Università degli Studi di Napoli "Federico II"

Scuola Politecnica e delle Scienze di Base
Corso di Laurea Magistrale in Ingegneria Informatica

Giuseppe Francesco Di Cecio - M63001211
Nicola D'Ambra - M63001223
Emma Melluso - M63001176

Elaborato di *Impianti di Elaborazione*

Università degli Studi di Napoli *Federico II*
Napoli
A.A 2020/2021

Indice

Capitolo 1

Workload Characterization

Il dataset di partenza è composto da **3000 righe** e **24 colonne**, ciascuna delle quali rappresenta uno dei parametri del sistema oggetto di studio. Si tratta di parametri caratterizzanti l'esecuzione di vari Threads su un sistema operativo.
In particolar modo le colonne con prefisso Vm rappresentano informazioni sulla memoria virtuale occupata e utilizzata dai Threads, mentre le altre colonne rappresentano informazioni di carattere generale, come memoria libera, numero di threads, pagine inattive ecc.

1.1 Filtraggio

1.1.1 Colonne Identiche

Innanzitutto osservando il workload e effettuando un grafico delle distribuzioni è possibile rendersi conto della presenza di ben 4 colonne costanti:

- **Active**
- **AnonPages**
- **AvbLatency**
- **Error**

Tali colonne in quanto costanti non spiegano varianza, dunque possono essere tranquillamente trascurate ai fini dell'analisi.
Osservando le distribuzioni dei parametri **WriteBack** e **MemFree** si sono notate alcune caratteristiche comuni. Per avere una maggiore chiarezza si è preferito calcolare la matrice delle correlazioni su questi due parametri.

	'MemFree'	'Writeback'
'MemFree'	1,0000	1,0000
'Writeback'	1,0000	1,0000

Figura 1.1: *Matrice di correlazione tra MemFree e WriteBack*

Osservando la matrice appare evidente che le due colonne sono esattamente identiche, fornendo quindi la stessa informazione. Per questo motivo si è deciso di trascurare una delle due, in particolare quella di WriteBack.

Le 24 colonne iniziali sono state ridotte a 19 colonne, riducendo il dataset di un numero di osservazioni pari a:

$$n_{dati} = 3.000 \times (24 - 19) = 15.000 \tag{1.1}$$

1.1.2 Outlier

Gli outliers sono valori anomali all'interno dell'insieme di osservazioni, in altre parole sono valori che si discostano notevolmente dagli altri valori dell'insieme. Essendo valori anomali la loro frequenza di occorrenza è bassa rispetto agli altri valori e ciò li porta ad essere identificati all'esterno del range interquartile. In alcuni casi gli outlier possono essere eliminati ma ciò è possibile solo a monte di una analisi accurata. Tali valori infatti influiscono sulle analisi statistiche in modo considerevole e non sempre rappresentano situazioni trascurabili per l'analisi da svolgere . Osservando attraverso box plot e grafici di distribuzione l'andamento dei seguenti parametri :

- **VmSize** (quanta memoria virtuale utilizza l'intero processo);
- **VmHWM** (di quanta RAM il processo necessita al massimo);
- **VmRSS** (quanta RAM il processo sta correntemente usando);
- **VmPTE** (quanta memoria Kernel è occupata dalle entries della tabella delle pagine);

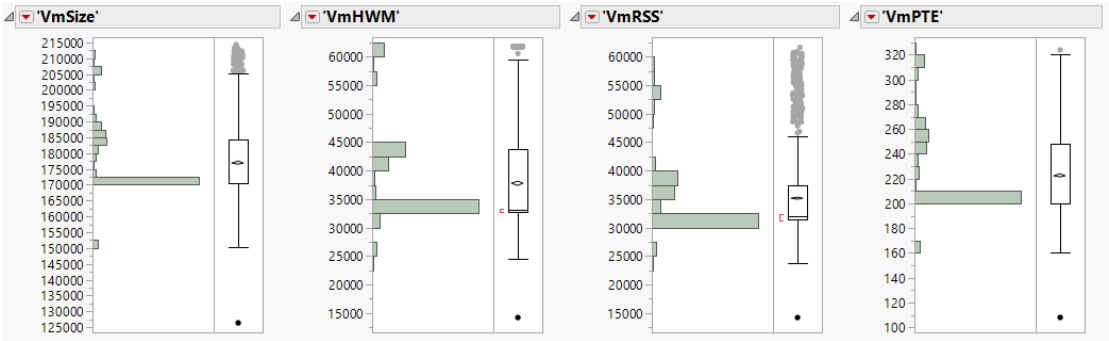

Figura 1.2: *Grafici di distribuzione di VmSize, VmHWM, VmRSS, VmPTE*

Si è notato che essi presentano un outlier isolato (in basso ad ogni grafico) in comune associato alla prima riga del dataset. Analizzando gli altri parametri (*MemFree*, *Dirty*, *PageTables*, *Buffer*, ...) è stato possibile evidenziare che anche per la maggior parte di essi lo è, ma non è un punto isolato.
L'ipotesi fatta è che con molta probabilità le prime righe del dataset (da 0 a 100 circa), rappresentano la fase di avvio del processo e l'outlier oggetto di studio è la prima istanza di questa fase. Dato che l'obiettivo della caratterizzazione del workload è quello di analizzare le prestazioni a regime del sistema oggetto di studio (in questo caso), si è deciso di trascurare quel singolo outlier. Tuttavia è bene notare che in ogni caso le informazioni riguardo questa fase di avvio non saranno del tutto perse dato che è stato rimosso un singolo punto e non tutti i punti che la rappresentano.

Un secondo outlier che può essere agevolmente rimosso è la riga 512 in cui il parametro **Slab** assume valore 4.

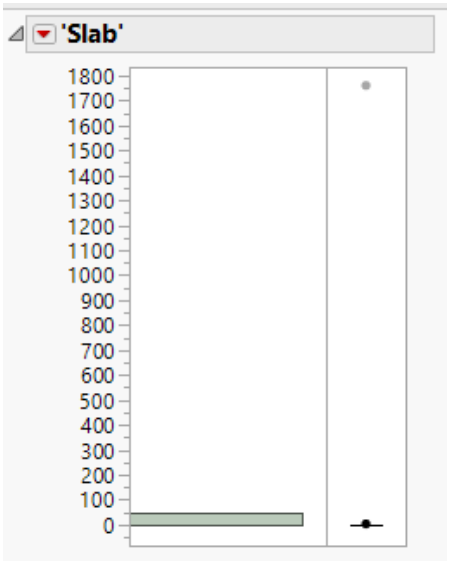

Figura 1.3: *Grafico di distribuzione di Slab*

Oltre ad avvicinarsi molto al valore medio assunto da Slab (zero), esso risulta essere un outlier solo per il parametro stesso dato che per gli altri è un valore compreso tra i quartili. Una sua rimozione quindi non influenza gli indici di caratterizzazione sintetica dei parametri del workload complessivo.

Un terzo outlier è il valore 1760 del parametro *Slab*, associato alla riga 90 del workload.

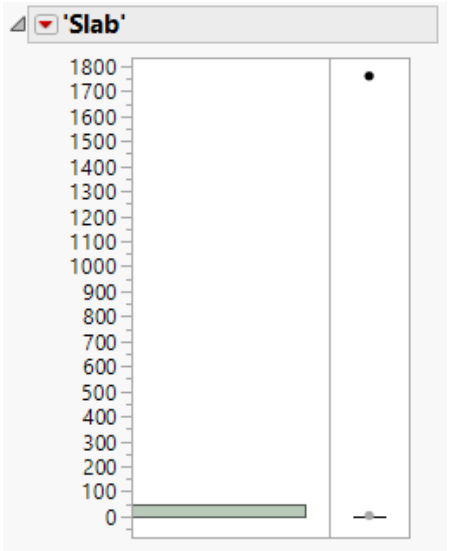

Figura 1.4: *Grafico di distribuzione di Slab*

Rispetto al precedente, tale outlier richiede un' analisi più approfondita visto che influenza significativamente l'andamento di parametri quali *Mapped* e *PageTables*. Per

descrivere meglio la dipendenza tra questi parametri si può effettuare un grafico tra il numero dell'osservazione e il valore assunto da *Mapped*, analogo discorso con *PageTables*.

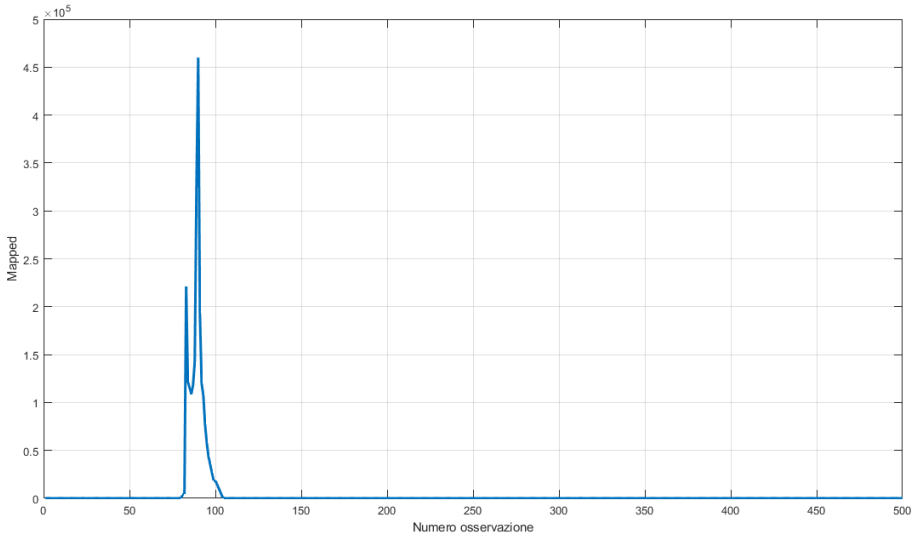

Figura 1.5: *Grafico tra numero di osservazione e valore assunto dal parametro Mapped*

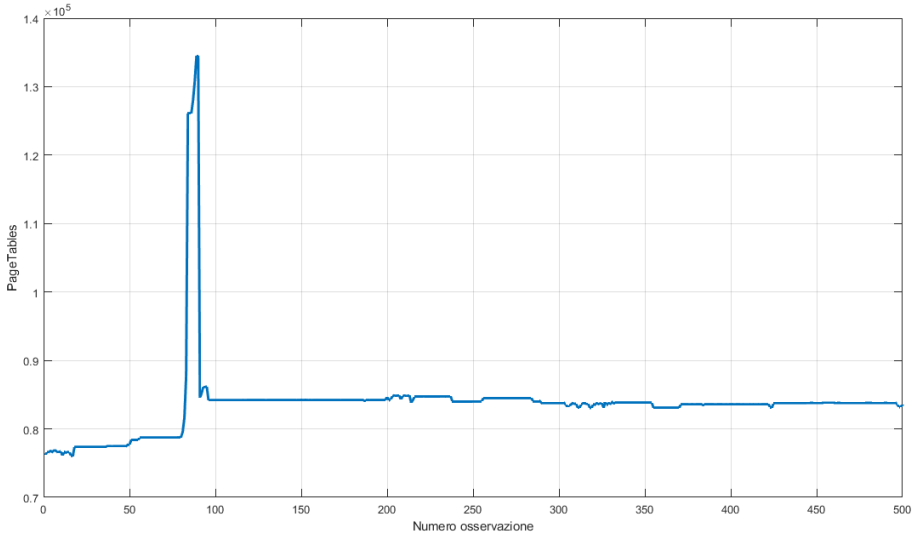

Figura 1.6: *Grafico tra numero di osservazione e valore assunto dal parametro PageTables*

Si nota che in corrispondenza (in realtà nell'osservazione appena precedente) dell'outlier del parametro *Slab* i due parametri sopra indicati hanno un picco, durante la fase di avvio del sistema.
Lo Slab si riferisce ad un particolare meccanismo di allocazione/deallocazione della memoria nel Kernel. Dato che influenza in particolar modo altri parametri si è preferito non trascurarlo.
In conclusione sono stati eliminati dal dataset solo 2 outlier che corrispondono a 38 osservazioni. Quindi il dataset è stato ridotto in totale di 15.038 elementi, provocando una

diminuzione dei dati iniziali di poco più del 20%.

1.2 PCA

A seguito del filtraggio il dataset risulta ridotto grazie alla rimozione di alcune colonne che non esprimevano varianza e alcune righe rappresentanti outlier trascurabili.
Su questo dataset si possono quindi iniziare a fare le prime considerazioni.
Utilizzando la tecnica della *Principal Component Analysis* il dataset può essere estremamente ridotto, sfruttando solo le *Componenti Principali* che mantengono più varianza.
Il risultato della PCA è quindi:

Numero	Autovalore	Percentuale	20 40 60 80	Percentuale cumulativa
1	9,8395	51,787		51,787
2	3,6873	19,407		71,194
3	2,4576	12,935		84,129
4	0,7782	4,096		88,225
5	0,7075	3,724		91,949
6	0,6258	3,293		95,242
7	0,3409	1,794		97,036
8	0,2658	1,399		98,435
9	0,1612	0,848		99,283
10	0,0904	0,476		99,759
11	0,0178	0,094		99,853
12	0,0114	0,060		99,913
13	0,0078	0,041		99,954
14	0,0058	0,030		99,985
15	0,0014	0,007		99,992
16	0,0009	0,005		99,997
17	0,0006	0,003		100,000
18	0,0000	0,000		100,000
19	0,0000	0,000		100,000

Figura 1.7: *PCA applicata al dataset filtrato*

La scelta del numero di componenti principali ricade in particolar modo sulla devianza che quelle componenti mantengono rispetto al dataset reale. Inoltre essa dipende anche dal tipo di osservazioni ed esperimento che è stato effettuato.
In questo caso la scelta è ricaduta sul prendere 5 componenti principali poiché rappresentano il 92% della devianza totale. Esso rappresenta un valore abbastanza elevato, ma è stato scelto per mantenersi in una regione di tolleranza durante la clusterizzazione.
Anche se il workload sintentico verrà costruito considerando 5 componenti principali, in seguito sono riportati i risultati di PCA e clusterizzazione anche nel caso in cui fosse stato scelto un numero diverso di componenti principali, ovvero:

- Prendere 4 PC
$$DEV_{PCA-MANTENUTA} \approx 88\%$$
- Prendere 5 PC
$$DEV_{PCA-MANTENUTA} \approx 92\%$$
- Prendere 6 PC
$$DEV_{PCA-MANTENUTA} \approx 95\%$$

Per ognuno di questi 3 insiemi di Principal Components è stata effettuata la procedura di clustering .

1.3 Clustering

Il clustering è una tecnica che consiste nel raggruppare osservazioni "simili" tra loro. La similitudine tra un elemento e un cluster, o tra un cluster e un altro cluster, può essere calcolata secondo varie tecniche. In questa analisi si è preferito utilizzare il **metodo di Ward**, il quale pesa la distanza tra due cluster in relazione al numero di elementi che li compongono.
Dati due cluster P e Q (un elemento non appartenente ad un cluster, può essere visto come un cluster di dimensione 1), sia $|P|$ la cardinalità di P, analogo con $|Q|$, e sia $\bar{x}_p$ il centroide di P, analogo con $\bar{x}_q$, la distanza tra P e Q viene calcolata come:

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

Per ogni raggruppamento viene poi scelto una singola osservazione che la rappresenta, riducendo quindi il numero di righe del dataset pari al numero di cluster scelti durante l'analisi.
Il numero di cluster da scegliere può dipendere da vari fattori:

- Omogeneità dei cluster: i cluster devono raggruppare un numero di osservazioni quanto il più possibile omogeneo rispetto agli altri cluster. Avere un cluster con un numero di elementi di vari ordini di grandezza rispetto ad un altro cluster non sempre può portare a buoni risultati (in termini di devianza).
- Devianza mantenuta: a seguito della PCA parte della devianza nei dati viene persa. Dato che il clustering viene effettuato sulle *Componenti Principali* allora esso produce un'ulteriore perdita di devianza nel risultato finale.

Devianza Persa

Per effettuare il calcolo della devianza totale persa persa bisogna prima calcolare la devianza intra-cluster (la somma delle devianze per ogni cluster) e sulla base di questa si può calcolare la quantità richiesta.
Matematicamente, definita $DEV_{PCA-PERSA}$ la devianza persa (in termini percentuali) a causa della PCA, viceversa $DEV_{PCA-MANTENUTA}$ la devianza mantenuta dalla PCA, e DEV_{INTRA} la devianza intra-cluster, la devianza totale persa percentuale vale:

$$DEV_{PCA-LOST} + DEV_{INTRA} \times DEV_{PCA-MANTENUTA}$$

Essa può essere calcolata in MATLAB passando ad uno script i cluster, le componenti principali e il dataset iniziale.
Per dare valore ai fattori sopra citati il clustering viene effettuato scegliendo un numero di cluster che varia da 6 a 16 (per ogni gruppo di componenti principali) su cui poi viene calcolata la devianza persa.

1.3.1 4 Componenti Principali

Utilizzando le prime quattro PC si ha:

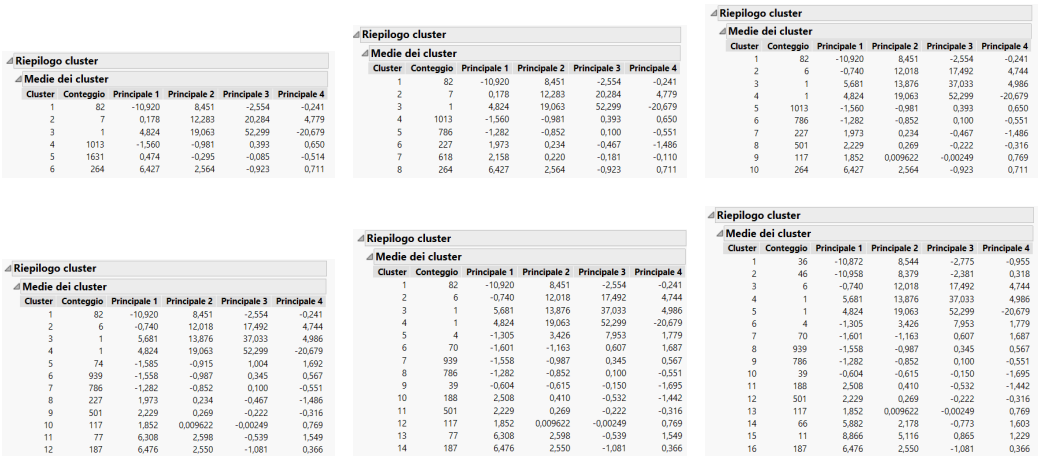

Figura 1.8: *Numero di cluster e dimensione per diversi valori*

La devianza persa durante la clusterizzazione varia in relazione al numero di cluster scelti. Si possono racchiudere le informazioni in un'unica tabella:

6 Cluster	8 Cluster	10 Cluster	12 Cluster	14 Cluster	16 Cluster
26%	17%	16%	15%	14.5%	14%

1.3.2 5 Componenti Principali

Utilizzando le prime quattro PC si ha:

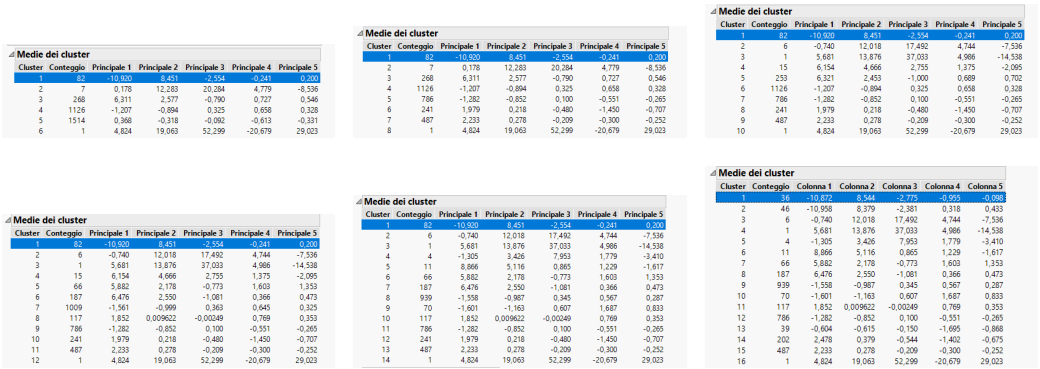

Figura 1.9: *Numero di cluster e dimensione per diversi valori*

La devianza persa durante la clusterizzazione varia in relazione al numero di cluster scelti. Si possono racchiudere le informazioni in un'unica tabella:

6 Cluster	8 Cluster	10 Cluster	12 Cluster	14 Cluster	16 Cluster
25.5%	16%	15%	12%	11%	10%

1.3.3 6 Componenti Principali

Utilizzando le prime quattro PC si ha:

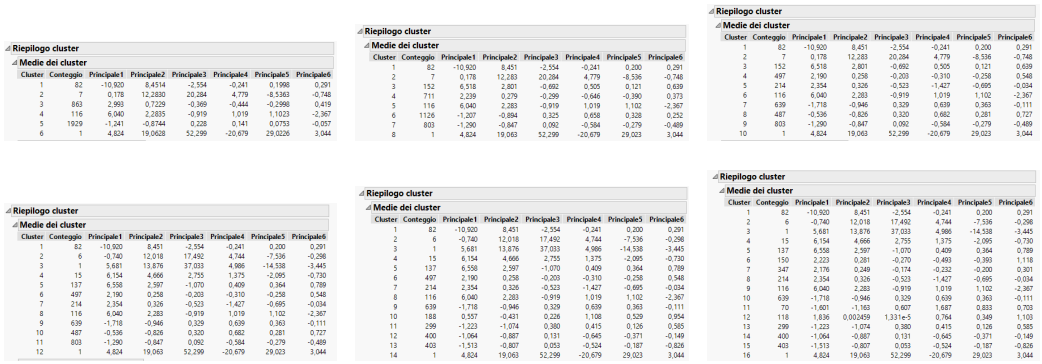

Figura 1.10: *Numero di cluster e dimensione per diversi valori*

6 Cluster	8 Cluster	10 Cluster	12 Cluster	14 Cluster	16 Cluster
22%	15%	13%	12%	11%	9%

1.3.4 Interpretazione

All'inizio dell'analisi sono state effettuate delle ipotesi che hanno trovato riscontro nella procedura di clustering:

1. La fase iniziale del sistema (descritto dalle prime righe) trova riscontro con il cluster numero uno qualunque siano le componenti principali e qualunque sia il numero di cluster scelto. Questo quindi prova l'ipotesi definita inizialmente

2. L'ultimo cluster contiene sempre un elemento singolo. Questo accade a causa del fatto che è stato identificato un picco nella fase iniziale delle misure. Esso inoltre è stato definito grazie all'outlier nel parametro Slab che non è stato eliminato, di conseguenza il picco viene racchiuso in un unico cluster in tutte le situazioni.

In conclusione si può costruire una tabella che racchiude le informazioni riguardanti le PCA e la clusterizzazione in termini di percentuale di devianza persa.

	6 Cluster	8 Cluster	10 Cluster	12 Cluster	14 Cluster	16 Cluster
4 PC	26%	17%	16%	15%	14.5%	14%
5 PC	25.5%	16%	15%	12%	11%	10%
6 PC	22%	15%	13%	12%	11%	9%

1.4 Workload Sintetico

Come anticipato nel paragrafo precedente, sono state scelte 5 PC e per non perdere troppa varianza ma al tempo stesso non sfociare in un numero di cluster molto elevato, si è scelto di considerare 10 cluster. In tal caso la perdita di devianza con PCA e Cluster è di circa del 15%.
In conclusione dopo aver calcolato i centroidi con uno script MATLAB il workload sintetico risulta essere:

VmPeak'	VmSize'	VmHWM'	VmRSS'	VmPTE'	Threads'	MemFree'	Buffers'	Cached'	Inactive'	Dirty'	Mapped'	Slab'	PageTables'	Committed_AS'	NumOfAlloc...	proc-fd'	avgThrough...	avgElaps...	Cluster
152272	150216	25228	25208	160	58	5604588	33440	334672	141440	304136	148	0	77472	23772	27852	7192	294280	1530	1
153484	151436	26612	26468	160	20	5044360	83684	732680	365572	576940	115360	0	126116	28324	91172	7984	347336	2040	2
170344	170340	31144	31140	200	52	4655664	89480	1090572	541220	773332	324400	1760	134500	29956	106788	8028	365264	2040	10
170344	170340	31144	31140	200	43	4471580	94884	1259568	597900	891332	459632	0	134504	29956	116216	8024	365692	1530	3
172392	170340	32696	31516	200	35	4608100	158180	1117364	424808	934596	176	0	83860	23740	110212	7208	318956	1020	7
172392	170340	32696	31340	200	35	4605508	160628	1117388	427232	934468	152	0	83684	23740	110320	7208	318508	1530	6
192256	184832	43804	37584	252	87	4557264	198720	1118136	488652	918124	56	0	89880	23848	113836	7260	329604	510	8
192384	185460	43804	37396	256	97	4555172	201052	1118212	494300	914696	36	0	89736	23848	113556	7264	330928	1020	9
215352	210232	61784	59084	316	116	4527992	204160	1118424	526896	908536	148	0	112848	23996	113028	7772	356604	510	4
215352	206136	61784	53204	312	9	4539096	204400	1118460	520240	908140	148	0	105548	23848	110300	7320	348476	510	5

Figura 1.11: *Workload sintetico*

Capitolo 2

Web Server - Capacity Test

L'obiettivo del Capacity Test è quello di valutare le performance di un qualsiasi sistema quando è sottoposto a carichi di lavoro di diversa intensità, in modo da caratterizzare le sue prestazioni al limite (sotto condizioni di lavoro severe).
Per realizzare queste valutazioni sono necessari gli **high-level parameters**, ovvero tutti quei parametri reperibili ed osservabili lato client. Essi possono riferirsi alla richiesta (quando è stata fatta, chi l'ha fatta ecc..) o alla risposta (tempi di risposta, errori).
Essendo il sistema in questione un server, si è scelto di descrivere le sue performance attraverso:

1. **Response Time**, intervallo di tempo che intercorre tra l'istante in cui il client inoltra la richiesta e quello in cui riceve la risposta.
2. **Throughput**, richieste servite correttamente per unità di tempo.

L'andamento atteso da parte di queste due metriche è il seguente:

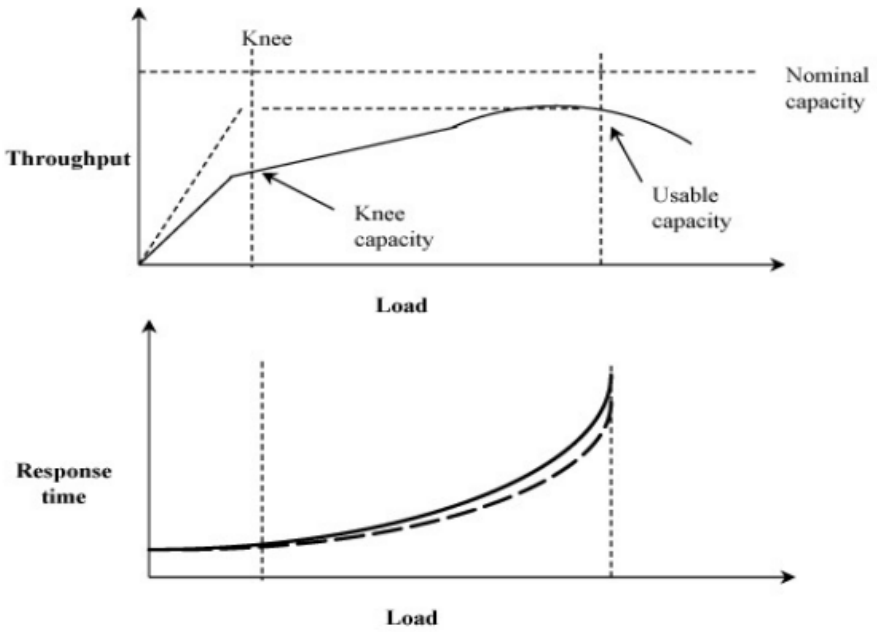

Figura 2.1: *Grafici Throughput e Response time*

Di nostro interesse sono i valori di:

- *Knee Capacity*, punto prima del quale il throughput cresce linearmente all'aumentare del carico, ma il tempo di risposta non varia significativamente ed oltre il quale il guadagno in throughput è basso mentre il tempo di risposta aumenta con il carico.
- *Usable Capacity*, massimo throughput raggiungibile portando il sistema al limite, senza eccedere un dato tempo di risposta.

Per ottenere agevolmente la Knee Capacity, viene introdotto un terzo parametro, la *Potenza*.

$$Power = \frac{Throughput}{ResponseTime} \tag{2.1}$$

Tale punto coincide con il punto di massimo della potenza e rappresenta l'ottimo in corrispondenza del quale conviene operare per ottenere le prestazioni migliori.

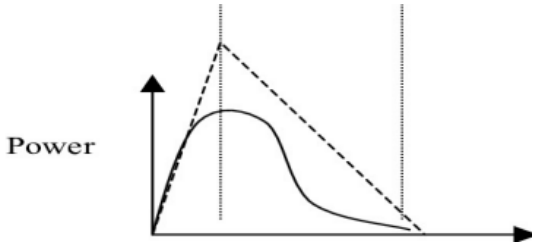

Figura 2.2: *Grafico Potenza*

2.1 Experimental Setup

Il sistema oggetto di studio è un *Web Server Apache* installato sulla macchina virtuale guest, che funge da server.
Tramite la modalità *Host-only Network Adapter*, configurabile nelle impostazioni della macchina virtuale, è stato possibile far comunicare la macchina guest con quella host, che lo ospita. Su quest'ultima è stata installata l'applicazione Java *JMeter*, che ha permesso l'analisi delle prestazioni complessive del Server, sottoponendolo a diversi tipi di carico.

In questa analisi è stato scelto di valutare le prestazioni in media del server, considerando solo richieste (HTTP di tipo GET) casuali. Esse sono differenziate dalla dimensione della risorsa che chiedono al server.

2.1.1 Server Setup

Il server è stato installato su una macchina virtuale *Ubuntu 2021* in esecuzione su una macchina host di uso comune. Essa è stata dotata di circa 4GB di RAM e di 2 processori (intel I5-5200u con frequenza massima di 2.70 GHz).
Per creare un scenario reale, sul Server sono state caricate 5 pagine in formato testuale, di diversa dimensione:

- **Small**: 50 KB

- **Small-Medium**: 100 KB
- **Medium**: 300 KB
- **Medium-Large**: 500 KB
- **Large**: 1 MB

Questi sono i file oggetto delle richieste realizzate da ipotetici client.

2.1.2 Clients Setup - JMeter

Innanzitutto è stato settato, nel *ThreadGroup*, il numero di thread che JMeter usa per realizzare i test. Questa quantità rappresenta il numero di utenti "virtuali" che visitano il nostro server. Nel nostro esperimento sono stati previsti **50 threads**, un valore in linea con i suoi scopi (dato che si tratta di un banale webserver virtualizzato su una macchina host di uso comune). In più, prevedendo dei test di durata pari a *5 min*, sono stati impostati:

- il **Ramp-up period** - numero di secondi entro il quale deve essere attivato l'ultimo thread - a *300 s*. Ciò ci ha permesso di dilazionare l'attivazione degli utenti nei 5 minuti.
- il **Thread lifetime** - durata massima di ogni thread - a *300 s*.
- il **Loop count** - numero di volte in cui un singolo thread effettua una richiesta. Esso corrisponde al numero di richieste nell'intervallo di tempo di simulazione (in questo caso 300s) diviso il numero di threads.

Al ThreadGroup sono stati aggiunti 5 *HTTP Request Sampler*, uno per tipologia di richiesta da realizzare e nei quali sono stati specificati i path delle rispettive risorse sul server. Ad essi è stato integrato un *Random Controller*, grazie al quale, quando un thread viene attivato, effettua solo una tra le cinque tipologie di richieste, selezionata in maniera randomica. Ancora una volta, aggiungendo variabilità alle nostre richieste, è stato possibile simulare una situazione realistica e, soprattutto, non predicibile.
Attraverso il *Constant Throughput Timer* è stato possibile impostare il carico da sottoporre al sistema, in termini di numero di richieste al minuto. Infine, il listener *Simple Data Writer*, ci ha permesso di collezionare in un file, quei parametri di alto livello che sono d'interesse ai fini dell'esperimento.
L'idea è quella di simulare quindi 50 utenti, di cui ognuno effettua un numero di richieste in relazione al carico, per 5 min.
Ciò equivale a:

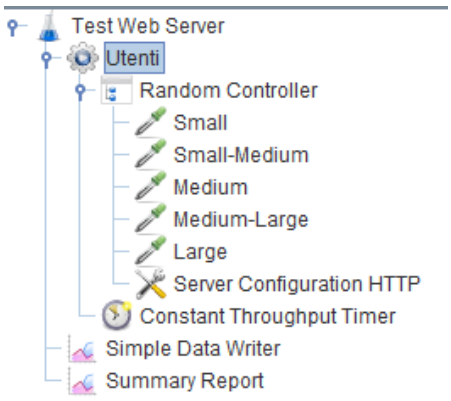

Figura 2.3: *Configurazione delle richieste e del carico in JMeter*

I risultati, grazie al Simple Data Wirter, vengono salvati in formato .csv, i cui paramentri vengono raggruppati in forma tabellare.

TimeStamp	Elapsed	Latency	...
⋮	⋮	⋮	⋮

2.2 Esecuzione Capacity Test

Inizialmente sono stati effettuati dei test il cui scopo era quello di far operare il Web Server "al limite". Ci si è resi conto che il massimo valore di carico entro il quale il sistema risponde adeguatamente (in quelle condizioni), è di circa *6000 richieste al minuto*.
A partire da questo limite e ragionando sull'andamento di throughput e response time, si sono scelti i seguenti valori di carico da sottoporre al sistema:

$$workloads = 100, 500, 800, 1000, 2000, 3000, 4000, 5000, 6000, 7000, 8000, 9000$$

Gli ultimi tre carichi (7000, 8000 e 9000 richieste al minuto) hanno permesso di evidenziare nei grafici il degradamento delle prestazioni del sistema.
Per ogni valore di carico è stato calcolato il **Throughput**:

$$Throughput = \frac{NumeroRichieste}{Timestamp(N) - Timestamp(1)} \quad \left[\frac{N}{s}\right] \tag{2.2}$$

Il *timestamp* fa parte degli high-level parameters collezionati dal Simple Data Writer, e corrisponde all'istante di tempo (in millisecondi poi convertito in secondi) in cui il client ha inoltrato una data richiesta.
Come **Response Time** è stato scelto il parametro *Elapsed*, coincidente con il tempo che intercorre tra la sottomissione della richiesta da parte del client e la risposta del server. Dato che contiene anche il tempo di elaborazione della richiesta da parte del server, esso cresce all'aumentare della dimensione dei file richiesti, oltre che all'aumentare del carico.

Ogni misurazione (per ogni carico) è stata ripetuta **3 volte** in modo da tenere traccia dell'errore, e notando che i dati ottenuti non differivano di molto tra loro, come indice di posizione è stata scelta la loro media.

2.2.1 Risultati

I file .csv sono stati caricati in uno script Matlab tramite cui sono stati automatizzati i procedimenti descritti sopra, i parametri sono stati plottati in funzione del carico considerato. differenziale).

```
%% Data
workloads = [100 500 800 1000 2000 3000 4000 5000 6000 7000 8000 9000];
througputs = zeros(1, length(workloads));
resp_times = zeros(1, length(workloads));
k = 1; %Indice di riferimento per i due vettori

%% Elaborazione
for i = workloads
        mean_resp_t = zeros(1,3);
        thr = zeros(1,3);
        for j = 1:3
                path = strcat(num2str(i,
↪  '%d'),'\dati',num2str(j,'%d'),'.csv');

                %Data from Jmeter output file (csv format)
                simple_data = readmatrix(path);

                %Calculating the number of requests
                [N, M] = size(simple_data);
                num_req = N; %Number of requests

                %Throughput = number_of_requests_completed /
↪  time_window_of_the_experiment
                t_wind_mills = simple_data(num_req,1) - simple_data(1,1);
↪  %Time window (milliseconds)
                t_wind_sec = t_wind_mills/1000; %Time window (seconds)
                thr(j) = num_req/t_wind_sec; %Throughput

                %Average response time
                elap_times = simple_data(:,2);
                mean_resp_t(j) = mean(elap_times);
        end

        check deviazione_std
        COV_thr(k) = std(thr)/mean(thr);
        COV_resp(k) = std(mean_resp_t)/mean(mean_resp_t);

        if COV_thr(k) > 0.5
                througputs(k) = median(thr);
        end

        if COV_resp(k) > 0.5
                resp_times(k) = median(mean_resp_t);
        end

end

power = througputs./(resp_times/1000);
power_max = max(power);
KNEE_CAPACITY = througputs(find(power == power_max));
```

Effettuando un grafico dei risultati:

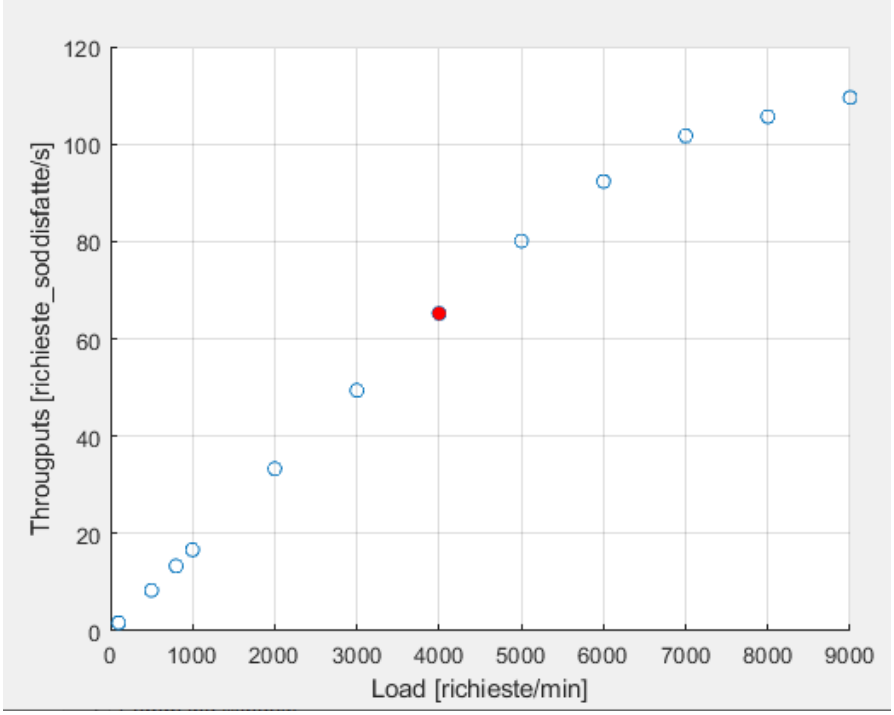

Figura 2.4: *Grafico Throughput*

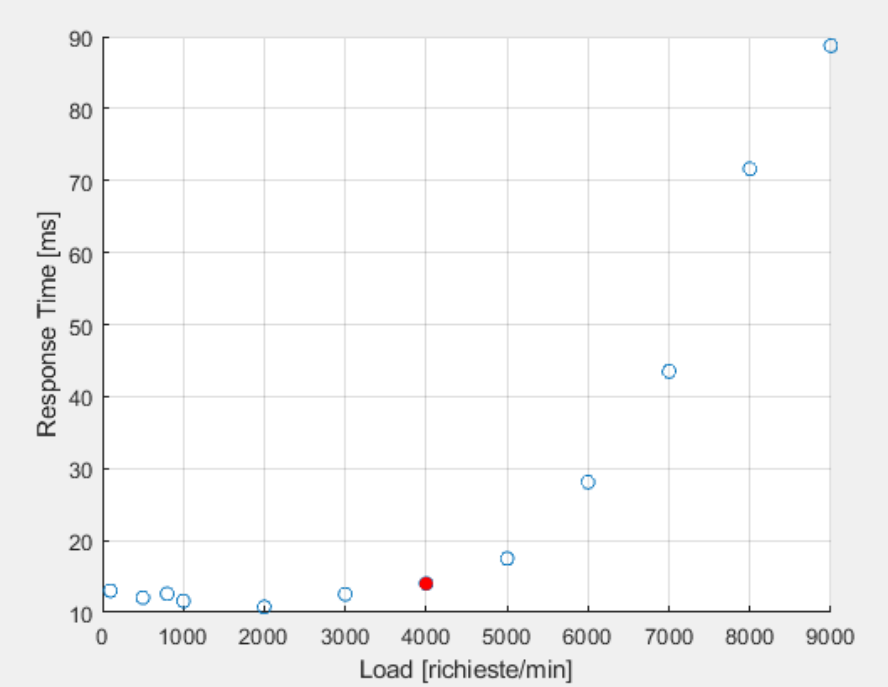

Figura 2.5: *Grafico Response Time*

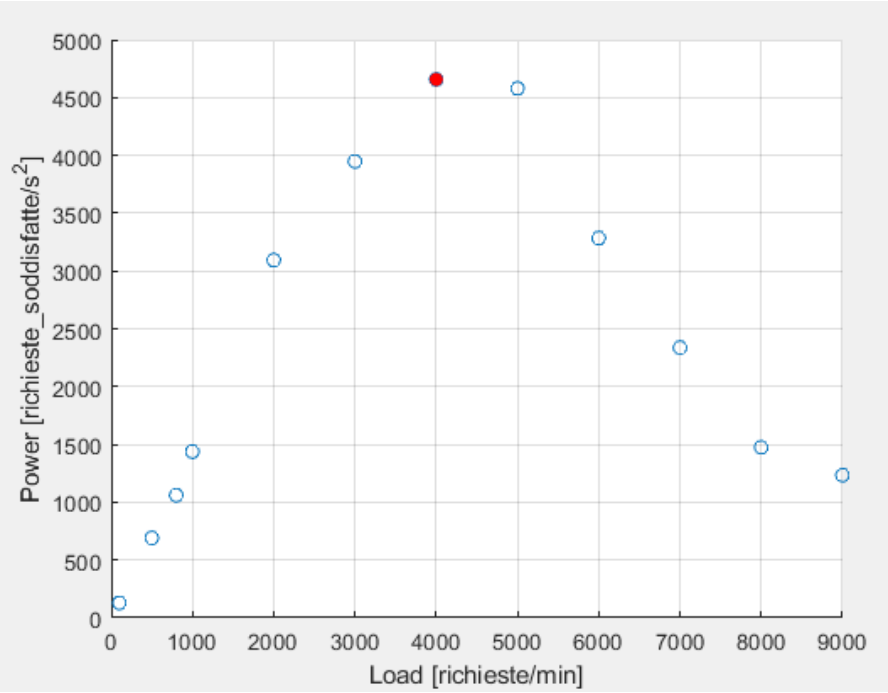

Figura 2.6: *Grafico Potenza*

Come si evince dal grafico della Potenza, il suo punto di massimo è associato ad un carico di 4000 richieste/min.
La *Knee Capacity* (in rosso nel grafico dei Throughputs) è il valore throughput associato a questo carico. Ciò significa che se il nostro server opera in condizioni ottimali riesce a soddisfare circa 65,2112 richieste al secondo, ovvero 3836 richieste al minuto mediamente.

Per quanto riguarda il calcolo della *Usable Capacity*, possiamo relazionarla al tempo di risposta del server quando è sottoposto ad un carico di 6000 richieste/min (il carico limite). Questo response time è pari a 28,11 ms e coincide con quel tempo oltre il quale il sistema inizia a non rispondere più adeguatamente. Pertanto, la Usable Capacity può essere espressa come il valore di throughput associato al carico limite, ovvero 92,3314 richieste al secondo (5431 richieste al minuto circa).
Ovviamente il valore di carico a cui conviene far lavorare il nostro Web Server non è quello massimo che riesce a soddisfare (Usable Capacity). Difatti non sarebbe efficiente per due motivi:

1. I tempi di risposta associato a tale carico sono elevati.
2. Essendo il carico limite, bastano poche richieste in più per ricadere nella zona in cui i tempi di risposta diventano estremamente elevati, rendendo inutilizzabile il server stesso.

Capitolo 3

Web Server - Workload Characterization

L'obiettivo dell'homework è quello di realizzare un workload sintetico semplice e ripetibile, sulla base di un workload reale, attraverso le tecniche descritte nei capitoli precedenti. Successivamente esso deve essere applicato al sistema e, infine, si deve dimostrare che statisticamente si ottiene lo stesso risultato di un workload reale. L'esperimento può essere descritto in tre fasi:

1. *Simulazione di un workload reale.* In questa fase si simulano delle richieste random con carico prefissato al sistema. Si collezionano quindi i dati del client (lista delle richieste) di "alto livello" e i dati del server (memoria, utilizzo della CPU, ecc.) di "basso livello". Alla fine di questa fase bisogna analizzare i dati di alto livello per costruire un workload sintetico.

2. *Applicazione di un workload sintetico.* Dopo aver ricavato il workload sintetico al termine della fase precedente, esso deve essere applicato al sistema. Nuovamente quindi devono essere collezionati i dati di alto livello e basso livello.

3. *Validazione dei dati.* Prevede un'analisi approfondita dei dati di basso livello del workload reale e sintetico. Essi devono essere oppurtunamente caratterizzati per essere infine confrontati statisticamente. Per farlo si utilizzano dei test statistici meglio descritti successivamente.

Le tre fasi possono essere rappresentate graficamente come nella successiva figura.

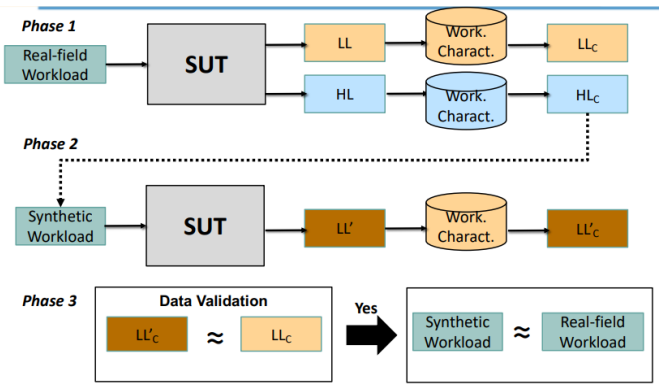

Figura 3.1: *Overview WL Characterization*

Il server deve essere configurato allo stesso in tutte le fasi. In particolare si è utilizzato lo stesso Web Server descritto nel *Capitolo 2* ma con una diminuzione di prestazioni solo a scopo didattico.

3.1 Real-field Workload

Il primo step prevede di applicare o simulare un workload reale. I dati di alto e basso livello devono essere presi contemporaneamente nel client e nel server.

3.1.1 Server

Il server contiene 10 file di diversa dimensione. I file sono stati scelti tutti dello stesso tipo (file di testo) solo per dimensionarli a piacimento. Nulla vieta però di utilizzare file di tipologie diverse (immagini, documenti, audio, etc.).
Essi sono stati dimensionati differenziando i file tra loro di 50 KB e partendo da un minimo di 50 KB. Quindi:

- 50k.txt - File di 50 KB
- 100k.txt - File di 100 KB
- 150k.txt - File di 150 KB
- 200k.txt - File di 200 KB
- 250k.txt - File di 250 KB
- 300k.txt - File di 300 KB
- 350k.txt - File di 350 KB
- 400k.txt - File di 400 KB
- 450k.txt - File di 450 KB
- 500k.txt - File di 500 KB

Parametri di basso livello

Il web server è una macchina virtuale linux. Esistono dunque molti tool in grado di collezionare i parametri caratteristici del sistema. In questo caso è stato utilizzato il tool **vmstat** (**Virtual Memory STATistics reporter**) il quale fornisce informazioni circa le performance del sistema su cui viene eseguito. In particolare esse riguardano:

- **Processi**: numero processi in esecuzione o in attesa di essere eseguiti.
- **Memoria**: memoria libera, swap, buffer etc. Ad esempio
    1. *free*, quantità di memoria libera.
- **Input/Output**: blocchi ricevuti o inviati da/verso un dispositivo a blocchi.
- **Sistema**: parametri di sistema come ad esempio:
    1. *in*, numero di interruzioni al secondo.

2. *cs*, numero di cambi di contesto al secondo.

- **CPU**: tipologia di istruzioni che esegue la CPU etc. Ad esempio:
  1. *us*, tempo trascorso dalla CPU nell'eseguire codice non-kernel
  2. *sy*, tempo trascorso dalla CPU nell'eseguire codice kernel
  3. *id*, tempo trascorso dalla CPU nello stato di idle

Tali informazioni sono tutte utili ai fini dell'esperimento, ma quelle che evidenziano maggiormente gli effetti della nostra analisi sono quelle relative alla CPU e all'I/O.
Vmstat offre inoltre la funzionalità di eseguire un campionamento dei parametri ad una frequenza e durata prefissata, tramite l'apposito comando eseguibile da terminale:

```
vmstat -n 1 400
```

Il primo parametro "1" indica il periodo di campionamento in secondi, mentre il secondo parametro "400" indica la durata totale di esecuzione in secondi. L'output poi può essere semplicemente salvato in un file di testo o csv.
Tale comando è stato avviato pochi secondi prima dell'avvio del test su JMeter, ed ha continuato a campionare per qualche secondo anche dopo la sua fine, dunque nell'analisi dei dati sono attesi parametri il cui andamento evidenzia le varie fasi.

3.1.2 Client - JMeter

La simulazione degli utenti che fanno accesso al server è stata possibile tramite il tool JMeter, già descritto nei capitoli precedenti.
In particolare sono stati realizzati tre Thread Group ognuno composto da 10 Thread (l'equivalente di 10 utenti) con una durata di simulazione pari a 2 min ciascuno. Ogni gruppo contiene 10 richieste riferite alle 10 risorse disponibili nel web server. Esse vengono poi eseguite in modo casuale tramite un apposito controller.

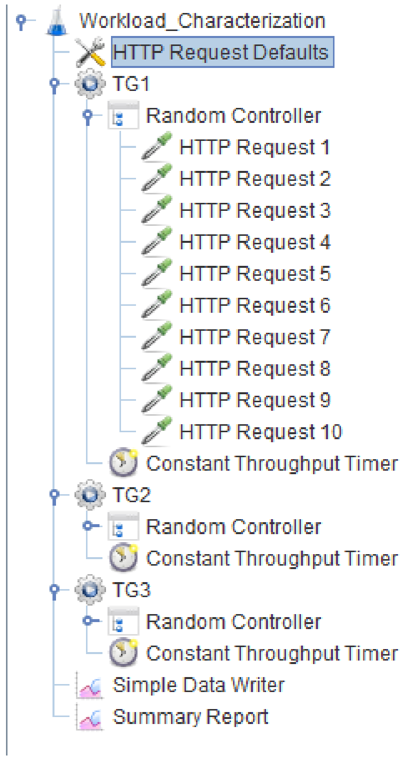

Figura 3.2: *Configurazione di JMeter per la simulazione di un workload reale*

Ogni gruppo inoltre ha un suo specifico carico. In questo caso si ha:

- **TG1** : 400 richieste al minuto
- **TG2** : 550 richieste al minuto
- **TG3** : 700 richieste al minuto

Essi vengono poi eseguiti in sequenza e non in parallelo. Questo perché si possono creare possibili conflitti sulle risorse, dati dal fatto che ogni gruppo richiede le stesse risorse degli altri.

Parametri di alto livello

I parametri di alto livello possono essere collezionati direttamente tramite il tool JMeter e salvati in formato *.cvs*. Non sono necessari dunque programmi esterni. I parametri utili ai fini dell'analisi sono:

- **Timestamp**, l'istante di tempo in cui viene effettuata la corrispettiva richiesta (in millisecondi)
- **elapse**, inteso come Response Time
- **label**, contiene l'informazione categorica della richiesta effettuata.
- **bytes**, numero di byte ricevuti tramite la relativa richiesta.
- **sentBytes**, numero di byte inviati per effettuare la richiesta.

- **latency**
- **connect**, tempo di connessione misurato per effettuare l'handshake TCP (in millisecondi).

3.1.3 Workload Characterization

Le misure vengono effettuate correttamente avviando prima *vmstat* nel server e in seguito JMeter sul client in modo che i dati vengono salvati contemporaneamente nel lato client (alto livello) e nel lato server (basso livello). Al termine della simulazione quindi ci si ritrovano due file di dati.

Parametri di alto livello

I parametri di alto livello vanno incontro alla procedura di filtraggio, PCA e Clustering per ridurne la dimensionalità.
Essi appaiono nel seguente modo:

	timeStamp	elapsed	label	bytes	sentBytes	Latency	Connect
1	1,638814e+12	7	HTTP Request 1	51512	123	6	3
2	1,638814e+12	27	HTTP Request 10	512313	124	2	0
3	1,638814e+12	8	HTTP Request 4	205113	124	2	0
4	1,638814e+12	6	HTTP Request 3	153913	124	2	0
5	1,638814e+12	11	HTTP Request 5	256313	124	2	0
6	1,638814e+12	4	HTTP Request 2	102713	124	2	0
7	1,638814e+12	5	HTTP Request 3	153913	124	2	0
8	1,638814e+12	5	HTTP Request 4	205113	124	1	0

Figura 3.3: *Parametri di alto livello utili ai fini dell'analisi*

La fase di filtraggio non prevede nessuna azione di modifica del dataset.
Sul dataset originale quindi deve essere effettuata la PCA per cercare di ridurne la dimensionalità senza perdere troppa varianza. Bisogna soprattutto considerare che per questi parametri la fase di Clustering è molto importante poiché racchiude le informazioni principali per costruire il workload sintetico.

Tramite la PCA sono state scelte tutte le componenti principali, mantenendo una varianza del 100%.

Autovalori

Numero	Autovalore	Percentuale	20 40 60 80	Percentuale cumulativa
1	1,7076	28,459		28,459
2	1,4831	24,718		53,177
3	0,9992	16,653		69,830
4	0,9836	16,393		86,224
5	0,4669	7,782		94,006
6	0,3597	5,994		100,000

Figura 3.4: *Analisi della varianza tramite autovalori*

Sulla base di queste può essere effettuato il clustering gerarchico.

Il numero di cluster rappresenta il numero di richieste del workload sintetico, poiché in ogni cluster viene scelto un elemento rappresentativo di esso stesso. A tal proposito quindi il numero di cluster non deve essere maggiore del numero di *HTTP Request* totali utilizzate durante la simulazione (in questo caso 30) e non deve essere un numero molto elevato. Al tempo stesso però non si deve perdere molta varianza a causa della clusterizzazione.
La via più semplice è quella di effettuare delle prove scegliendo un numero di cluster minore della metà (in questo caso minore di 15) e valutare per ogni numero quanta varianza si perde.
Partendo da 6 componenti principali si può scegliere un numero di cluster variabile e calcolare la devianza persa per ogni valore.

6 Cluster	8 Cluster	10 Cluster	12 Cluster
35%	27%	21.5%	17%

La scelta ricade su 10 cluster in modo da avere un workload sintetico abbastanza ristretto e una perdita di varianza non troppo elevata.
Per scegliere gli elementi rappresentativi di un cluster si può ricorrere a vari metodi:

- Il punto più centrale possibile
- Il punto in cui un valore categorico si ripete più volte. Applicabile però solo se il dataset ha un parametro categorico in un insieme limitato di valori.
- Casualmente
- Il punto che si avvicina il più possibile alla media del cluster
- Etc.

Parametri di basso livello

	r	b	swpd	free	buff	cache	si	so	bi	bo	in	cs	us	sy	id	wa	st
1	0	0	0	1394332	44704	946952	0	0	6025	234	660	2998	51	20	23	5	0
2	0	0	0	1394324	44704	946952	0	0	0	0	494	236	8	1	91	0	0
3	0	0	0	1394324	44704	946952	0	0	0	0	468	221	12	1	87	0	0
4	1	0	0	1394324	44704	946952	0	0	0	0	494	389	8	0	92	0	0
5	0	0	0	1394324	44704	946952	0	0	0	0	478	202	9	1	90	0	0
6	0	0	0	1394324	44704	946952	0	0	0	0	472	237	12	1	87	0	0
7	0	0	0	1394324	44704	946952	0	0	0	0	503	370	10	1	89	0	0
8	0	0	0	1394308	44704	946952	0	0	0	0	522	488	20	2	78	0	0
9	0	0	0	1394308	44704	946956	0	0	4	0	457	249	13	1	86	0	0
10	0	0	0	1394308	44704	946956	0	0	0	0	462	249	15	1	84	0	0
11	0	0	0	1394016	44712	947000	0	0	64	0	482	251	11	1	88	0	0
12	0	0	0	1393760	44712	948188	0	0	1208	0	632	310	20	6	73	1	0
13	0	0	0	1393032	44712	948628	0	0	400	0	612	263	13	5	82	0	0
14	0	0	0	1392044	44712	949380	0	0	752	4	741	308	22	10	68	0	0
15	0	0	0	1391664	44712	949776	0	0	352	8	711	280	13	8	79	0	0
16	0	0	0	1391696	44712	949780	0	0	0	0	674	292	18	7	75	0	0
17	0	0	0	1391696	44720	949780	0	0	0	52	582	318	20	5	75	0	0
18	0	0	0	1391696	44720	949780	0	0	0	0	699	265	11	6	83	0	0
19	0	0	0	1391696	44720	949780	0	0	0	0	635	262	17	4	79	0	0
20	0	0	0	1391696	44720	949780	0	0	0	0	660	295	19	7	74	0	0
21	0	0	0	1391696	44720	949784	0	0	0	0	665	286	13	7	80	0	0
22	0	0	0	1391696	44728	949784	0	0	0	12	655	306	19	6	75	0	0
23	0	0	0	1391632	44728	949784	0	0	0	0	684	312	13	7	79	0	0
24	0	0	0	1391316	44872	949792	0	0	144	0	716	289	11	7	81	1	0
25	0	0	0	1391156	44872	949792	0	0	0	0	671	296	18	8	74	0	0
26	0	0	0	1391092	44872	949796	0	0	0	0	697	303	12	7	80	0	0
27	0	0	0	1390996	44880	949788	0	0	0	12	633	284	17	7	76	0	0
28	0	0	0	1390836	44880	949796	0	0	0	0	658	293	16	6	78	0	0
29	0	0	0	1390772	44880	949796	0	0	0	0	740	280	14	7	79	0	0
30	0	0	0	1390676	44880	949800	0	0	0	0	697	272	17	8	75	0	0
31	2	0	0	1390388	44880	949800	0	0	0	0	620	314	15	5	80	0	0

Figura 3.5: *Porzione dataset - Low Level Parameters*

I parametri di basso livello costituiscono un dataset formato da 17 colonne e 400 righe. Su di esso sono state dunque effettuate operazioni di *filtraggio*, *PCA* e *clustering*, procedendo in maniera analoga a quanto già si era fatto nel Capitolo 1.

Analizzando le distribuzioni sono state individuate 4 colonne costanti (e quindi da rimuovere): **swpd**, **si**, **so**, **st**.
Dopodichè sono state eliminate tutte le righe associate ai campioni prelevati da vmstat quando il client non stava sottoponendo richieste al server, in quanto non forniscono informazioni utili agli scopi della caratterizzazione.
Per fare ciò sono stati analizzati gli andamenti in funzione del tempo dei parametri relativi all'utilizzo della CPU precedentemente descritti.

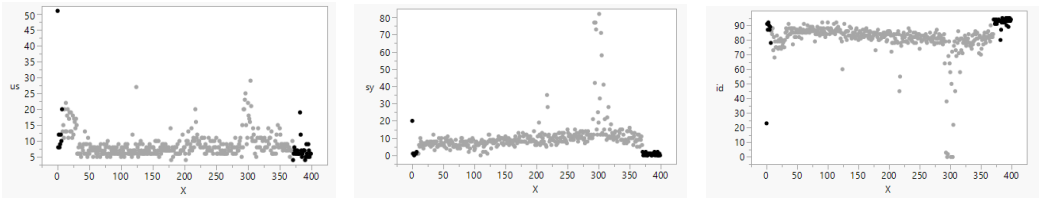

Figura 3.6: *Andamento parametri CPU*

In particolare, osservando i grafici di *sy* e *id* risultano evidenti queste "fasi" nelle quali il processore trascorre meno tempo ad eseguire codice kernel e più tempo in idle rispetto a quando si sta sottoponendo il workload, visto che non è impegnato nel servire le richieste.
Discorso analogo lo si può fare osservando le interruzioni al secondo (**in**), che incrementano drasticamente nel corso della recezione delle richieste, per poi diminuire in queste fasi.

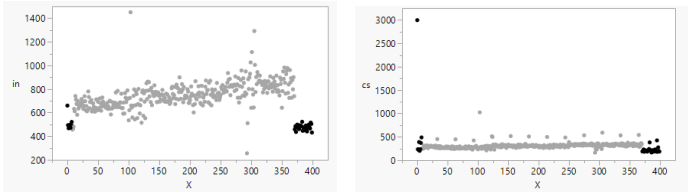

Figura 3.7: *Andamento parametri di Sistema e CPU*

Contestualmente è stata eliminata anche la prima riga (outlier per quasi tutti i parametri), in quando è stata associata alla fase di avvio dell'esecuzione del comando vmstat e quindi non di particolare interesse per gli scopi dell'esperimento.
L'ultima operazione effettuata in questa prima fase di filtraggio è stata la rimozione di un outlier associato al parametro **b**, il quale è indice del numero di processi sospesi ed in attesa di risorse per poter essere riattivati. Essendo outlier anche di **wa** (tempo trascorso dalla CPU in attesa di input/output) è quasi sicuramente indice dello stesso fenomeno, il quale è stato da noi considerato come casuale e quindi trascurabile.

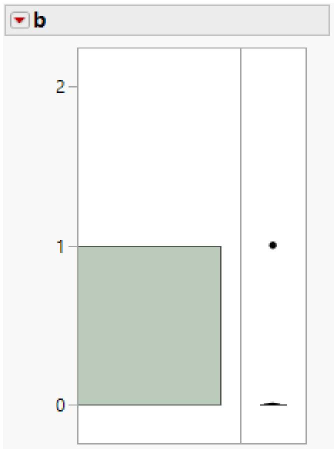

Figura 3.8: *Distribuzione di b*

La riga ad esso associata è la 122, la quale è stata opportunamente rimossa. A seguito di questa cancellazione, la colonna associata al parametro b è diventata costante ed è dunque stata eliminata.
Il dataset ottenuto è formato da 12 colonne e circa 360 righe. Il numero di righe risultante può essere ulteriormente validato se consideriamo che la durata del test è proprio di 360 secondi.

Su questi dati è stata effettuata la PCA, a seguito della quale sono state prese in considerazione *7 componenti principali*, le quali spiegano il 92,768 % della devianza totale.

Numero	Autovalore	Percentuale	20 40 60 80	Percentuale cumulativa
1	3,8516	32,096		32,096
2	2,6266	21,888		53,985
3	1,4119	11,766		65,751
4	1,1762	9,802		75,552
5	0,8663	7,219		82,772
6	0,6281	5,235		88,006
7	0,5713	4,761		92,768
8	0,4057	3,381		96,149
9	0,2423	2,019		98,168
10	0,1464	1,220		99,388
11	0,0726	0,605		99,992
12	0,0009	0,008		100,000

Figura 3.9: *PCA Low-Level Filtered*

Per non perdere una porzione significativa di devianza, dopo vari test, si è deciso di selezionare **20 Cluster** conservandone il 78,36%.
Infine è stata implementata in Matlab la seguente funzione per scegliere l'elemento rappresentativo di ogni cluster:

```
function [new_workload] = random_selection(workload)
%workload = colonne PCA + colonna cluster
N_cluster = max(workload(:,end)); %numero cluster
[r,c] = size(workload);

%isolo la colonna dei cluster
cluster_data = workload(:,end);

new_workload = zeros(N_cluster,c);
```

```
%itero per ogni cluster
for i = 1:N_cluster
%prelevo tutti gli indici di riga associati ad uno stesso cluster i
        index = find(cluster_data==i);
        %se ho più di una riga ne scelgo una random
        if length(index) > 1
                new_workload(i,:) = workload(randsample(index,1),:);
        else
                %se ho solo una riga scelgo quella
                new_workload(i,:) = workload(index,:);
        end
end
end
```

La funzione riceve in input le colonne estratte dalla PCA affiancate alla colonna dei cluster, la quale specifica il cluster di appartenenza di ciascuna riga. Come si evince dal codice, i centroidi sono stati selezionati in maniera randomica nel caso in cui nel cluster sia presente più di un elemento.
L'output della funzione è un workload costituito da 7 colonne e 20 righe, rappresentativo di quello originario.

3.2 Synthetic Workload

A partire dalla clusterizzazione di alto livello, identificati gli elementi rappresentativi di ogni cluster, si deve rifare la simulazione ma con il workload sintetico.
Il dataset in esame è composto da un parametro categorico che rappresenta la richiesta effettuata (risorsa e utente) facendo parte di un insieme molto limitato. La scelta dell'elemento rappresentativo può ricadere nel scegliere la *label* che si ripete in più punti nello stesso cluster.
A tal proposito si può calcolare una tabella che per ogni cluster indica il numero di ricorrenze del valore del parametro categorico.

	Cluster 10	N righe	N(HTTP Request 1)	N(HTTP Request 2)	N(HTTP Request 3)	N(HTTP Request 4)	N(HTTP Request 5)	N(HTTP Request 6)	N(HTTP Re
1	1	51	1	3	0	1	0	2	
2	2	587	0	70	70	90	79	54	
3	3	701	0	0	0	0	0	20	
4	4	977	0	0	0	0	0	0	
5	5	598	0	0	0	0	0	0	
6	6	2	0	0	0	0	0	0	
7	7	39	0	0	0	0	0	0	
8	8	9	0	0	0	0	0	0	
9	9	15	0	0	0	0	0	0	
10	10	325	79	0	0	0	0	0	

Figura 3.10: *Tabella che associa ad ogni cluster il numero di punti con una determinata label*

La matrice che compone la tabella è una matrice 10x30 (10 cluster e 30 richieste possibili) e bisogna calcolare il massimo di riga per ogni riga. Inoltre se il massimo di riga è associato ad una *label* già selezionata in precedenza, allora si deve scegliere il secondo massimo nella riga e così via. Si può automatizzare il tutto tramite uno script MATLAB:

```
%% Dati
[data, txt] = xlsread('label-cluster 10');
```

```
data_filter = data(:, 3:end);
txt_filter = txt(:, 3:end)';

%% Ricerca centroidi
[r,c] = size(data_filter);
max_list = zeros(1,r);  % Lista dei massimi
max_index = zeros(1,r); % Lista degli indici dei massimi

%Inizializzo vettore degli indici
for i=1:r
        max_index(i) = -1;
end

for i=1:r
        [temp_v, temp_i] =  max(data_filter(i,:));
        while ismember(temp_i,max_index)
                data_filter(i,temp_i) = -1;
                [temp_v, temp_i] =  max(data_filter(i,:));
        end
        max_list(i) = temp_v;
        max_index(i) = temp_i;
end

%% Stampa risultati
sort(txt_filter(max_index))
```

Il risultato dello script è la lista delle *label* che identificano il workload sintetico.

1. HTTP Request 1 : TG1 con risorsa *200k.txt*
2. HTTP Request 11 : TG2 con risorsa *50k.txt*
3. HTTP Request 17 : TG2 con risorsa *350k.txt*
4. HTTP Request 21 : TG3 con risorsa *50k.txt*
5. HTTP Request 22 : TG3 con risorsa *100k.txt*
6. HTTP Request 23 : TG3 con risorsa *150k.txt*
7. HTTP Request 25 : TG3 con risorsa *250k.txt*
8. HTTP Request 27 : TG3 con risorsa *350k.txt*
9. HTTP Request 28 : TG3 con risorsa *400k.txt*
10. HTTP Request 29 : TG3 con risorsa *450k.txt*

3.2.1 Server

Il server non deve essere assolutamente modificato. Tutte le configurazioni effettuate per il workload reale rimangono invariate

Parametri di basso livello

I parametri di basso livello devono essere collezionati allo stesso modo utilizzato nel workload reale. Essi devono essere confrontati con i parametri di basso livello salvati durante la simulazione del workload reale. Questa operazione verrà analizzata nel paragrafo *Data Validation*.

3.2.2 Client - JMeter

Il client deve, a questo punto, simulare le nuove richieste applicando il workload sintetico ricavato con l'analisi. Anche in questo caso la configurazione di JMeter non deve essere modificata, tranne che per le richieste.

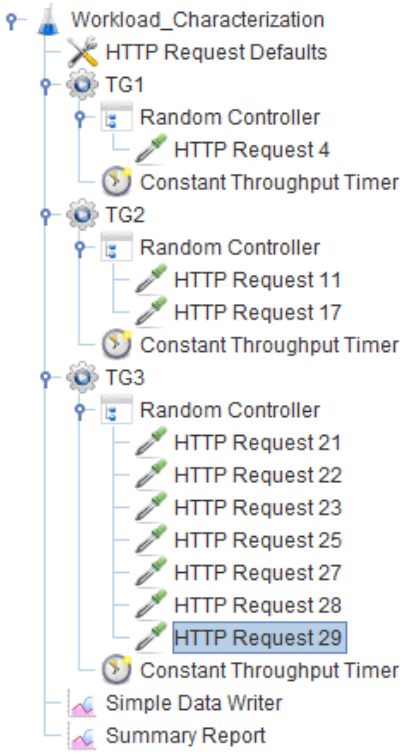

Figura 3.11: *Configurazione di JMeter per il workload sintetico*

Parametri di alto livello

I parametri di alto livello possono anche non essere collezionati poiché non servono per ulteriori analisi. Per completezza però si possono salvare tramite lo stesso JMeter usato per la simulazione.

3.2.3 Workload Characterization

La caratterizzazione può essere applicata solo ai parametri di basso livello. Il risultato viene confrontato con la caratterizzazione degli stessi parametri misurati con il workload reale. Deve necessariamente accadere che il numero di colonne di questi parametri sia lo stesso.

Parametri di basso livello

Anche in questo caso il dataset di basso livello è costituito da 17 colonne e 400 righe. Analogamente a quanto fatto in precedenza per i parametri relativi al Workload reale sono state eseguite le operazioni di filtraggio, PCA e clustering.
Sono state eliminate le colonne costanti, che in questo caso sono: **b**, **swpd**, **si**, **so** e **st**.

Dopodichè, eliminando le righe relative ai campionamenti nelle fasi in cui il sistema non stava servendo alcuna richiesta, il set di dati è stato ulteriormente ridotto. L'analisi degli outlier non ha portato all'eliminazione di alcuna riga, in quanto tutti i punti oltre i quartili dei box-plot sono stati considerati significativi.

Come accaduto anche in precedenza, il dataset risultate è composto da 12 colonne e circa 360 righe (coerentemente con il tempo di sottomissione del workload).

Numero	Autovalore	Percentuale	20 40 60 80	Percentuale cumulativa
1	4,6975	39,146		39,146
2	2,0232	16,860		56,006
3	1,1235	9,362		65,368
4	1,0373	8,644		74,012
5	0,8961	7,468		81,480
6	0,7930	6,608		88,088
7	0,6228	5,190		93,279
8	0,5109	4,258		97,536
9	0,2079	1,732		99,268
10	0,0625	0,521		99,789
11	0,0207	0,172		99,961
12	0,0047	0,039		100,000

Figura 3.12: *PCA syn Low-Level Filtered*

Per la PCA è stato scelto lo stesso numero di componenti selezionate nella caratterizzazione dei parametri di basso livello del workload reale. Questa è stata una scelta obbligata dal fatto che le funzioni Matlab utilizzate per la validazione operano su matrici le quali devono necessariamente avere lo stesso numero di colonne.
In ogni caso selezionando tali componenti si riesce a mantenere un'ottima percentuale di devianza: il 93,279%.

Anche qui la scelta più conveniente in termini di devianza persa è stata quella di suddividere il dataset in 20 Cluster, conservandone il 75,51% della totale.
Infine il workload di basso livello così ottenuto, è stato dato in input alla funzione *random_selection*, già descritta in precedenza, per ricavare i centroidi di ogni cluster.

3.3 Data Validation

I parametri di basso livello relativi a workload reale e sintetico, ottenuti in seguito alla caratterizzazione sono i seguenti:

real =

-7.6467	3.8108	-0.2363	-1.1546	1.8595	0.9876	-1.1104	1
-2.0806	1.2423	0.0243	-0.7371	0.5829	0.2139	-0.6864	2
-3.2323	2.7898	2.1293	-1.2852	-0.7765	-1.3999	1.1639	3
-1.3219	-0.2341	-0.5549	-0.2241	0.1265	0.2619	-0.2032	4
0.0722	-0.6433	-0.3106	-0.2053	-0.4204	-0.1077	0.0387	5
0.7494	-1.9382	-0.8197	0.2383	-1.1200	-0.4296	0.5033	6
4.1076	2.7490	-0.0809	-0.2197	0.3025	-0.2231	-0.9849	7
-2.8183	2.7099	1.4750	0.9798	2.6722	-1.3798	-0.5066	8
-1.7812	0.0375	-0.2656	1.5176	1.5185	-0.0486	1.1666	9
-0.5033	-1.6519	3.3529	6.3848	6.0619	-4.6155	1.9613	10
2.9730	4.3130	-1.4532	1.0525	-0.1025	1.0709	2.2629	11
1.3715	-0.8105	-0.3164	-0.1945	-0.1326	0.5953	0.7807	12
4.8340	7.2083	-3.4839	2.8833	0.1221	2.3996	4.0902	13
1.0610	-0.7975	1.4556	-1.5270	0.8094	1.1570	-0.0215	14
7.7519	4.5686	3.1382	-2.6228	2.7335	1.8081	-1.3047	15
2.9350	-2.8465	8.4219	-5.4299	6.2165	4.6344	0.4080	16
4.9911	8.8931	-3.8761	3.1680	-0.6799	-1.1646	-0.8345	17
-0.1524	-0.1828	1.1103	2.5281	-1.2307	1.8948	-0.0178	18
-0.4422	0.5430	2.8811	4.3395	-3.1234	2.2766	-2.1920	19
-6.8105	7.8404	9.0113	-2.4265	-5.9991	-2.5519	4.3179	20

Figura 3.13: *Low-Level Parameters Real WL*

synthetic =

1.3613	3.8828	-2.3363	-0.5061	0.5182	4.2447	1.0046	1
-2.0107	-0.1983	0.1911	-0.1098	-0.3192	-0.2824	-0.7827	2
1.2592	1.1510	-0.4001	-0.0740	-0.5915	-0.6392	0.5182	3
-2.4652	-0.8116	0.3711	-0.5470	-0.1421	0.7935	0.0496	4
-1.4773	-1.0322	-0.0326	-0.3627	-0.0170	0.1397	-0.3634	5
-1.1208	0.2234	0.8272	-0.0980	-1.0114	-0.1495	2.4649	6
0.1193	-0.3989	-0.3491	-0.5094	-0.2201	0.1424	1.2521	7
4.9240	2.6595	5.4939	1.4931	-1.6292	-1.3502	2.2706	8
1.6783	-0.3867	2.5966	-0.5528	0.8415	0.7942	-0.3666	9
2.0998	-1.8761	-0.1893	-0.2033	0.2341	0.1760	-0.2447	10
3.8096	0.5758	-0.3417	0.3092	-0.4044	-0.8870	-0.5317	11
2.6983	-0.3619	0.1014	0.1586	-0.4047	-0.5110	0.2097	12
-2.5112	2.0493	-0.3773	1.1422	0.0644	-0.5302	0.0175	13
-2.2360	2.5785	-0.7645	2.3690	0.9516	-0.3645	-0.2480	14
-1.9063	0.7859	-0.5217	2.1736	1.2891	0.1960	0.5051	15
-0.4700	0.6165	-0.2866	1.0818	0.3627	-0.1609	0.4268	16
-2.9928	-0.5533	-0.7877	2.9954	2.5809	1.1836	0.9577	17
-0.3187	5.7494	6.7172	-7.4846	11.6770	-4.0430	0.4878	18
3.8729	0.4539	8.2417	0.7729	-0.8239	4.1174	0.2207	19
7.2334	13.1554	-6.6097	-5.0437	1.1857	9.6592	-1.1957	20

Figura 3.14: *Low-Level Parameters Synthetic WL*

Sono questi che andranno statisticamente confrontati per validare il Workload Sintetico.
La procedura per la validazione è molto semplice:

1. Normalità verificata. Se i due dataset provengono da una distribuzione normale allora si possono eseguire test parametrici per la validazione. In particolare si possono usare vari tipi di test statistici anche in base all'omoschedasticità dei campioni.

   (a) Omoschedasticità verificata. Se i due dataset sono omoschedastici allora si possono utilizzare test sulla base di questa condizione verificata.

   (b) Omoschedasticità non verificata. Se i due dataset non rispettano la proprietà di omoschedasticità allora bisogna utilizzare test che si basano su tale condizione non verificata.

2. Nomarlità non verificata. Se i due dataset non provengono da una distribuzione normale allora si devono usare necessariamente test statistici non parametrici per la validazione.

3.3.1 Normalità

Per verificare se un campione proviene da una popolazione con distribuzione normale, ci si può affidare a test visivi oppure a test statistici analitici.

Test Visivo

Visivamente si può capire se un campione proviene da una popolazione con distribuzione normale effettuando un grafico dei quantili del campione rispetto ai quantili di una distribuzione normale. Ciò lo si realizza sfruttando la funzione Matlab **qqplot**.
Ad esempio prendendo una componente principale del workload reale (dopo la caratterizzazione) si può effettuare un grafico dei quantili:

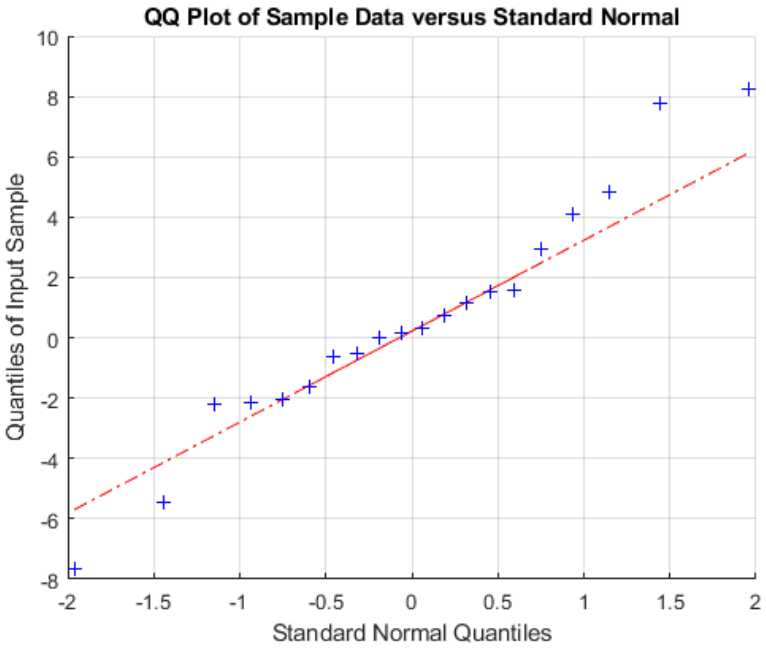

Figura 3.15: *Grafico Quantili-Quantili della prima componente principale del workload reale caratterizzato*

Come si può notare il campione non proviene da una distribuzione normale.

Test analitico

Un test analitico è il test di **Kolmogorov-Smirnov**. Esso si basa sull'ipotesi nulla H_0:

$$H_0 : N(0,1)$$

ovvero che i dati in input provengono da una distribuzione normale standard.
In MATLAB esiste una funzione già definita per eseguire questo tipo di test: **[h,p] = kstest(x)**. Esso fornisce in output il risultato del test (se H_0 viene rigettata o meno) con il relativo *P-Value*. In particolare h è 1 se il test rigetta l'ipotesi nulla H_0 con un livello di significatività del 5%, 0 altrimenti.

Caso di studio - Implementazione

Per la verifica della normalità si sono quindi applicate queste funzioni ai dataset (Fig.3.13 e Fig.3.14).

```
%% Data
real1 = xlsread('real');
synthetic1 = xlsread('syn');

real = random_selection(real1);
synthetic = random_selection(synthetic1);

N = size(real,2); %numero di colonne lo stesso per i due set di dati
real = real(:, 1:N-1); % rimuovo la colonna associata ai cluster
synthetic = synthetic(:, 1:N-1); % lo stesso

%verifica Normal Distribution kstest
[h_ks_real, p_ks_real] = kstest(real);
[h_ks_syn, p_ks_syn] = kstest(synthetic);

%verifica Normal Distribution visual test
figure();
subplot(2,1,1);
qqplot(real);
subplot(2,1,2);
qqplot(synthetic);
```

L'output della kstest è 1 per entrambi i workload, quindi entrambi non provengono da una distribuzione normale. Per completezza si riportano i grafici del quantile-quantile plot associato ad ogni PC dei dataset, i quali confermano il risultato del test di Kolmogorov-Smirnov.

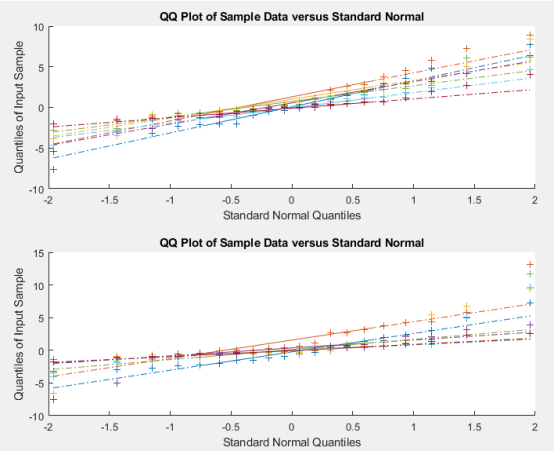

Figura 3.16: *quantile-quantile plot di Fig.3.13 e Fig.3.14*

Ciò suggerisce l'utilizzo di test non parametrici per la validazione.

3.3.2 Omoschedasticità

Anche se nel caso di studio la verifica dell'omoschedasticità non è richiesta, visto che entrambi i set di dati non provengono da una distribuzione normale, per completezza ne riportiamo il procedimento.

Tale proprietà è verificata nel momento in cui diversi campioni hanno stessa varianza (non è detto che essa sia nota), anche se provengono da popolazioni (distribuzioni) differenti. Questa è una caratteristica necessaria da verificare prima di applicare i test parametrici, ad esempio. Difatti, eseguire un test senza aver effettuato questo controllo può avere un impatto significativo sui suoi risultati, fino ad invalidarli.

Test Visivo

Visivamente possiamo capire se le varianze delle due distribuzioni sono simili, tramite il **boxplot**. All'aumentare della lunghezza del box aumenta la varianza dei dati che "rappresenta".

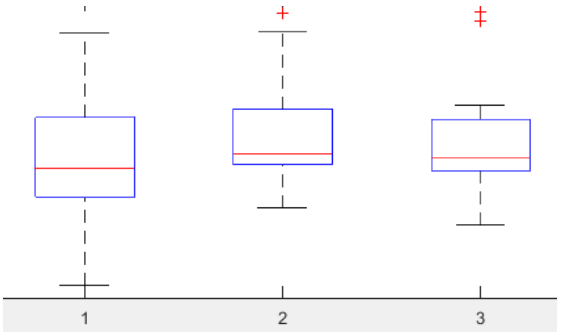

Figura 3.17: *boxplot I,II e III componente del WL reale*

Test analitico

Un test analitico per il check sull'uguaglianza delle varianze dei campioni è:
h = vartest2(x,y).
Esso è applicabile esclusivamente se i due campioni provengono da una distribuzione normale. Si basa sull'ipotesi nulla H_0:

1. H_0 : i vettori x ed y provengono da distribuzioni normali e con uguale varianza.

Il suo output h è 1 se il test rigetta l'ipotesi nulla H_0 con un livello di significatività del 5%, 0 altrimenti.

3.3.3 Validazione

Per completezza è riportato l'intero script, nel quale è stato previsto anche il caso in cui i campioni provengono da distribuzioni normali.

```
%se almeno una delle due distribuzioni non è normale
%applico il test non parametrico
if ((h_ks_real | h_ks_syn) == 1)
[p_wilc,h_wilc] = NoParametric(real,synthetic,N);
else
%distribuzioni normali (risultato di quell if è 0)
%check sulle varianze
[h_var, p_var] = vartest2(synthetic, real);

        %se le due distribuzioni hanno stessa varianza
        if (h_var == 0)
```

```
                %applico il two sample t-test
                [h_ttest, p_ttest] = ttest2(syntetic, real);
                else
                %se le due distribuzioni non hanno stessa varianza
                [h_ttest_novar, p_ttest_novar] = ttest2(syntetic, real,
↪  'Vartype', 'unequal');
                end

        end
```

Come già detto precedentemente, i nostri campioni soddisfano la condizione del primo *if* quindi viene eseguita la funzione *NoParametric*:

```
        function [p_wilc,h_wilc] = NoParametric(real,synthetic,N)
                p_wilc = zeros(1,N-1);
                h_wilc = zeros(1,N-1);
                for i = 1:N-1
                        [p_wilc(i),h_wilc(i)] =
                        ranksum(synthetic(:,i), real(:,i));
                end
        end
```

Essa richiama la funzione MATLAB
[p,h] = ranksum(x,y) la quale esegue il test non parametrico di **Wilcoxon**. Esso si basa sull'ipotesi nulla H_0:

1. H_0 : i dati di x ed y provengono da distribuzioni continue con uguale mediana.

Tale funzione opera sulle singole colonne delle matrici x ed y e quindi bisogna iterare il procedimento per il numero di colonne. Se l'output h è uguale ad 0, si ha un fallimento nel rigettare l'ipotesi nulla con un livello di significatività del 5 %, il contrario se è 1.
L'output di *NoParametric* sarà dunque costituito da due vettori:

1. p_wilc : il vettore degli p-value per ogni componente principale.
2. h_wilc : il risultato della ranksum su ogni componente principale di x ed y.

L' **h_wilc** calcolata sui campioni sotto studio è un vettore di tutti zeri:

```
h_wilc =

     0     0     0     0     0     0     0
```

Figura 3.18: *output test di Wilcoxon*

ciò significa che l'ipotesi nulla non è stata rigettata per ogni componente principale e che quindi Workload Reale e Workload Sintetico hanno determinato effetti statisticamente simili sul server.
Il Workload Sintetico ottenuto è una buona approssimazione di quello Reale.

Capitolo 4

Web Server - Design of Experiment

Obiettivo: Design an experiment to study the impact of two factors (intensity and page type) on the response time

4.1 Design

Descrizione fattori scelti ecc collezionamento response time medio

4.2 Analisi

4.2.1 Importanza - Allocation of Variation

4.2.2 Significatività - Analysis of Variance