Impianti di elaborazione

Anno Accademico
2025/26

Rocco Lo Russo
Agostino D'Amora

Contents

Introduzione

In questo documento verranno raccolti appunti presi a lezione del corso di *Impianti di elaborazione* tenuto nell'anno 2025-26 dai professori Cotroneo e Pietrantuono, con l'aggiunta di alcuni richiami e approfondimenti

Part I

Performance

Chapter 1

Valutazione delle Performance

La valutazione delle performance di un sistema(o system evaluation) è un argomento importante da dover trattare. Negli anni ci sono stati vari problemi ai sistemi che sono stati ideati, poichè o valutati in modo scorretto o progettati male. Lo scopo principale della system evaluation è quallo di misurare le prestazioni di un determinato sistema, in modo che sia anche possibile confrontare i parametri con quelli di valutazione di altri sistemi (nella maniera più oggettiva possibile).

1.1 System Evaluation

Per la system evaluation, quindi, è importante impostare e delineare un metodo formale di valutazione. Ciò ci permette di ridurre gli errori legati a particolari operazioni e di poter definire una serie di "passi" da seguire per effettuare una corretta valutazione delle performance. In linea formale, la system evaluation si divide un due principali categorie:

- **Performance Analysis**: Tale valutazione presuppone che il sistema non possa avere alcun tipo di fallimento (failure-free). Il che va a valutare solo le performance legate al suo funzionamento. (Bisogna stare attenti quando si effettua Performance Analysis di non andare a valutare in alcun modo i casi di fallimento)
- **Dependability Analysis**: Tale valutazione ci permette di valutare per quanto tempo il sistema sia in grado di funzionare e quindi anche il caso di problematiche ed errori. (tra le analisi di dependability rientra anche la **reiability**, che definisce un parametro per identificare il tempo medio in cui il sistema fallisce [System Mean Time To Failure]

*Un esempio pratico per capire i concetti di **Performance Analysis** e **Dependability Analysis** è quello di un auto di formula 1. Nel caso della Performance Analysis vado solo a valutare le specifiche performance (velocità massima, tenuta in curva, aerodinamica), senza tener conto in alcun modo di qualunque tipo di fallimento; mentre nel caso della Dependability Analysis si va a valutare quanto la macchina riesca a resistere in pista (durata delle gomme, tempo effettivo di funzionamento, casi di guasti imprevisti).*

1.1.1 Passi per la valutazione di un sistema

Come introdotto precedentemente, per la valutazione "corretta" (o di buona qualità) di un sistema, è ottimale definire una serie di passi da seguire, in modo da redere il criterio

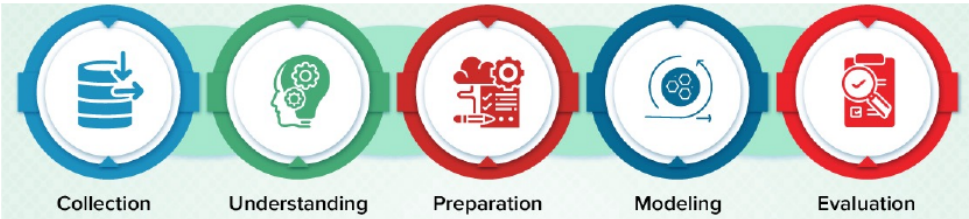


Figure 1.1: Passi da effettuare per la CM e la CP

di valutazione il quanto più formale possibile. I passi per valutare un sistema sono i seguenti:

1. Definire cosa bisogna valutare
2. Ottenere informazioni sul sistema
3. Effettuare le misurazioni
4. Analisi dei risultati
5. Trovare e valutare il corretto feedback da dare sulle considerazioni iniziali

1.1.2 Performance Analysis

Nell'analisi delle performance, quindi, si va a presupporre che il sistema di base funzioni e che quindi bisogni valutare solo il "come" tale sistema funziona (failure-free). L'analisi delle performance può essere suddivisa formalmente in due principali parti:

- **Capacity Management**: Fa in modo che le odiene risorse disponibili diano le migliori performance possibili (quindi si basa solo sulla valutazione del caso presente senza aver fatto alcuna supposizione sul carico fututo)
- **Capacity Planning**: Si assicura che ci sia un allocazione delle risorse in base ai workload successivi (si va a fare delle previsioni sul possibile carico futuro)

Nella spiegazione dei termini precedenti si è fatto riferimento ad una parola, ovvero ***workload****. Il significato e la definizione vera e formale di workload sarà data nei capitoli successivi, per il momento possiamo definire workload come: richieste effettuate da utenti verso il sistema*

La Capacity Management e la Capacity Planning sono strutturate principalmente in 5 passi che portano alla corretta esecuzione della propria valutazione [1.1]

1.1.3 Errori comuni nella System Evaluation

Quando si fa systema evaluation, solitamente, si possono commettere degli errori, che possono portare poi ad avere uno scorretto parametro di confronto o giudizio del sistema. Gli errori che principalmente vengono fatti sono:

- **Nessun obiettivo**: senza obiettivi chiari non esiste un modello universale; gli obiettivi determinano tecniche, metriche e workload da usare, e non sono mai banali.

- **Obiettivi distorti**: porsi come obiettivo. Dimostrare che il nostro sistema è migliore di un altro porta a una valutazione di parte, in cui gli analisti fanno da giudici invece che da osservatori imparziali.
- **Approccio non sistematico**: condurre la valutazione senza un metodo strutturato (obiettivi → metriche → workload → esperimenti → analisi) porta a risultati incompleti o non riproducibili.
- **Analisi senza capire il problema**: raccogliere dati e produrre grafici senza aver compreso a fondo la natura del problema significa ottenere informazioni non utili alle decisioni.
- **Metriche di performance scorrette**: scegliere metriche che non riflettono gli aspetti importanti del sistema (es. guardare solo il throughput quando è cruciale la latenza) porta a conclusioni fuorvianti.
- **Workload non rappresentativo**: utilizzare un carico artificiale che non riproduce il comportamento reale degli utenti (picchi, mix di richieste, distribuzioni) rende i risultati poco affidabili.
- **Tecnica di valutazione sbagliata**: adottare un metodo inadeguato (analisi analitica, simulazione o misurazioni reali) rispetto agli obiettivi porta a risultati poco significativi o addirittura falsati.

Dati tali errori di valutazione si comprende il motivo a cui è legato il bisogno di definire un path formale per la performance evaluation. Pertanto si va a definire una serie di passi sistematici che ci spiega come poter realizzare la syustem evaluation senza andare in contro alle problematiche descritte in precedenza. I passi da seguire sono i seguenti:

1. Definire gli obiettivi e descrivere il sistema
2. Elencare i servizi e i risultati attesi
3. Selezionare le metriche
4. Elencare i parametri
5. Selezionare i fattori da studiare
6. Scegliere la tecnica di valutazione
7. Selezionare il workload
8. Progettare gli esperimenti
9. Analizzare e interpretare i dati
10. Presentare i risultati
11. Ripetere il processo

Guardando tali passi si comprende che bisogna effettuare alcune scelte fondamentali. Le scelte che principalmente bisogna effettuare sono legate a: Tecniche di valutazione, Metriche di performance e Performance richieste

Tecniche di valutazione

Per valutare le performance di un sistema si possono utilizzare diverse tecniche che racchiudono una metodologia differente di approccio rispetto al sistema, che ci permette di poter valutare le prestazioni prescindendo dal sistema stesso (o in parte). Le tecniche principali di valutazione sono:

- **Modellazione Analitica**: Si va a ricostruire il sistema mediante un modello matematico. Tale tecnica permette di avere una soluzione in forma chiusa (utilizza formule matematiche senza dover simulare o replicare un sistema). Tali sistemi, però, fanno delle assunzioni sul sistema, che permettono la semplificazione e la modellazione matematica
- **Simulazione**: Tale tecnica cerca di combinare la modellazione analitica del sistema con il mondo reale, cercando di emulare quanto più è possibile il caso reale tramite particolati software. È una buona soluzione, poichè richiede un costo intermedio per essere effettuata; se non fosse per il tempo che bisogna dedicargli per la ricostruzione del sistema
- **Misura**: Si vanno a valutare le performance con misurazioni sul sistema reale. Tale approccio è il più costoso, sia in termini di carico che di costo, ma è il più efficiente poichè si è a contatto con il caso reale effettivo

Per selezionare la tecnica adatta al nostro caso bisogna fare una valutazione completa del sistema cercando quella che è la tecnica più adatta in base al nostro criterio di valutazione richiesto. Talvolta può essere anche possibile utilizzare più tecniche insieme. (Un esempio potrebbero essere i Twin Systems, dove vado ad effettuare modifiche prima in simulazione, e se noto miglioramento delle performance applico le stesse scelte anche al sistema reale andando ad analizzare ulteriormente le performance).

Selezione della metrica

Altro parametro importante da scegliere è la metrica da utilizzare. È importante capire il corretto modo di scegliere una metrica dato che è il concetto su cui si basa il confronto tra vari sistemi. Le metriche possono basarsi su diversi criteri, i principali possono essere classificati come:

- **Metriche per le performance**: Si vanno a valutare dei parametri di valutazione discreti (non probabilistici), dipendenti da: Tempo, Processing Rate, Consumo di risorse ecc.
- **Metriche per la Dependability**: Si va a valutare un sistema in base alla sua "efficenza", vista nel senso di probabilità di diversi eventi (guasti, eccezioni ecc.), esempi di tali metriche sono: Availability, Performability, Reliability ecc.
- **Metriche per i costi**: Si basano i criteri sui costi impiegati per l'implementazione di particolari sistemi

Uno dei criteri di selezione della metrica è quello presentato nell immagine [1.2], che ci permette di capire, in base a come reagisci il sistema, quale tipologia e quale classe di parametri andare a considerare. Fare attenzione all'immagine, essa suddivide le possibili

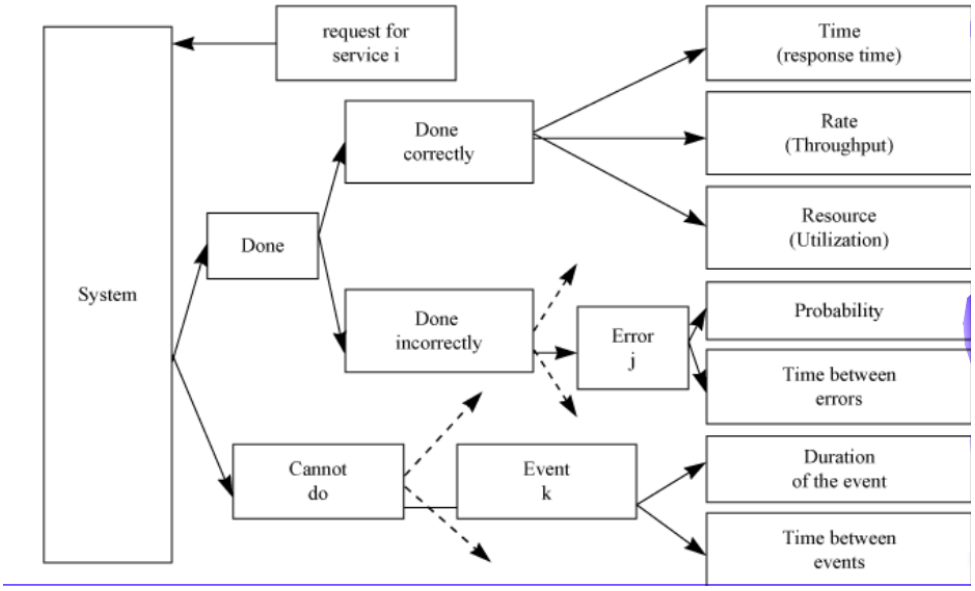


Figure 1.2: Criterio di selezione della metrica

metriche in tre macroaree, la prima è quella riguardanti le metriche deterministiche, ovvero quelle metriche che vengono valutate su casi effettivi e risultati non di natura probabilistica; a differenza delle altre 2 categorie che, avendo a che fare con errori ecc., ricadono nei casi di dover andare a valutare il sistema mediante dei valori di natura probabilistica.

Le metriche presentate, quindi, sono le seguenti:

- **Response Time**: Tale parametro ci permette di capire ogni quanto di tempo un sistema produce un risultato valido. Per la valutazione di tale parametro, però, possono essere effettuate varie tipologie di valutazione:

  - **Response time**: Misurazione basata sul tempo tra l'inizio e la fine della richiesta
  - **Reaction time:** Fine della richiesta dall'inizio del suo processamento
  - **Turnaround time:** Inizio della richiesta fino alla fine della risposta

  Generalmente la Response Time cresce all'aumentare del carico sul sistema, per tale caso è stato definito quello che verrà chiamato **Strech Factor** (permette di capire quando non saranno più usabili un certo numero di risorse)

- **Processing Rate**: Il processing rate non rappresenta altro che il throughput associato al mio sistema, e che quindi calcola la quantità di lavoro svolto da un singolo componente per unitò di tempo.

Talvolta, utilizzare sia il thoroughput che il response time può risultare ridondante, pertanto si decide di utilizzare un unico parametro, dipendente da entrambi, chiamato **potenza**, che si può calcolare come $\frac{Throughput}{ResponseTime}$. Pertanto è giusto andare a ridefinire anche i diversi punti di evoluzione della potenza, che racciudono la nostra attenzione

- **Capacità nominale:** Rappresenta il throughput massimo raggiungibile in condizioni di carico di lavoro ideali. Tuttavia, a questo livello di throughput, il tempo di risposta è generalmente troppo elevato.
- **Capacità utilizzabile:** È il throughput massimo che si può ottenere, quindi è il punto massimo dopo il quale il sistema potrebbe andare in crash (quindi ad esempio ha tempi di risposta molto lunghi ma riesce a gestire molto più carico)
- **Capacità di "ginocchio":** È il punto operativo ottimale, considerato il miglior equilibrio tra un alto throughput e un basso tempo di risposta. Questo punto corrisponde al valore massimo della metrica Power.

Oltre la potenza un ulteriore parametro utile è la **fairness**, che ci permette di capire se un aprticolare sistema distribuisce bene il carico o meno. Ciò lo veniamo a scoprire mediante il calcolo del preciso valore di fairness che è normalizzato, e quindi compreso tra 0 ed 1.

A livello formale, si definisce:

- x_i, frazione del throughput associato ad x_i
- n, numero di utenti nel sistema

A questo punto comprendo se sto usando il throughput in maniera fair, calcolando la **fairness**, che mi dice che se ogni utente ha a disposizione una porzione eguale di throughput, allora sarà 1, mentre nel caso opposto sarà vicino allo 0. Per calcolare la fairness si utilizza la seguente formula:

$$fairness = \frac{\left(\sum_{i=1}^{n} x_i\right)^2}{n \sum_{i=1}^{n} x_i^2}$$

Con tale formula si comprende che se tutti i throughput x_i sono uguali allora => $fairness = 1$.

Chapter 2

Workload

Un Workload, nella sua definizione di base, è: *Tutti i possibili input che un sistema riceve in un certo periodo di tempo*
Tale definizione, pertanto, prescinde dalla sola valutazione delle performance. Nel caso specifico i workload che sono utilizzati nella valutazione delle performance sono detti **test workload**, tali test workload possono essere generati e categorizzati secondo particolari tecniche di utilizzo e schematizzazione

2.1 Test Workload

I test Workload sono dei normali workload utilizzati per gli studi delle performance. In generale, i test workload possono essere classificati in due particolari categorie:

- **Workload Reali**: Workload che sono caratterizzati dall'osservazione di specifici casi reali
- **Workload Sintetici**: Workload che sono caratterizzati da pochi parametri ma che cercano di replicare quelli che potrebbero essere dei workload reali

Le due classificazioni differiscono fortemente sotto molti punti di vista. Il principale tipo di workload utilizzato è quello sintetico, dato che viene caratterizzato tramite pochi parametri e può essere replicato senza dover tenere memoria di un workload reale (che richiederebbe memorie molto ingenti per essere memorizzato).

2.1.1 Addition Istruction

L'**Addition Istruction** è una tecnica di workload sintetico per i computer systems. Essa è stata utilizzata in passato per la valutazione delle performance sui vari computer e comprendeva quello di valutare la velocità e l'efficienza dell'operazione di addizione (Operazione più usata all'interno dei programmi passati).

2.1.2 Instruction Mixes

L'**Istruction Mixes** è un evoluzione dell'Addition Istruction, poichè non va a considerare solo l'operazione di addizione intera, ma anche tutte le altre istruzioni possibili. Per avere uno schema più adatto alla sintetizzazione di un workload si hanno delle specifiche di

Istruction Mixes, che non sono altro che tabelle che listano le varie **classi di istruzioni** con la loro "percentuale" di utilizzo (frequenza). In questo modo sarà possibile, andare a selezionare a caso le istruzioni, rispettando la distribuzione descritta dall'Istruction Mixes.

Disadvantages

La complessità sempre crescente delle architetture e quindi delle classi di istruzioni, non viene riflessa all'interno delle tabelle di istruction mixes, pertento risulta difficile valutare completamente la totalità dell'architettura. Oltre alla complessità delle classi, incidono anche i tempi di esecuzione, che è altamente variabile e dipendente da diversi parametri quali:

- **Modalità di indirizzamento**: Come le modalità di indirizzamento diretto o indiretto di un architettura
- **Cache Hit**: Probabilità di trovare il dato su cui si vuole lavorare in Cache
- **Pipeline Efficiency**: Se la pipeline mantiene per molto tempo l'esecuzione di un comando per ciclo di clock (ad esempio gestendo bene la questione dei salti e della branch prediction)
- **Interferenza dei dispositivi esterni durante i cicli di accesso processore-memoria**: Ad esempio concorrenza dei bus mentre si accede alla memoria per il prelievo di un dato

Oltre a problematiche di tipo puramente architetturale e strutturale, il tempo di esecuzione può variare anche in base alla forma e alle operazioni che devono essere fatte sui dati, come operazioni del tipo:

- La frequenza con cui compare lo zero come parametro
- Quante volte compare lo zero in operazioni di moltiplicazione
- il numero medio di spostamenti richiesti in un'operazione in virgola mobile
- numero di volte in cui un ramo condizionale viene eseguito

Oltretutto, le combinazioni di istruzioni delle istruction mixes non riflettono le funzionalità di indirizzamento virtuale della memoria

Considerazioni

Nonostante le varie problematiche che porta con se, l'istruction mixes, ci permette comunque di poter avere un singolo parametro di confronto tra sistemi diversi. Il parametro è un numero che esprime l'inverso del tempo di esecuzione e può essere espresso come:

- **MIPS (Millioni di Istruzioni Per Secondo)**
- **MFLOPS (Millioni di FlOaring Point istructionS)**

Però un'altro problema legato a questo valore è che stima le prestazioni solo del **processore** e quindi **non dell'intero sistema**. Il divario tra le performance reali e non dei sistemi che vengono valutati con tali modelli è fatto, quindi, solo dalla differenza dei programmi che vengono eseguiti, e quindi non è una statistica affidabile per una tipologia generale di applicazioni.

2.1.3 Kernels

I **Kernels** sono un'evoluzione dell'istruction mixes, poichè non vanno a considerare più le istruzioni nella loro singolarità, ma vanno a considerare dei gruppi di istruzioni (delle funzioni). Le funzioni, in particolare, sono dette **kernel** e sono implementare solo per il consumo della CPU, quindi nessuna prevede o fa uso dei dispositivi di I/O (almeno nelle loro versioni iniziali, dato che oggi tale classe di kernel è chiamata processing kernel). Il nome **kernel** viene dal fatto che si vuole identificare una serie di passaggi chiave che poi sono utilizzati nelle più comuni applicazioni. Ad esempio si possono utilizzare tutti i passaggi che servono per il calcolo dell'inverso di una matrice o di tutte le operazioni che vengono richieste da un algoritmo di sorting. Difatti le tecniche più utilizzare ad oggi che rispettano un modello kernel sono:

- **Sieve**: Algoritmo per trovare tutti i numeri primi fino ad N (Crivello di Eratostene)
- **Puzzle**: Algoritmi per la risoluzione del gioco del 15, le N-Regine o il Sudoku
- **Tree Searching**: Operazioni che possono essere effettuate all'interno di un'albero
- **Ackermann's Function**: Funzione matematica e molto ricorsiva che permette di valutare la reazione e la gestione di tali chiamate (funzione di Ackermann)
- **Matrix Invertion**: Va a valutare il comportamento del sistema rispetto alle operazioni che bisogna effettuare per ottenere l'inverso di una matrice NxN (anche tramite diversi metodi di calcolo)
- **Sorting**: Agglomerato di algoritmi di ordinamento differenti

Però, molti dei problemi che ritroviamo all'interno dell'istruction mixes si ripercuotono anche sull'utilizzo dei kernels, quali tutti i problemi dipendenti dall'applicazione e dalla forma dei dati e non instrinsecamente dall'architettura (dove vi è sempre la mancanza però della gestione dei dispositivi di I/O)

2.1.4 Application Benchmarks

Gli **Application Benchmarks** sono dei workload che vengono costruiti in base all'applicazione che si sta andando a testare. Quindi si vanno a verificare i casi d'uso di un'applicazione in base all'impiego che ne devo fare. Nella letteratura, in realtà, benchmark viene utilizzato come sinonimo di workload, pertanto molte volte le tecniche come quella dei kernel (test di funzioni e non di singole istruzioni), vengono visti come benchmark. Il processo che vuole valutare le performance in base ad un determinato benchmark viene detto **benchmarking**. L'application benchmark, quindi, fa riferimento e cerca di replicare un workload reale in base alla tipologia di applicazione che voglio andare a valutare, ad

esempio, se voglio valutare un servizio bancario è inutile che vada a testare delle funzioni di high performance sul processore (dato che non vengono mai fatte), ma vada a valutare la qualità di utilizzo e di controllo del database.

In generale, per confrontare due sistemi, posso utilizzare i benchmark, oltretutto, una cosa importante di caratterizzazione dei benchmark, sono le proprietà che essi devono mantenere, ovvero:

- **Representativeness**(Rappresentatività): Si garantisce che il benchmark sia rappresentativo di un workload reale che si vuole andare a valutare
- **Portability**: Il benchmark deve poter essere eseguito su piattaforme diverse, e quindi non dipende dalla macchina e dall'hardware di un sistema specifico
- **Repeatability**: Eseguendo più volte lo stesso benchmark nelle stesse condizioni, i risultati devono essere coerenti. Questo garantisce l'affidabilità e la robustezza delle misure
- **Scalability**: Il benchmark deve poter funzionare su sistemi di dimensioni diverse (ad esempio da un singolo nodo a un cluster) e adattarsi a diversi livelli di carico, senza perdere significato
- **Non-intrusiveness**: L'esecuzione del benchmark non deve alterare significativamente il comportamento del sistema misurato. Deve misurare senza influenzare in modo rilevante le prestazioni stesse
- **Easy-to-use**: Deve essere semplice da configurare, avviare ed eseguire, in modo che chiunque possa utilizzarlo senza particolari complessità tecniche
- **Easy-to-understand**: I risultati prodotti devono essere chiari e facilmente interpretabili, anche da chi non è un esperto tecnico

La cosa importante quando si sceglie un benchmark e trovare uno specifico **agreement**, e quindi un accordo su quale tipologia di applicazione andare a testare. In generale un benchmark nasce per poter comparare diverse tipologie di strutture, di componenti e di architetture, rispettando però lo specifico agreement, che oltre a dare un ordine a quello che si vuole testare, permette di comparare le diverse architatture per lo specifico compito che andranno a svolgere (sempre in linea con l'agreement).

2.1.5 Esempi di benchmark

Sieve

Algoritmo che utilizza il criterio di Eratostene per la determinazione dei numeri primi da 0 ad N, con N dato in ingresso all'algoritmo. Il suo funzionamento principale è quello di partire da tutti i numeri interi da 1 ad N, e poi eliminare tutti i multipli dei valori (in ordine), da 1 a $\sqrt{N}$. I valori che però vengono considerati sono solo quelli non eliminati.
Per esempio:
N = 20, $\sqrt{20} \approx 5$

Passo 0: $[\underline{1}, \underline{2}, \underline{3}, \underline{4}, \underline{5}, \underline{6}, \underline{7}, \underline{8}, \underline{9}, \underline{10}, \underline{11}, \underline{12}, \underline{13}, \underline{14}, \underline{15}, \underline{16}, \underline{17}, \underline{18}, \underline{19}, \underline{20}]$

Passo 1: $[1, \underline{2}, \underline{3}, \underline{4}, \underline{5}, \underline{6}, \underline{7}, \underline{8}, \underline{9}, \underline{10}, \underline{11}, \underline{12}, \underline{13}, \underline{14}, \underline{15}, \underline{16}, \underline{17}, \underline{18}, \underline{19}, \underline{20}]$

Passo 2 (eliminazione multipli di 2): $[1, \underline{2}, \underline{3}, 4, \underline{5}, 6, \underline{7}, 8, \underline{9}, 10, \underline{11}, 12, \underline{13}, 14, \underline{15}, 16, \underline{17}, 18, \underline{19}, 20]$

Passo 3 (eliminazione multipli di 3): $[1, \underline{2}, \underline{3}, 4, \underline{5}, 6, \underline{7}, 8, 9, 10, \underline{11}, 12, \underline{13}, 14, 15, 16, \underline{17}, 18, \underline{19}, 20]$

Risultato finale: $[1, \underline{2}, \underline{3}, 4, \underline{5}, 6, \underline{7}, 8, 9, , 10, \underline{11}, 12, \underline{13}, 14, 15, 16, \underline{17}, 18, \underline{19}, 20]$

Algoritmo di Ackermann

Tale algoritmo è adatto per lo studio delle chiamate ricorsive, dato che costituisce la funzione ricorsiva più pura. In generale con tale tipologia di struttura si può andare a valutare bene:

- **Tempo di esecuzione medio per le call**
- **Numero di istruzioni eseguite per call**
- **Stack space per le call**

SPEC Benchmark Suite

Corporazione non profit che ha stilato una serie di bechmark (circa una decina), che possono essere utilizzati per valutazioni di varia natura. Essi sono una base per la valutazione più generale dei sistemi a prescindere dalla loro struttura architetturale

Chapter 3

Tecniche di caratterizzazione dei workload

I workload reali sono la migliore opzione per andare a valutare le performance specifiche di un dato sistema, il problema è che mantenere tale workload richiede un ingente quantità di memoria. Quindi è nata la necessità di trovare e ricercare delle tecniche che permettessero di poter caratterizzare un workload reale e ridurre la quantità di dati da memorizzare per poterne avere una statistica quanto meno affidabile

3.1 Terminologia

Gli argomenti che saranno affrontati durante tale capitolo richiedono una conoscenza della specifica terminologia utilizzata. In generale i concetti fondamentali da conoscere inerenti al dispositivo ed al componente da testare sono:

- **DUT**(Device Under Test): Sistema che viene sottoposto ad uno specifico test (es. CPU o un processo di transazione)
- **CUT**(Component Under Test): Componente del sistema di cui si vogliono conoscere le performance (es.ALU o Unità Disco)
- **Metrica**: Metrica che si vuole andare a valutare per il CUT (es. MIPS o T/s)

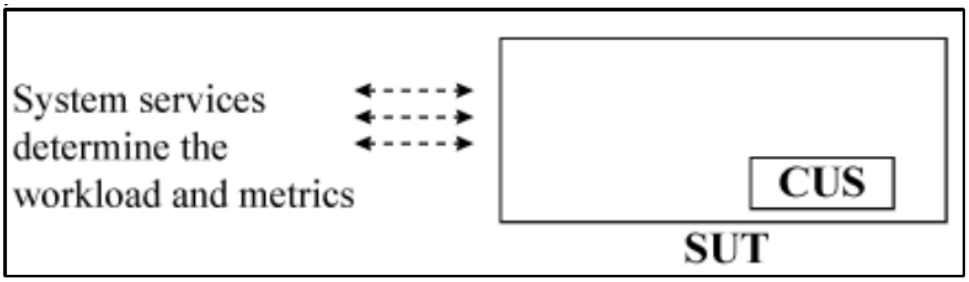


Figure 3.1: Struttura del sistema di test

Oltre la terminologia intrinseca al sistema di test bisogna definire anche la terminologia inerente ad altre entità interagenti. Quindi si definiscono i seguenti termini:

- **User**: entità che esegue le richieste di servizio

- **Workload components**(Qualitative identifiers): Componenti basilari che mi permettono di capire a livello qualitativo cosa fa un workload, quindi la natura e la struttura delle attività che vengono svolte dalle varie componenti (es. transazione in un database, una query in un motore di ricerca o Un processo o thread in un sistema operativo)

- **Workload parameters**(Quantitative Identifiers): Sono parametri quantitativi associati al workload e che quindi descrivono come le componenti si comportano. Servono principalmente per avere una misura numerica delle caratteristiche del workload (es. Arrival rate [Quante richieste al secondo], il service time [quanto tempo serve per completare un compito], Resource Usage [quante risorse vengono consumate], I/O operations [quante lettura/scritture su disco avvengono])

3.2 Workload Characterization Tecniques

Info: La **Workload Characterization** è un processo che permette di definire un workload di test di dimensione ridotta, ma che conservi tutte le caratteristiche e le proprietà (sia statiche che dinamiche), del workload reale. Ciò ne permette la replicazione e l'utilizzo in ambito di performance analysis.

C'è però da capire come sia possibile estrarre il workload sintetico dal workload reale, pertanto sono state utilizzare e definite diverse tecniche negli anni. L'obbiettivo principale di tali tecniche è quello di trovare dei parametri ridotti con cui cercare di poter descrivere il workload reale a meno di una certa quantità di informazione persa (tale quantità saraà valutata in vari modi, solitamente si utilizzerà la varianza o la devianza).

3.2.1 Averaging

La tecnica dell'**Averaging** è molto semplice, cerca di ridurre il workload in una singola istanza i cui parametri vengono valutati con la media dei parametri presenti nel workload reale. Quindi quello che si va a fare è:
Siano: $x_1, x_2, \ldots, x_n$ i valori assunti nel workload dal parametro x, allora si può approssimare tale parametro tramite la media aritmetica definita come:

$$\overline{x} = \frac{1}{n} \sum_{i=1}^{n} x_i$$

Talvolta però non è proprio l'ideale utilizzare la media, ciò dipende fortemente dalla tipologia di dati che si ha. Solitamente si potrebbe pensare di utilizzare altre tecniche come: la mediana (permette di prendere un valore appartenente al workload più centrale), oppure il 50-percentile, la media geometrica ecc.

Specifying Dispersion

Caratterizzare un workload reale mediante un workload sintetico prevede di avere, intrinsecamente, degli errori. Ciò accade principalmente per la limitatezza che ho nei parametri che voglio andare a considerare (non posso avere memoria di tutte le istanze del workload reale, ma solo di alcune di esse). Si potrebbe pensare di andare a stimare l'errore mediante la somma di tutte le deviazioni, ovvero:

$$errore\_totale = \sum_{i=1}^{n}(x_i - \overline{x})$$

Tale rappresentazione, però, non è proprio utile, il problema principale risiede nel segno che possono avere le deviazioni (immaginiamo il caso di avere 5 e 15, la media è 10, ma l'errore, se calcolato con la formula sopra è 0 [(5-10) + (15-10)]), il che porta a rendere tale ragionamento errato. Un'altra soluzione potrebbe essere quella di andare a considerare la deianza, ovvero la somma degli errori quadratici

$$errore\_totale = \sum_{i=1}^{n}(x_1 - \overline{x})^2$$

La devianza, quindi, risolve il problema del segno delle deviazioni, ma dipende fortemente dal numero di dati. Data quindi tale dipendenza dal numero di dati della devianza, si preferisce utilizzare la **varianza campionaria**, che va a dividere la devianza per n - 1. Si va a considerare n - 1 per via dei gradi di libertà, ovvero, il numero di deviazioni linearmente indipendenti [warning successivo]

$$s^2 = \frac{1}{n-1}\sum_{i=1}^{n}(x_i - \overline{x})^2$$

Warning:

Tale dimostrazione non è stata fatta in aula, per quanto sia semplice non è richiesta ai fini dell'esame ma solo per questione di conoscenza personale.

Tale formula ci fa capire perchè dividiamo per n-1 nella varianza capionaria e perchè tale divisione è giustificata come il numero di **gradi di libertà**

$$\sum_{i=1}^{n}(x_i - \overline{x}) = \sum_{i=1}^{n}x_i - \sum_{i=1}^{n}\overline{x} = \sum_{i=1}^{n}x_i - \overline{x}\sum_{i=1}^{n}1 = \sum_{i=1}^{n}x_i - n\overline{x} =$$

$$= \sum_{i=1}^{n}x_i - n\left(\frac{1}{n}\sum_{i=1}^{n}x_i\right) = \sum_{x=1}^{n}x_i - \sum_{x=1}^{n}x_i = 0$$

Dopo tale dimostrazione si comprende che le devianze linearmente indipendenti sono n-1, dato che una sarà esprimibile come somma delle altre

Oltre al concetto di varianza, solitamente, si preferisce parlare di deviazione standard, dato che è espressa nell'unità di misura della grandezza che si sta andando a

valutare. La deviazione standard si calcola come radice quadrata della varianza campionaria, ovvero:

$$s = \sqrt{s^2}$$

Oltre a tale valore, per rendersi conto dell'incertezza rispetto ai dati effettivi (essere in grado di confrontare un sistema piccolo con un sistema grande cercando di evitare un confronto rispetto alle grandezze di misura), è utile considerare il **coefficente di variazione**, che viene definito come:

$$COV = \frac{s}{\overline{x}}$$

Un esempio di utilizzo di tale valore è il seguente: Immaginiamo di avere due sistemi e di averne valutato il tempo di risposta e la deviazione standard associata ad ogni sistema. Si avrà il seguente scenario:

- **Sistema 1**: response time = 10 ms, dev. standard = 2 ms
- **Sistema 2**: response time = 200 ms, dev. standard = 15 ms

Se mi chiedessi quale dei due sistemi risulta più **stabile**, allora intuitivamente andrei a confrontare le dev. standard e valuterei quella minore come più stabile. Ma questo, però, non viene messo a confronto con gli andamenti medi (non guardo la larghezza della campana rispetto alla sua altezza [gaussiana]). Pertanto se calcolo i coefficenti di variazione, avrò che per il sistema 1: COV = 0,2 = 20%; mentre per il sistema 2: COV = 0,075 = 7,5%. Il che mi dimostra che il sistema 2 è più stabile rispetto al sistema 1 (completamente il contrario rispetto alla decisione iniziale).

3.2.2 Single Parameter Histogram

Il **Single Parameter Histogram** si occupa di costruire un istogramma delle occorrenze che vada a caratterizzare il **singolo parametro** considerato ed analizzato. La costruzione di un istogramma viene effettuata andando a valutare la frequenza di occorrenza di un dato valore in base ad una sua distribuzione discreta.

Più formalmente quello che si va a costruire è una funzione $f(x)$ che mi dice quante volte il parametro x assume un certo valore. Tale funzione viene costruita andando a suddividere l'intervallo di valori che il parametro può assumere in **buckets** (o bins), ovvero sottointervalli. Quindi si va a contare quante volte il parametro assume un valore compreso in un certo bucket e si va a riportare tale conteggio sull'asse delle ordinate, mentre sull'asse delle ascisse si riporta il bucket considerato.

Vi sono però dei problemi nell'utilizzare tale tecnica, utilizzare un istogramma per ogni parametro vuol dire utilizzare grandi quantità di memoria, a livello numerico: Consideriamo $n$bucket per ogni valore (intervalli di cui si deve tenere traccia), m il numero di parametri per ogni componente, ed k il numero di componenti, allora la memoria richiesta per memorizzare tale istogramma sarà:

$$Memoria = nmk$$

Ciò risulta troppo dettagliato per la reppresentazione del workload, oltretutto, tale metodo non tiene conto delle correlazioni tra i vari parametri, dato che ogni istogramma

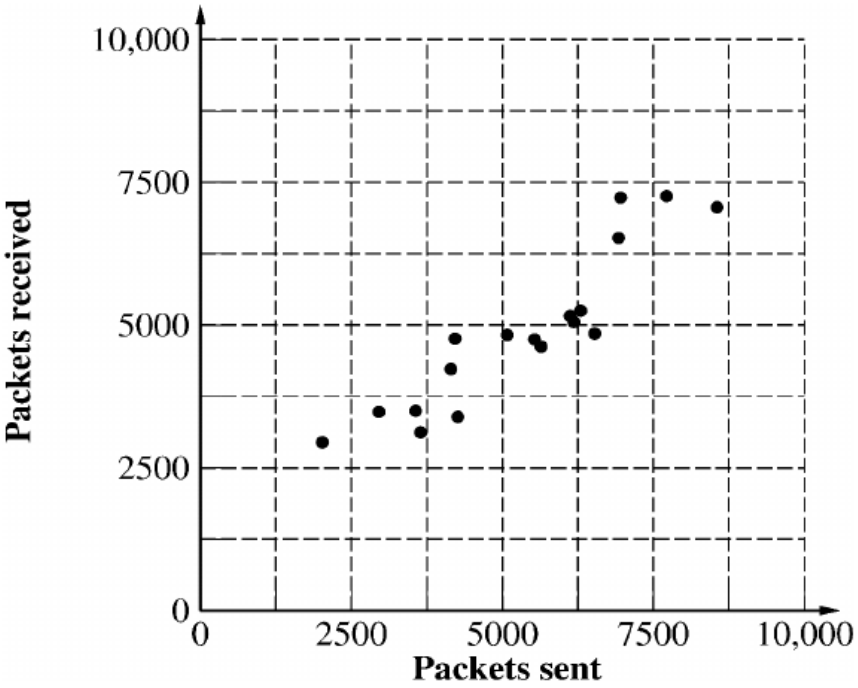
viene costruito in maniera indipendente dagli altri, ciò porta quindi a portare all'interno della descrizione del workload anche parametri che potrebbero essere deducibili da altri (ridondanza di informazioni). Questo pregiuduca anche la possibilità di selezionare i parametri da considerare mediante la varianza, dato che dati che non sono indipendenti porterebbero la stessa quantità di informazione.

3.2.3 Multiparameter Histogram

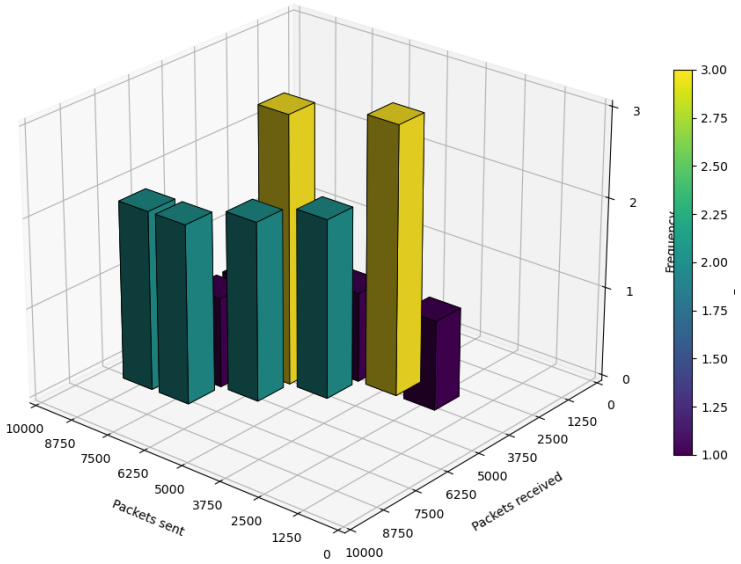
Il **Multiparameter Histogram** è una tecnica che cerca di risolvere i problemi del single parameter histogram, andando a considerare le correlazioni tra i vari parametri. Quello che si va a fare è costruire un istogramma che consideri non swolo le frequenze di un singolo parametro, ma le frequenze di n-parametri, correlate tra di loro. Per comprendere la complessità nella costruzione di un istogramma a n-variabili, vediamo un esempio con due variabili, che sarebbe quello mostrato in figura [**??**]. Per leggere tale grafico dobbiamo considerare 3 assi, che nel nostro caso sono rappresentati come:

- **Asse X**: rappresenta il range di valori del primo parametro
- **Asse Y**: rappresenta il range di valori del secondo parametro
- **Asse Z**: Viene rappresentato mediante la griglia quadrettata, ed il valore considerato è il numero di punti che sono presenti per un dato intreccio di valori.

Ciò ci aiuta a capire come si possono trovare dei pattern tra i vari parametri, data la specifica distribuzione dei parametri (in questo caso le due variabili crescono l'una rispetto all'altra). La problematica principale risiede nella quantità di parametri che bisognerebbe incatenare per trovare delle correlazioni tra le variabili, ed oltretutto, per workload molto grandi risulta complicato andare a memorizzare tale quantità di dati anche andando a considerare un filtro con la varianza.



(a) Multiparameter Histogram on 2D space



(b) Multiparameter Histogram on 3D space

Figure 3.2: Esempi di istogrammi multiparametrici

3.2.4 Principal Component Analysis (PCA)

Un modo utilizzato per ridurre il numero di parametri con cui andare a rappresentare le istanze del workload è la **Principal Component Analysis (PCA)**. Tale tecnica ha come compito quello di trasformare l'insieme di istanze in altre istanze, le nuove istanze vengono definite sulla base delle componenti principali (che differiscono dai parametri reali), poichè nel nuovo spazio, tali parametri sono tutti linearmente indipendenti, e non ci sono correlazioni. Il funzionamento matematico della PCA è basato sull'effettuazione di una media pesata per ogni parametro. Precisamente:
Dati i parametri $x_1, x_2, \ldots, x_N$, si vuole trovare un nuovo insieme di parametri $y_1, y_2, \ldots, y_N$, voglio però che l'insieme di parametri y_i sia linearmente indipendente, e che la varianza di ogni parametro y_i sia massimizzata. Di base per vedere come andare a costruire la matrice di trasformazione della PCA, si dovrebbe calcolare la matrice di covarianza, una volta calcolata si valutano gli autovettori di tale matrice, che costruiranno poi la matrice di trasformazione finale. Gli autovettori, quindi, rappresentano le direzioni principali e sono disposti in maniera che la prima componente principale sia quella con la varianza maggiore. Difatto si sta andando a fare una media pesata dei parametri per costruire il valore della nuova componente principale. Precisamente:

$$y_j = \sum_{i=1}^{N} w_{ij} x_{ij}$$

Tale formula va letta in questo modo:

- y_j: rappresenta la j-esima componente principale
- x_{ij}: rappresenta il valore del parametro i nell'istanza j
- w_{ij}: rappresenta il peso associato al parametro i per la j-esima istanza

Per effettuare la PCA bisogna seguire i seguenti passi:

1. Andare a calcolare la matrice di covarianza dei dati (prima si potrebbero effettuare anche operazioni di normalizzazione)
2. Andare a calcolare gli autovettori e gli autovalori della matrice di covarianza
3. Costruire la matrice di trasformazione mediante gli autovettori ordinati secondo gli autovalori in maniera decrescente (gli autovalori portano con loro la quantità di varianza, quindi ordinando gli autovettori si avrà uno spazio in cui il primo parametro copre la maggior varianza [utile per la selezione di un minor numero di parametri])

In maniera più compatta, quindi, effettuata la PCA, si avrà che:

- I nuovi parametri y sono calcolabili come combinazioni lineari dei parametri x
- I parametri y sono linearmente indipendenti, dato che il prodotto interno tra due parametri y è 0
- Il nuovo set di parametri y è ordinato in maniera tale che la varianza del primo parametro è maggiore della varianza del secondo e così via

Z-Score Normalization

La **z-score normalization** è una tecnica che permette di andare a normalizzare i dati, in maniera tale che ogni parametro abbia media 0 e deviazione standard 1. Tale tecnica è utile per poter effettuare la PCA, dato che si vuole evitare che parametri con range di valori molto diversi tra di loro possano influenzare in maniera sproporzionata il risultato finale. La formula per effettuare tale normalizzazione è la seguente:

$$x'_s = \frac{x_s - \overline{x_s}}{s_{x_s}}$$

Tale operazione permette di poter confrontare i parametri con un distribuzione normale, il che, quindi, andrà a rappresentare i dati come distanza da 0 e rappresentato secondo la deviazione standard s. Ciò ci permette anche di poter confrontare i dati tra di loro, senza andarea a considerare il range di valori che possono assumere.

3.2.5 Clustering

Mediante l'utilizzo della PCA si è andata ad effettuare la riduzione della quantità di parametri rappresentativi di un istanza (o componente) del workload. Per ridurre ancora di più la quantità di dati di rappresentazione del workload, si può andare ad effettuare una riduzione della quantità di istanze stesse. Per effettuare tale riduzione si fa utilizzo del **Clustering**, che è una tecnica di apprendimento non supervisionato che permette di andare a raggruppare le istanze in base alla loro similarità. Ciò richiede anche che la rappresentazione delle istanze sia adeguata per trovare degli specifici agglomerati di dati. (Per comprendere meglio, guardare la figura [3.3], se i dati non fossero ben divisi non potrei creare i cluster per bene dato che avrei molta confusione)

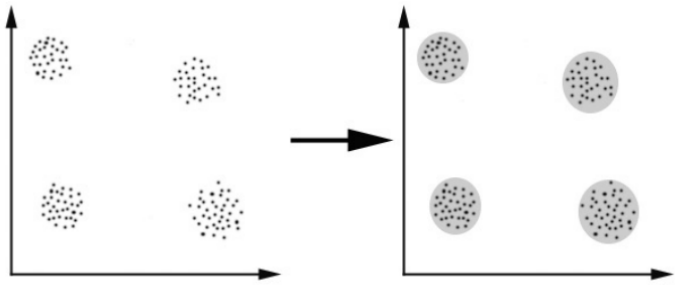


Figure 3.3: Esempio di clustering

Part II

Esercitazioni

Chapter 1

Web Server