i cosiddetti interrarivals) si va ad effettuare una stima della reliability.

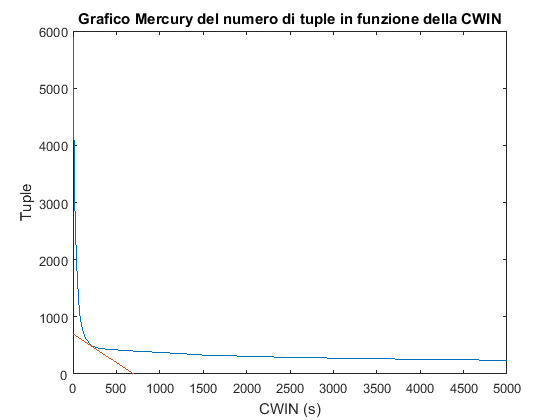
Vediamo di seguito come è stato fatto nello specifico tutto il procedimento, dapprima per il sistema Mercury e poi per Bluegene.

**MERCURY RELIABILITY FITTING**

Il file *MercuryErrorLog.txt* è il file contenente tutti i dati di error log del nostro sistema.

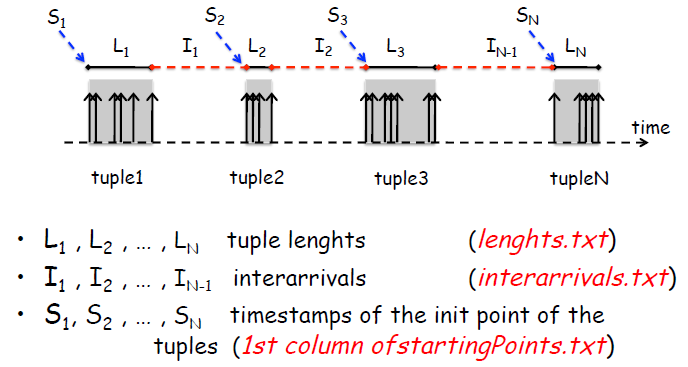
Attraverso l’istruzione da riga di comando da terminale LINUX *$./tupleCount\_func\_CWINpy.sh MercuryErrorLogTEST.txt tentative-Cwin.txt >mercurycwin.csv* eseguiamo lo script *tupleCount\_func\_CWINpy.sh,* che ci va a fornire per ogni valore di CWIN elencato nel file *tentative-Cwin.txt,* il valore del numero di tuple ad esso corrispondente e salva tali valori nel file *mercurycwin.csv.*

In ambiente Matlab andremo a plottare questi valori per avere la curva dei tuple counts in funzione delle CWIN selezionate.



In questa curva andiamo ad individuare più o meno dove si trova il ginocchio, identificandolo con il punto di tangenza tra la curva e una retta a 45 gradi di pendenza. Come valore valido di CWIN selezionabile per l’operazione di tupling, 350 ci è sembrato un valore alquanto ragionevole.

Scelto tale valore eseguiamo la riga di comando *$ ./tupling\_with\_Cwin.sh MercuryErrorLogTEST.txt 350* che esegue lo script bash *tupling\_with\_Cwin.sh*, il quale dato il file di log e la CWIN scelta va a tuplare i dati e genera una cartella con tutte le tuple generate e i file *interrarrivals.txt lenghts.txt* e *startingPoints.txt.* Il contenuto di tali file può essere ben spiegato tramite l’ausilio della seguente immagine.

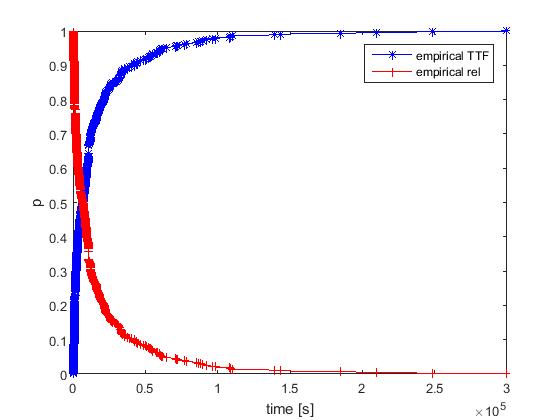


I valori di lenghts sono semplicemente la lunghezza temporale delle tuple mentre i valori di interrerivals sono i valori delle distanza fra la fine di ogni tupla e l’inizio della successiva. Per avere una stima a occhio della bontà della CWIN scelta, abbiamo controllato alcune coppie di tuple successive valutando che nella maggior parte dei casi raggruppavano l’una errori di nodi differenti dall’altra. Non mancavano casi di tuple molto grandi e con errori provenienti da nodi diversi probabilmente affette da collisione ma era impossibile agire sulla CWIN riducendola in modo tale da evitare queste situazioni.

Il passo successivo è stato quello di utilizzare i valori forniti dal file *interrarrivals.txt* al fine di ottenere una CDF empirica del TTF e della reliability.

Per fare ciò si è utilizzata il tool statistico di Matlab *cdfcalc* che a partire dai valori di interrarivals genera una CDF per l’unreliability e complementando quest’ultima anche una CDF per la reliability.

Il grafico per la cfd empirica ottenuta per Mercury è il seguente.



*Figura 5 CDF empirica Mercury*

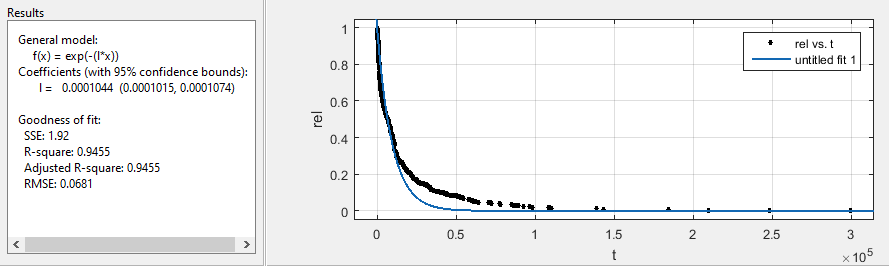
Una volta valutata una CDF empirica per la reliability, vogliamo trovare una forma di una distribuzione nota che fitti in maniera appropriata tale reliability empirica. Le distribuzioni note che utilizziamo come tentativi sono sostanzialmente tre e sono:

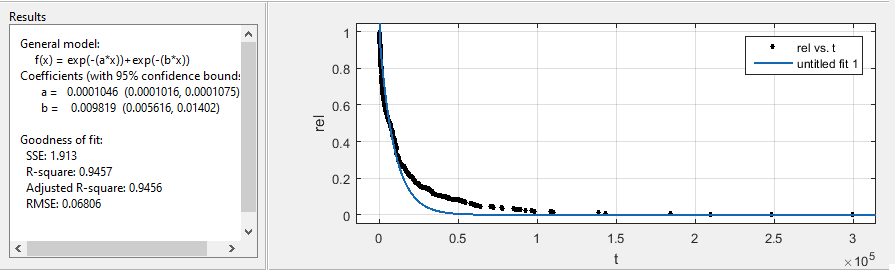
* Esponenziale:
* Weibull:
* Hyperesponenziale:

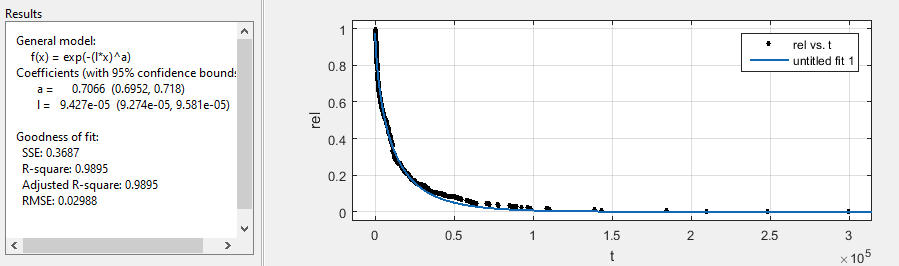
Per ognuna di queste distribuzioni note valutiamo i valori dei parametri che meglio le approssimano alla nostra distribuzione per la reliability.

Per fare ciò si è utilizzato il tool di Matlab cftool. Questo preso i valori della reliability, cerca di fittarli con una funzione nota agendo sui parametri in maniera tale da farlo al meglio con un certo intervallo di confidenza. Fornisce inoltre anche dei valori di SSE e R-square per valutare la bontà del fitting.

Di seguito tutti i risultati ottenuti cercando di fittare la nostra distribuzione con le tre distribuzioni teoriche elencate sopra.







*Figura 6: Tentative fitting per Mercury (esponenziale, hyperesponenziale, weibull)*

Da una prima analisi visiva, e dai valori di SSE e R-square forniti dal tool, si nota subito che la forma Weibull è quella che meglio approssima la nostra distribuzione.

Per avere però un’ulteriore conferma della bontà del fitting, applichiamo il test di **Kolmogorov-Smirnov** tramite l’apposito comando Matlab *kstest2.* Esso prende in ingresso la distribuzione empirica e quella dopo il fitting (assieme ad un eventuale valore che indica il grado di significatività del test a - impostato a 0.1) e fornisce in uscita 3 valori:

* H, che è 0 o 1 a seconda se viene accettata (0) o meno (1) l’ipotesi che i dati seguano la distribuzione
* P che è il p-value, cioè la significatività del test, che se maggiore di a indica che l’ipotesi che i dati vengano accettati non può essere rifiutata
* K che è il valore delle statistiche.

Di seguito i risultati dei test per le tre distribuzioni teoretiche con le quali si cerca di approssimare la reliability di Mercury

ESPO : h=1, p=0.007030614799430, k=0.115662650602410;

HYPER = ESPO.

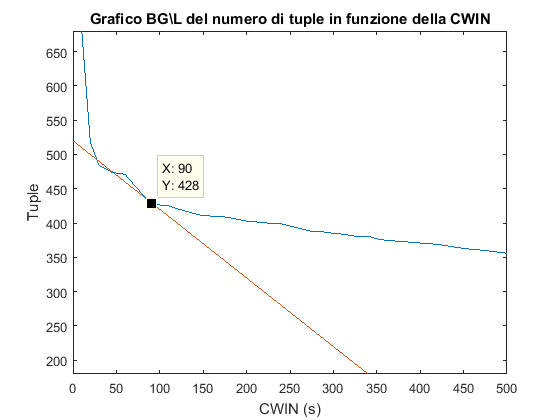
WEIBULL: h=0, p=0.083291942397569, k=0.086746987951807;

Come si può vedere, il risultato migliore si ha con la distribuzione Weibull (l’unica per cui H risulta 0), per cui una buona approssimazione per la nostra reliability è :

Valutando l’integrale tra 0 e +∞ di questa funzione possiamo avere un’approssimazione del MTTF che risulta essere pari 13309 secondi e cioè 3.697 ore.

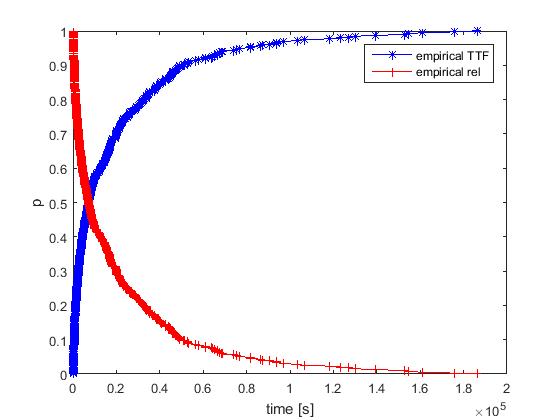
**BG/L RELIABILITY FITTING**

Tutto questo procedimento si può reiterare anche per i dati provenienti da BG/L. Con lo script *tupleCount\_func\_CWINpy.sh* applicator sul file *BGLErrorLog.txt* possiamo avere i valori di tuple count per i vari tentative CWIN. Plottandoli si nota che un valore di CWIN post ginocchio idoneo può essere 250.

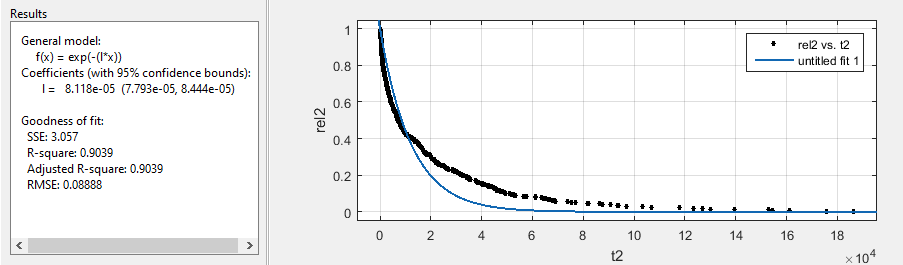


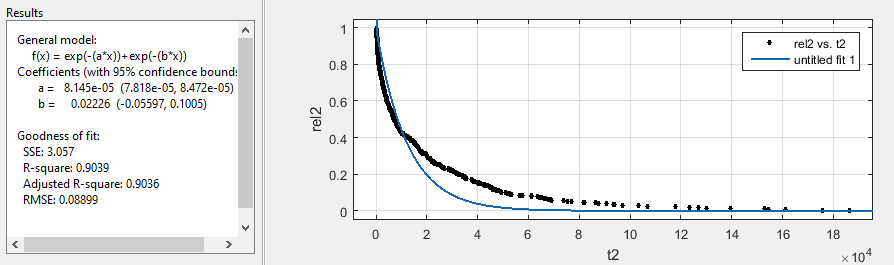
Eseguendo lo script *tupling\_with\_Cwin.sh* con tale valore di CWIN otteniamo una suddivisione in 395 tuple e il file degli interarrivals.

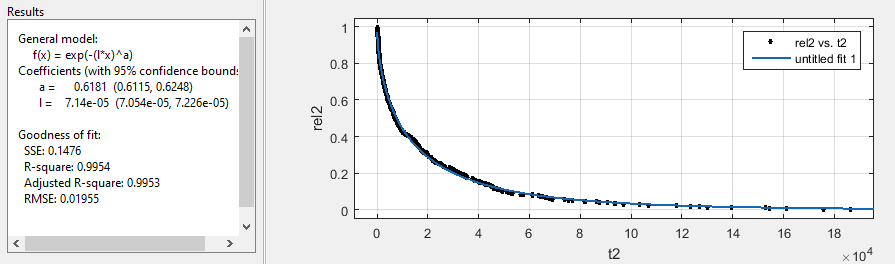
Grazie a questo otteniamo la nostra CDF empirica con cdcalc che poi andiamo a fittare con le tre distribuzioni note tramite cftool. Di seguito il grafico della cdf empirica della reliability per BGL e anche i tre tentativi di fitting.



*Figura 7 CDF empirica BG/L*







*Figura 8 Tentative fitting BG/L*

Applicando kolmogorov-Smirov anche in questo caso notiamo che ancora una volta la Weibull è la distribuzione migliore (H=0 e p=0.294169) e modelliamo la nostra reliability nel seguente modo:

Valutando MTTF si ha un valore di 20294 secondi e cioè 5.637 ore.

Nella seguente tabella riassuntiva il confronto tra i due sistemi in base alla reliability valutata

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| SISTEMA | CWIN | TUPLE COUNT | REALIBITY FIT | MTTF(h) |
|  |  |  |  |  |
| MERCURY | 350 | 440 |  | 3.697 |
|  |  |  |  |  |
| BG/L | 250 | 395 |  | 5.637 |
|  |  |  |  |  |

1. Analisi dei nodi e delle categorie più contributive agli errori

In questa seconda parte ci viene chiesto di selezionare , sia per Mercury che BG/L i top entries-prone nodes, cioè quei nodi che più volte sono presenti come originating node nei file di log.

Tramite lo script *logStatistic.sh* applicato a entrambi i file di log otteniamo una classifica di questi nodi e delle categorie che più contribuiscono agli errori

Tramite lo script *filter.sh* andiamo a isolare in file diversi gli errori relativi ai primi 5 top nodi di Mercury e i primi 5 di BG/L.

Su ognuno di questi nodi ottenuti applichiamo lo script del tuple count e valutiamo di conseguenza una CWIN appropriata con il relativo tuple count (evitiamo di mostrare i grafici cwin-tuplecount per semplicità)

|  |  |  |  |
| --- | --- | --- | --- |
| TOP ENTRIES\_PRONE  NODE MERCURY | OCCORRENZE | CWIN | TUPLE COUNT |
|  |  |  |  |
| Tg-c401 | 62340 | 250 | 46 |
| Tg-master | 4098 | 60 | 68 |
| Tg-c572 | 4030 | 20 | 2 |
| Tg-s044 | 3224 | 40 | 10 |
| Tg-c238 | 1273 | 150 | 24 |

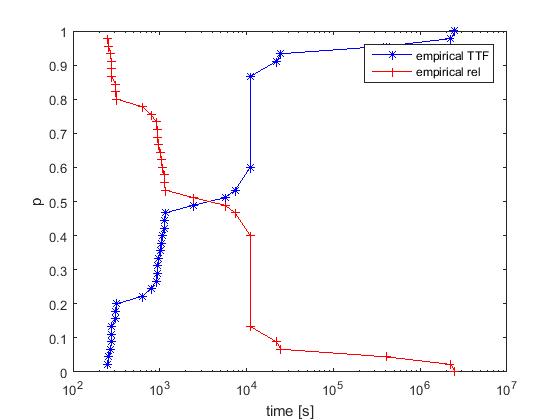
|  |  |  |  |
| --- | --- | --- | --- |
| TOP ENTRIES\_PRONE  NODE BG/L | OCCORRENZE | CWIN | TUPLE COUNT |
|  |  |  |  |
| R71-M0-N4 | 1716 | 250 | 74 |
| R12-M0-N0 | 1563 | 250 | 77 |
| R63-M0-N2 | 976 | 250 | 4 |
| R03-M1-NF | 906 | 250 | 8 |
| R63-M0-N0 | 791 | 250 | 108 |

Possiamo notare che nel caso di BG/L possiamo scegliere una stessa CWIN (250) per tutti i nodi. In verità per R63-M0-N2 e R03-M1-NF erano validi anche valori molto inferiori di CWIN (20 e 5) ma si ha in questi casi che non c’è differenza di tuple count tra quei valori e 250 e quindi si è scelto di uniformare il CWIN. Analizzando i tuple counts non possiamo ravvisare eventuali colli di bottiglia, cioè nodi con un grande numero di tuple rispetto agli altri.

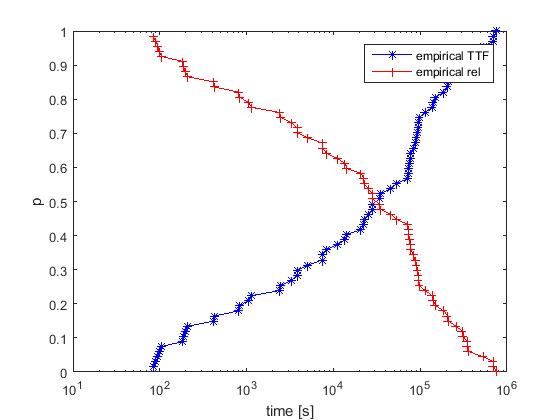
Ottenuti i CWIN possiamo applicare il solito procedimento per ottenere delle stime per le reliability dei top entries-prone nodes, selezionando però solo quelli con un campione degli interrarivals (e quindi un tuple count) abbastanza grande (diciamo almeno 40). In particolare sono stati selezionati tg-c401, tg-master e tg-c238 per Mercury e R71-M0-N4, R12-M0-N0 e R63-M0-N0 per BG/L.

Di seguito tutte le reliability ottenute per i vari nodi

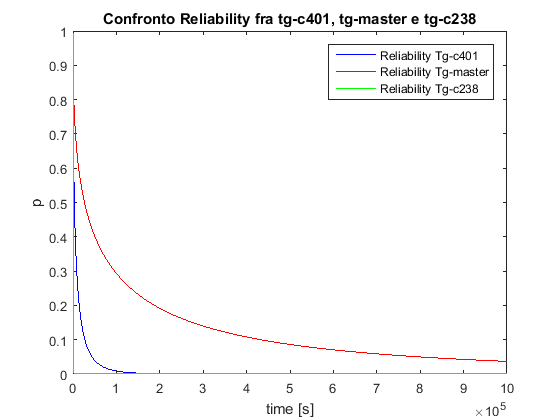
Nota: Per tutti la distribuzione di Weibull è risultata la migliore per il fitting.



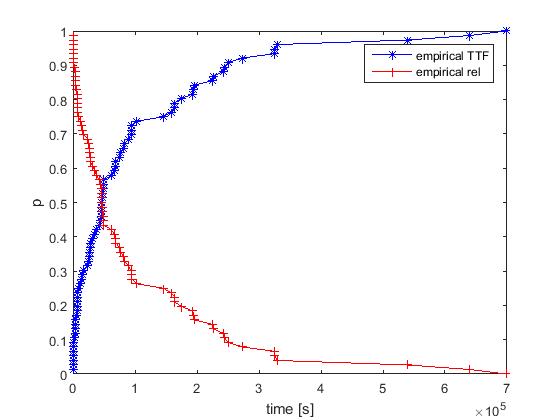
*Figura 9 CDF empirica tg-master in scala semilogaritmica*



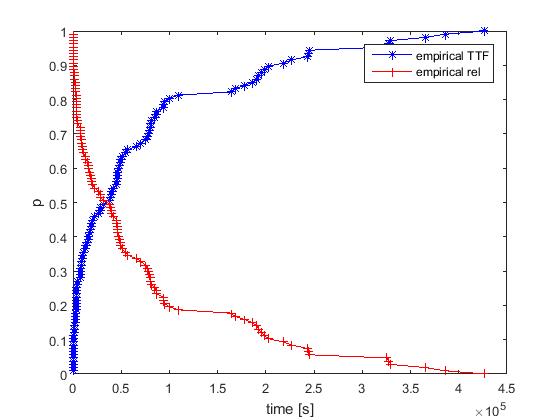
*Figura 10 CDF empirica tg-c401 in scala semilogaritmica*



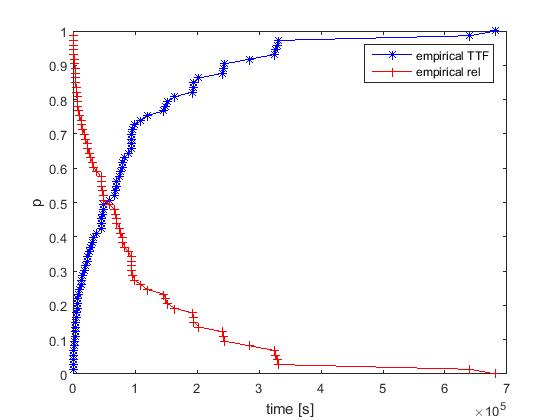
Di seguito i grafici per le CDF empiriche e i fitting per la reliability relativi ai top nodes di BG/L.



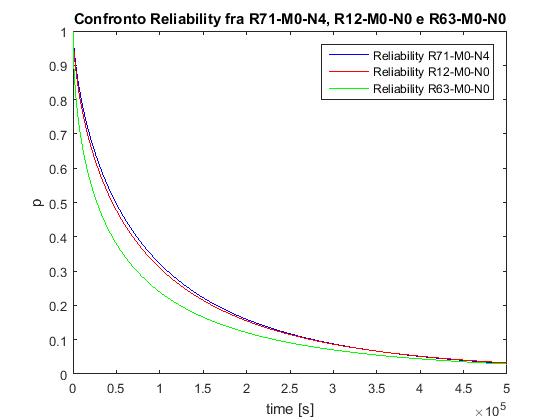
*Figura 11 CDF empirica per R12*



*Figura 12 CDF empirica per R63*



*Figura 13 CDF empirica per R71*



Dai grafici si può notare che per Mercury il tg-Master ha una reliability nettamente superiore al tg-c401, mentre per il BG\L risulta che il nodo R71-M0-N4 abbia una reliability superiore agli altri. Infine possiamo confrontare i nodi anche in base agli MTTF, riassunti nella seguente tabella

|  |  |
| --- | --- |
| TOP ENTRIES\_PRONE  NODE MERCURY | MTTF(h) |
|  |  |
| Tg-c401 | 2.8352 |
| Tg-master | 47.578 |
| Tg-c572 | Non valutato |
| Tg-s044 | Non valutato |
| Tg-c238 | 0.1113 |

|  |  |
| --- | --- |
| TOP ENTRIES\_PRONE  NODE BGL | MTTF(h) |
|  |  |
| R71-M0-N4 | 29.5533 |
| R12-M0-N0 | 29.005 |
| R63-M0-N2 | Non valutato |
| R03-M1-NF | Non valutato |
| R63-M0-N0 | 24.161 |

In seconda istanza, viene chiesto di valutare CWIN, tuple count e un modello per la reliability per tutte le categorie di Mercury (esclusa OTH).

Come fatto precedentemente per i nodi, filtriamo le entries in base alle categorie e otteniamo i vari file di log per categoria.

Applicando il solito script per il tuple count e analizzando i ginocchi delle curve, otteniamo i seguenti valori

|  |  |  |  |
| --- | --- | --- | --- |
| TOP CATEGORIES  MERCURY | OCCORRENZE | CWIN | TUPLE COUNT |
|  |  |  |  |
| DEV | 57248 | 350 | 284 |
| MEM | 12819 | 300 | 75 |
| I-O | 5547 | 80 | 111 |
| NET | 3702 | 80 | 79 |
| PRO | 1504 | 100 | 73 |

Sono tutti valori notevoli di tuple count. Il più grande è DEV, il che vuol dire che stando a questi dati DEV è la categoria che fornisce più errori dovuti a fault differenti.

Con il solito procedimento andiamo a modellare la reliability in base alle varie categorie di errore e di conseguenza gli MTTF

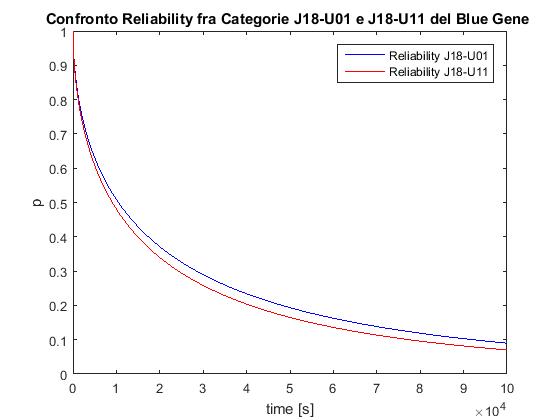


|  |  |
| --- | --- |
| TOP CATEGORIES  MERCURY | MMTF |
|  |  |
| DEV | 4.3389 |
| MEM | 3.8944 |
| I-O | 22.8481 |
| NET | 26.8623 |
| PRO | 32.9074 |

Come si può notare la categoria più reliable è il PRO, mentre quella meno reliable è il MEM prima dei 54400 secondi, mentre a regime è il DEV.

Per BG/L viene chiesto invece di analizzare le categorie J18-U11 e J18-U01, che sono anche le due top categorie di errore (sono quelle dedicate all’ IO). Ancora una volta il procedimento è lo stesso e i risultati ottenuti sono i seguenti:

|  |  |  |  |
| --- | --- | --- | --- |
| TOP CATEGORIES  BG/L | OCCORRENZE | CWIN | TUPLE COUNT |
|  |  |  |  |
| J18-U11 | 50055 | 200 | 283 |
| J18-U01 | 49932 | 200 | 244 |



Il MTTF per U11 risulta di 8.0378 ore mentre per U01 di 9.5648.

1. Analisi di nodi funzionalmente simili e analisi innestate

Viene chiesto di effettuare un focus su alcuni nodi funzionalmente simili in Mercury e BG/L scelti tra quelli top error per valutare quali esibiscono tuple count e parametri di reliability simili.

Ad esempio possiamo considerare i nodi tg-cN (sempre presi da quelli top error) per Mercury cioè quelli adibiti alla computation. Per BGL invece possiamo scegliere i nodi N0 e in particolare R12-M0-N0 e R63-M0-N0.

|  |  |  |
| --- | --- | --- |
| Confronto TG-C  Mercury | TUPLE COUNT | MTTF |
|  |  |  |
| Tg-c401 | 46 | 2.8352 |
| Tg-c572 | 2 | Non valutabile |
| Tg-c238 | 24 | 0.1113 |

|  |  |  |
| --- | --- | --- |
| Confronto N0  BG/L | TUPLE COUNT | MTTF |
|  |  |  |
| R12-M0-N0 | 77 | 29.005 |
| R63-M0-N0 | 108 | 24.161 |

Come si può notare, non si ravvisano parametri simili per nodi con funzionalità simili.

Con gli script a nostra disposizione, è possibile fare anche delle analisi innestate. Ad esempio per Mercury viene chiesto di prelevare i nodi che danno più contributi agli errori ma per ogni tipo. Per fare ciò basta applicare lo script *topStatistic.sh* anziché al file di partenza sequenzialmente ai vari file di log per categoria ottenuti in precedenza.

Si ottengono i seguenti risultati

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| DEV | | MEM | | IO | | NET | | PRO | |
| Node | **Entries** | **Node** | **Entries** | **Node** | **Entries** | **Node** | **Entries** | **Node** | **Entries** |
| tg-c401 | 50782 | tg-c401 | 11558 | tg-s044 | 3208 | tg-master | 3639 | tg-c648 | 616 |
| tg-c572 | 3176 | tg-c572 | 845 | tg-master | 452 | tg-c550 | 33 | tg-c324 | 239 |
| tg-c238 | 1071 | tg-c238 | 197 | tg-login3 | 381 | tg-c196 | 14 | tg-c284 | 180 |
| tg-c242 | 918 | tg-c242 | 149 | tg-s038 | 230 | tg-c238 | 3 | tg-c451 | 159 |
| tg-c117 | 263 | tg-c894 | 28 | tg-c550 | 222 | tg-s044 | 2 | tg-c447 | 136 |
| tg-c669 | 257 | tg-c407 | 14 | tg-login4 | 148 | tg-c027 | 2 | tg-c415 | 61 |
| tg-s176 | 208 | tg-c669 | 10 | tg-login1 | 146 |  |  | tg-c733 | 28 |
| tg-c894 | 187 | tg-c735 | 6 | tg-login2 | 145 |  |  | tg-c027 | 22 |

Si può notare che tg-c401 è il nodo che fallisce di più fra le tutte categorie. Per Dev MEM e PRO, la maggior parte dei nodi che falliscono sono nodi di computation. La maggior parte degli errori di IO, sono da ricondurre al nodo tg-s044, e ciò a senso perché è un nodo di tipo storage. Per quanto riguarda net, si può notare come il nodo master sia un collo di bottiglia per questa categoria. Infine si nota che i nodi con un alto tasso di fallimento sono di tipo computation.

Infine è stato chiesto di valutare per quanto riguarda BG/L quali fossero i rack con il maggior numero di entries. Per far ciò, si è modificato il file di log BG/L, per separare gli identificativi dei rack dal resto del nodo e si è modificato lo script bash *LogStatistics.sh* in modo tale che restituisse le occorrenze delle entries relative ai rack.   
I risultati sono forniti nella tabella seguente:

|  |  |
| --- | --- |
| TOP RACKS  BG/L | OCCORRENZE |
|  |  |
| R63 | 7192 |
| R62 | 4057 |
| R57 | 3581 |
| R56 | 3527 |
| R46 | 3485 |
| R03 | 2982 |
| R12 | 2976 |
| R71 | 2815 |
| R61 | 2366 |
| R36 | 2230 |
| R60 | 2222 |
| R65 | 2113 |
| R21 | 2107 |
| R30 | 2079 |
| R64 | 2069 |
| R05 | 2019 |
| R02 | 1982 |
| R55 | 1963 |
| R67 | 1883 |
| R23 | 1838 |