

# Performance Modeling of Computer Systems and Networks

Prof. Vittoria de Nitto Personè

Introduction to modeling

Università degli studi di Roma Tor Vergata  
Department of Civil Engineering and Computer Science Engineering

Copyright © Vittoria de Nitto Personè, 2021  
<https://creativecommons.org/licenses/by-nc-nd/4.0/>



1

## Queueing theory

is an area of mathematics, involving stochastic analysis, which allows one to **predict** the performance of a computer system and to **improve** performance

**Idea:** to analytically model the computer system as consisting of *resources* (like CPU, bandwidth, energy, VM, disk, etc.) and jobs which require these resources. Contention occurs when several jobs simultaneously require a resource, which means some jobs must wait.

Queueing theory allows you to predict what those “waits” will be and how to reduce it



*So improving system performance!*

Prof. Vittoria de Nitto Personè

2

2

teoria delle code: area matematica, con componente probabilistica. Non tutti i sistemi sono deterministici, ma sono la minoranza, e a noi interessano poco.

Ci consente di predire le prestazioni del sistema, fare valutazioni predittive del sistema. L'idea che sta dietro è che noi stiamo studiando la condivisione di qualche RISORSA, cioè c'è un insieme di utenti che richiedono accesso ad unica risorsa, ma "non c'è spazio per tutti", se non fosse così, la previsione sarebbe quasi banale.

RISORSA può essere qualsiasi cosa, come la banda o l'energia. Poiché non riesco mai a soddisfare tutte le richieste, si creano attese, che provocano RITARDI e fanno DEGRADARE le prestazioni.

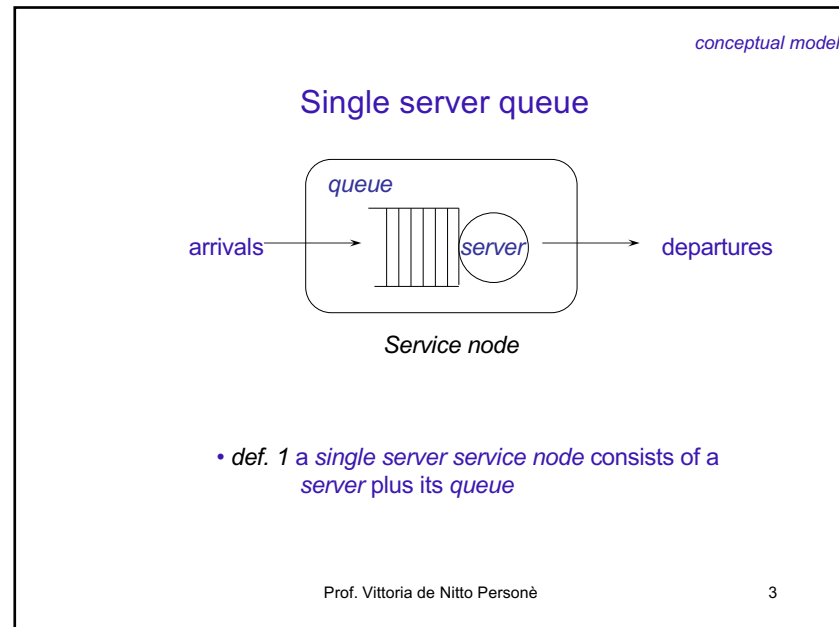
### SINGLE SERVER QUEUE/ centro a servente singolo

la risorsa "è unica", cioè una richiesta alla volta può essere soddisfatta. C'è servente singolo e la sua coda. Gli arrivi sono random (stocastici) nella maggior parte dei casi, concettualmente è random perchè non definiamo a priori quando vorrò fare la richiesta.

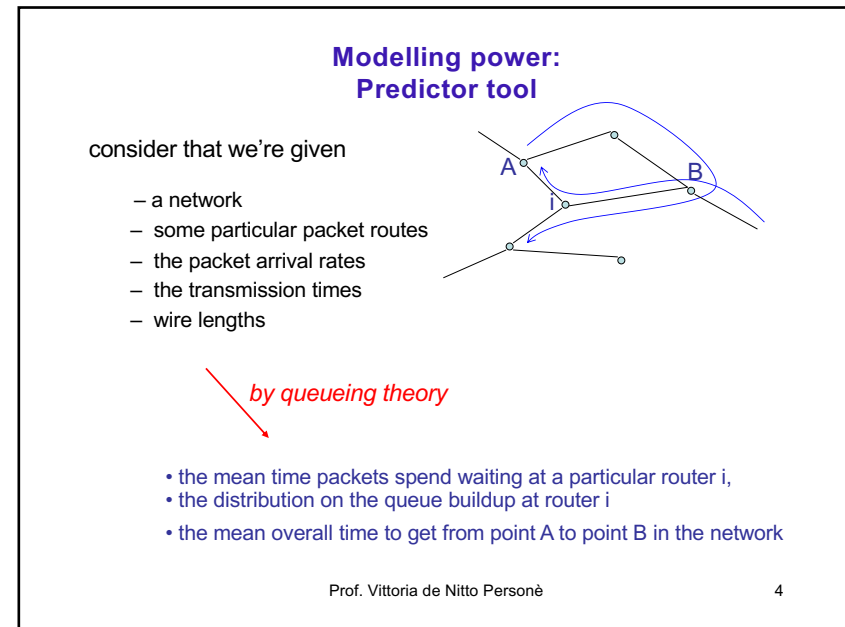
I tempi di interarrivo tra due richieste sono esponenziali, ma lo vedremo più avanti.

Il processo dei servizi è molto meno esponenziale, nella maggior parte dei casi no.

NB: non sono gli istanti di arrivo che sono cambiati (sempre random), ma è la domanda di servizio, non ben modellata da una esponenziale.



3



4

cosa facciamo con questi modelli?

ho rete = insieme risorse connesse, diversi cammini tra nodi, una freq. di arrivo. so quanti pacchetti/unità di tempo arrivo, so i tempi di trasmissione e la lunghezza dei cavi (tutto ciò viene incluso nel pallino del server).

Con la teoria delle code possiamo sapere:

- t.medio che pacchetto aspetta per avere accesso a specifico router "i".

- posso "prevedere" la distribuzione della cosa, cioè probabilità che quel router sia vuoto, che ci sia 1 pkt, 10 pkt, >100 pkt, e così via. Dobbiamo evitare il caso che la coda si avvicini a certi valori (es 90 pkt) e fare in modo di deviarlo verso un altro router. Distribuzione di probabilità.

Posso vedere la probabilità come % di tempo (es: prob che ci siano 90 pkt è 0.9 -> al 90%, osservando il sistema in tempo random, 9 volte su 10 troverò che la numerazione dei pacchetti in attesa è 90%).

Inoltre fissati due punti della rete A e B, posso prevedere tempo MEDIO (sempre sarà così) di percorrenza della rete, dal punto A a B (e viceversa).

### Modelling power: Design tool

system design is often a counter-intuitive process

#### Example 1: doubling arrival rate



*evolution of the configuration  
and workload (upgrade)*

- starting tomorrow the arrival rate will double.
- you should do whatever it takes to ensure that jobs experience the same mean response time. I.e., customers should not notice the effect of the increased arrival rate.

*Question: By how much should you increase the CPU speed in order to maintain the same mean response time?*

*Answer: Less than double!*

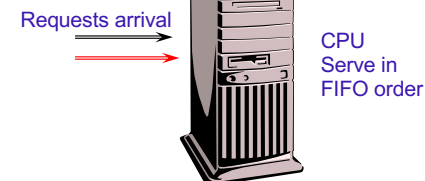
Prof. Vittoria de Nitto Personè

5

5

### Modelling power: Design tool

#### Design Example 1: doubling arrival rate



doubling CPU speed together with doubling the arrival rate will result in cutting **the mean response time in half!**

Prof. Vittoria de Nitto Personè

6

6

Suppongo di gestire un server, che ha CPU in ordine FIFO. C'è un certo flusso di arrivo. Se c'è previsione che domani traffico diventa doppio, come lo gestisco?


L'utente non deve rendersene conto, non deve sentire ritardo. Di quanto devo aumentare la velocità della CPU per mantenere stesso tempo medio, con traffico medio raddoppiato?

Serve meno del doppio, è contro-intuitivo ma è così. A livello di intuizione: se metto CPU veloce il doppio, con traffico doppio, il tempo di risposta è la metà di quello che era prima (come se raddoppiassi il clock). Un secondo del sistema 1 diventa mezzo secondo nel nuovo sistema 2. Se prima arrivavano 3 job, ora 6 job, il tempo di risposta è la metà. Serve solo una certa % di capacità in più.

**Modelling power:  
Design tool**

Design Example 1: doubling arrival rate

Requests arrival



CPU  
Serve in  
FIFO order

This is actually identical to the original system, but where time is sped up by a factor of 2 (i.e. a second of service in the original system now requires only half a second. Also, whereas in the old system 3 jobs per second arrive, now 6 jobs per second arrive)


A faster time clock! → the mean response time becomes half

Prof. Vittoria de Nitto Personè 7

**Modelling power:  
Design tool**

Design Example 1: doubling arrival rate

Requests arrival



CPU  
Serve in  
PS order

Does the answer change?

No!

se sistema opera secondo RR, la risposta cambierebbe rispetto al FIFO? no.

Prof. Vittoria de Nitto Personè 8

Processor sharing: modellazione sistemi time-sharing. se sistema ha 100 pkt, quei 100pkt sono serviti per un quanto di tempo ciascuno, serviti ciclicamente. Per modellare ciò, è come se questo quanto di tempo tendesse a 0, e i pkt simultaneamente venissero serviti, condividendo la capacità operativa di servizio. La capacità viene divisa tra quelli presenti sostanzialmente. "Simile" a Round Robin, con quanto di tempo che diventa piccolissimo.

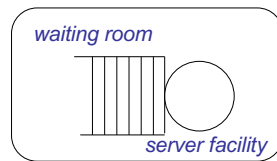
## Modelling power: Design tool

### Design Example 2

Resource



Server



Queue

Prof. Vittoria de Nitto Personè

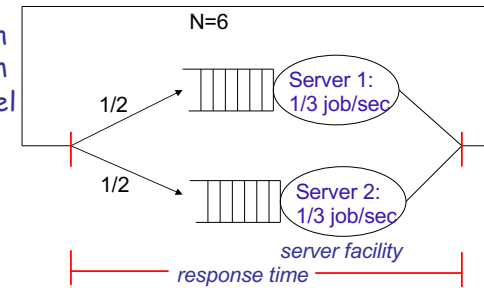
9

9

## Modelling power: Design tool

### Design Example 2

A batch  
system  
model



- The service time distribution is irrelevant

Prof. Vittoria de Nitto Personè

10

10

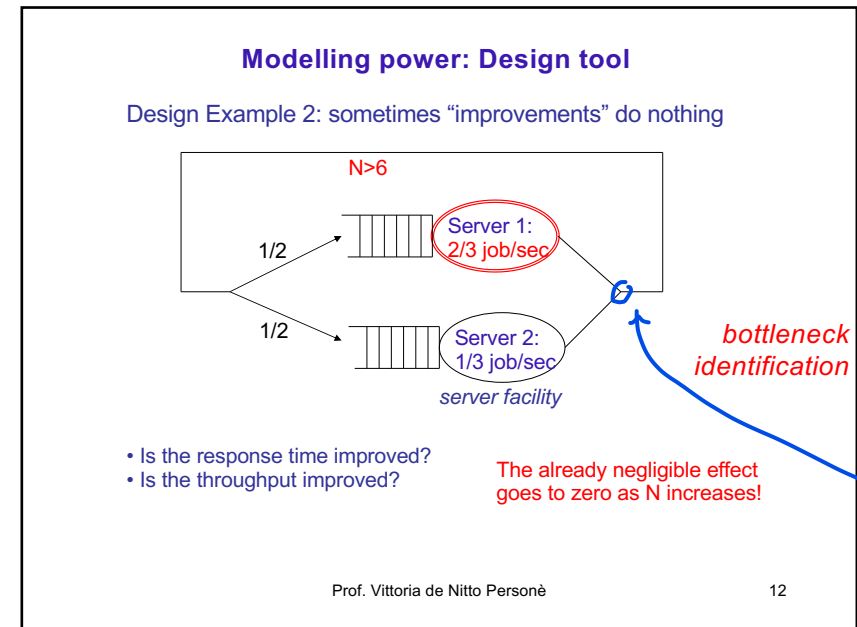
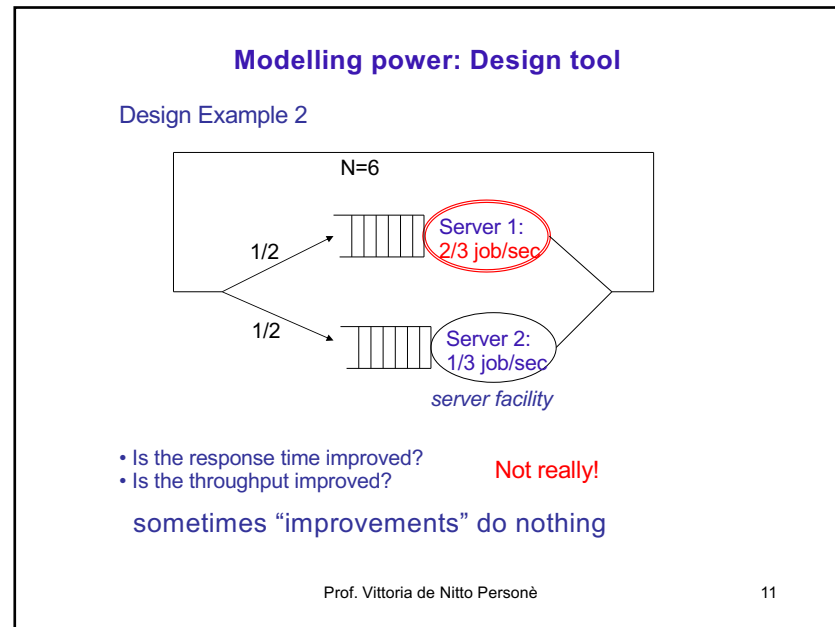
Modello server come coda. Prendo sistema con dimensione carico fissa ( $= 6$  'job'), quando termina si ripresenta istantaneamente alle due code, potendo andare a coda1 o coda2 con prob. 50% ciascuna.

Server identici, stessa capacità di calcolo: job/unità di tempo  $= 1/3$ , ovvero fanno  $1/3$  di quello che il job chiede.

Ad un certo punto aumento solo un server, raddoppio: ora è  $2/3$  job nell'unità di tempo.

- Distribuzione tempo servizio, cioè come modello domanda di servizio (media varianza forma) è IRRILEVANTE, ogni distribuzione VALIDA.

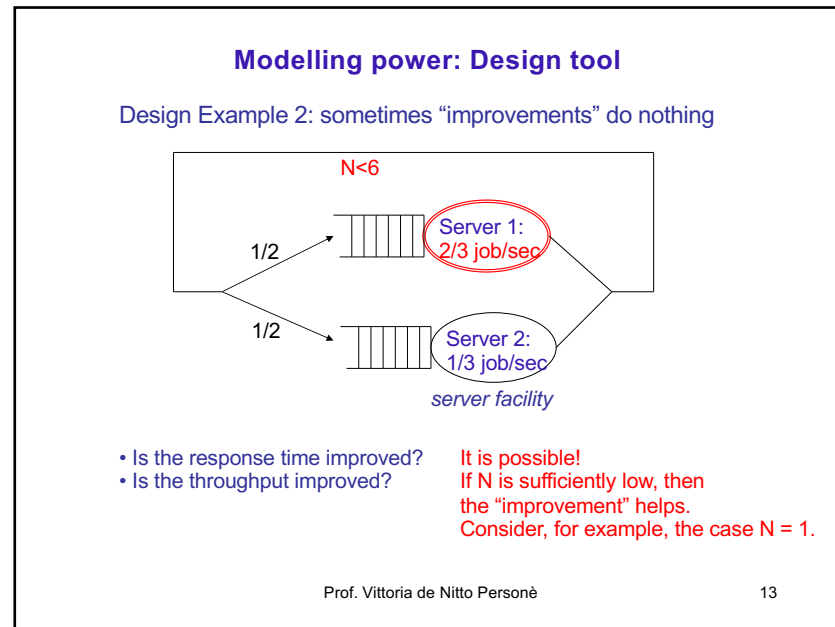
- tempo risposta medio: da quando richiesta arriva a quando termina. Lo faccio  $n$  volte, poi calcolo la media.



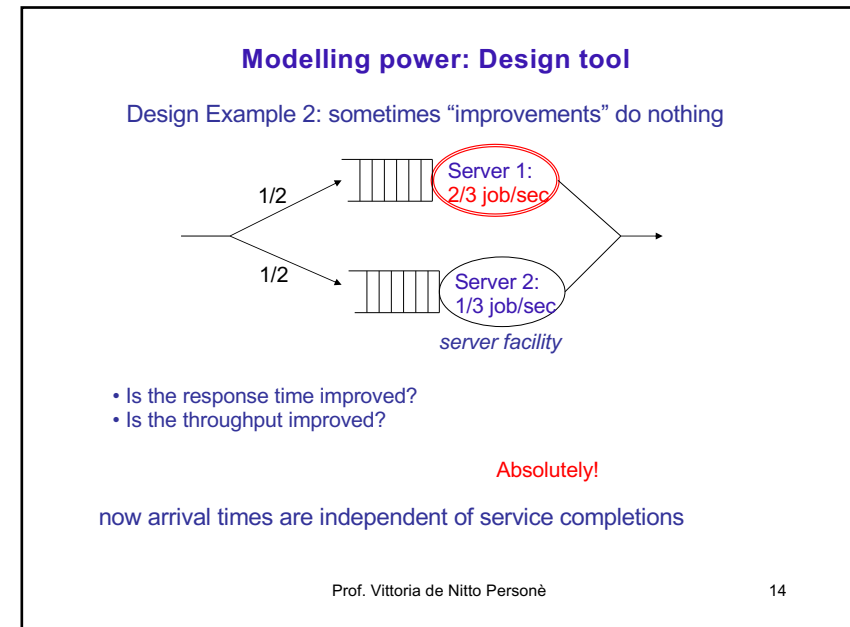
Tempo di servizio migliora? il throughput (lavoro/unità di tempo, lo vedo mettendomi fuori dal server, quando si congiungono le due linee a destra)? Scopriremo che throughput è l'inverso del tempo di risposta. (se throughput alto produco di più, e quindi attendo di meno).

Per  $N = 6$  cambia poco o nulla, i miglioramenti non sempre portano a qualcosa. Se  $N < 6$  il miglioramento si osserva di più, es  $N = 1$ , ho 50% prob di metterci la metà del tempo, 50% normale, facendo la media ho miglioramenti.

C'è bottleneck, il server2, se traffico alto non me ne accorgo che server1 è meglio. Se  $N$  cresce, non vedo miglioramenti, perchè c'è collo di bottiglia, dovrei aumentare la probabilità di andare nel server1, cioè il più potente, altrimenti se c'è bottleneck è sempre lui che limita.



13

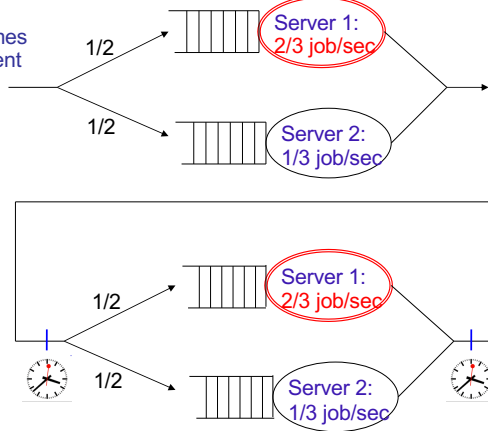


14

Se il sistema fosse aperto (non rientro appena uscito dal server) ho due processi indipendenti, cioè arrivi e completamente (prima ciò che entrava dipendeva da ciò che usciva). Qui avrò 50% del traffico che beneficerà del server più veloce.

### Modelling power: Design tool

now arrival times  
are independent  
of service  
completions



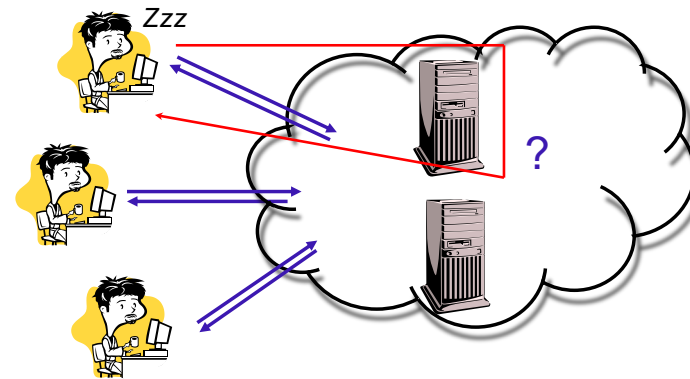
Prof. Vittoria de Nitto Personè

15

15

### Modelling power: Design tool

Design Example 3: case study



Prof. Vittoria de Nitto Personè

16

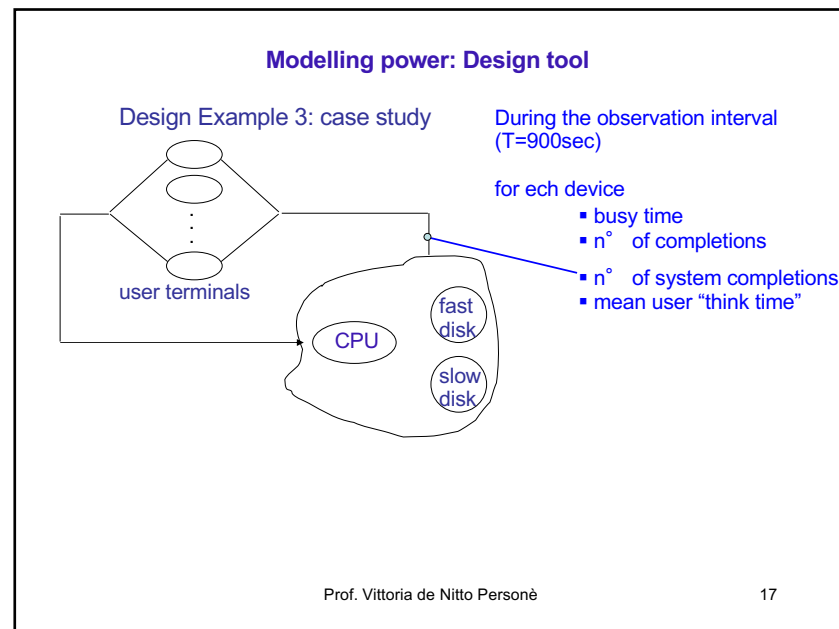
16



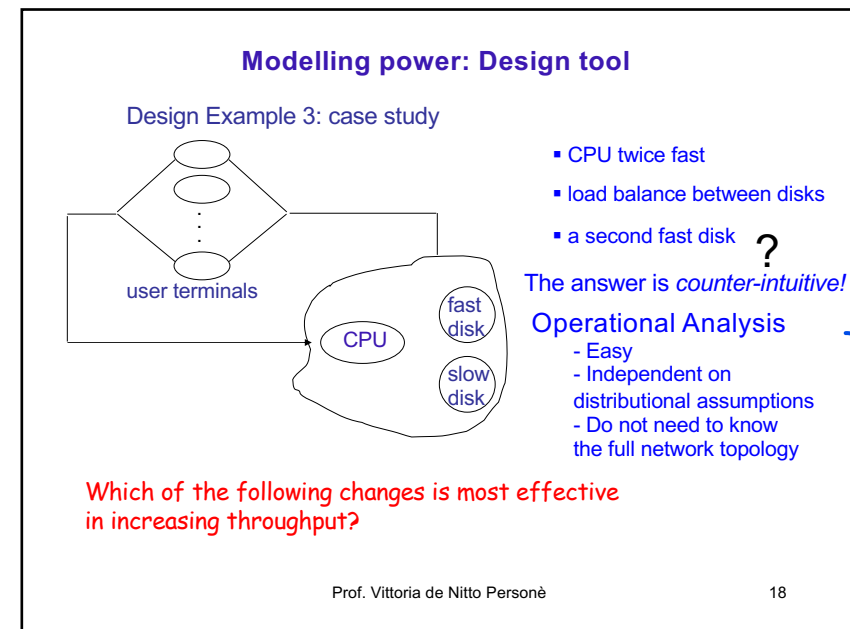
Ho nuvola (non so chi ci sia dietro) e sottopongo richieste. Faccio richiesta e torna risposta. Questo approccio è ben modellabile dal seguente modello: INFINITE SERVER  
 sistema ha quantità di risorse sufficienti per gestire tutte le richieste che arrivano. E' un caso "come se fosse infinito", perchè nella realtà non è mai così. Qui non c'è contesa, non c'è attesa. Nella nuvola non ho idea di cosa ci sia, però SAPPIAMO che c'è una CPU, un disco veloce disco lento.

Ho un tempo di osservazione ( $t = 15m = 900s$ ) nel quale sono in grado di misurare:

- quanto tempo sono occupati (busy time),
- n° completamenti del sistema (cioè sistema, quello all'uscita della nuvoletta),
- tempo medio "think time", ovvero quando arriva qualcosa arriva nella nuvoletta.



17



18

L'obiettivo dello studio è capire quali dei seguenti cambiamenti è migliore per far crescere il throughput:

- A - cpu 2x
- B - bilanciamento carico disco lento veloce
- C - prendere secondo disco veloce

L'ANALISI OPERAZIONALE non usa distribuzioni di probabilità, ma vi posso accedere, non richiede la piena conoscenza della topologia di rete. Nel progetto, non devo fare run per trovare il bottleneck, spesso bastano due conti per trovarlo, ed è indipendente dalle assunzioni sulle distribuzioni.

concentrazione vs distribuzione della capacità

meglio 1 server che fa 4 j/s, o 4 macchine che fanno 1 j/s?

al giorno d'oggi è tornato in "voga" per questioni di energia. Noi confrontiamo a parità di capacità globale (ovvero la somma totale è sempre 4, sennò non avrebbe senso). La config da 4 è più economica, ma potrebbe consumare di più.

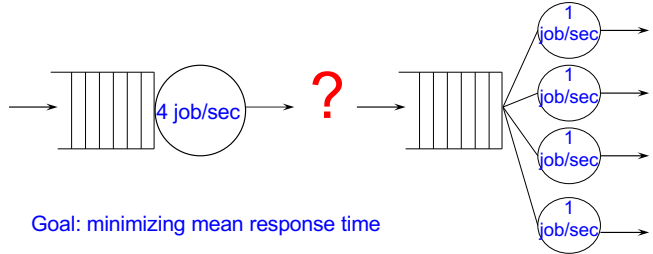
Vogliamo minimizzare tempo risposta medio. Attenzione: assumiamo jobs NON-PREEMPTIBLE, ovvero non posso interrompere i job.

Questo perchè non sempre posso bloccare, salvare stato, passare ad altro job e ritornare. Inoltre far ripartire, se non salvo lo stato, dovrei ripartire da 0 ed aggiungere lavoro.

La risposta dipende dalle caratteristiche del carico, in particolare dalla sua variabilità del carico.

**Modelling power: Design tool**

Design Example 4: One machine or many?



Goal: minimizing mean response time

Assumption: jobs *non-preemptible*  
each job must be run to completion

*hint: "it depends on the workload."*

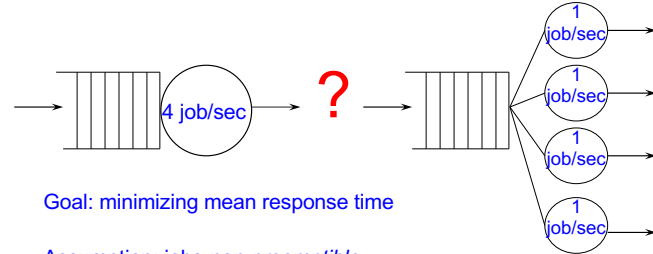
depends on the variability of the job size distribution

Prof. Vittoria de Nitto Personè 19

19


**Modelling power: Design tool**

Design Example 4: One machine or many?



Goal: minimizing mean response time

Assumption: jobs *non-preemptible*  
each job must be run to completion

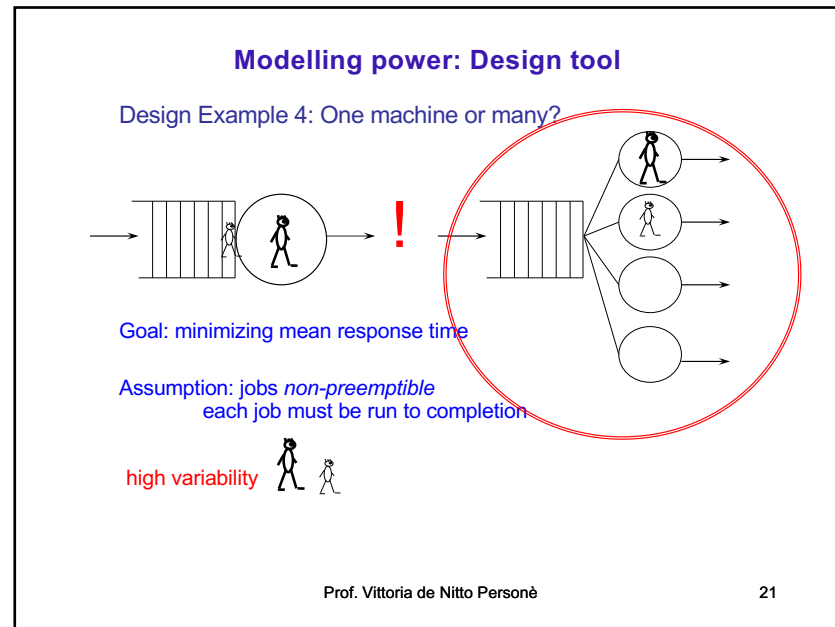
high variability 

Prof. Vittoria de Nitto Personè 20

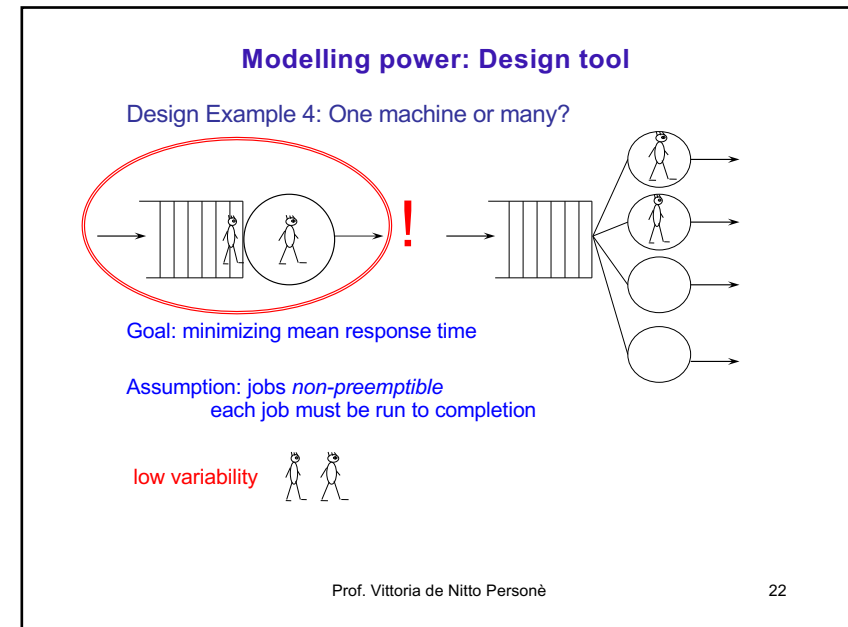
20

Se carico ALTAMENTE variabile (job chiedono tanto, e altri chiedono poco), un job che chiede tanto blocca uno che chiede poco nel singolo server, degradando prestazioni medie. Meglio 4 server. Se mi interessassi solo dei job GRANDI (piccoli sticavoli) allora avrebbe senso il single server. Noi qui stiamo valutando un tempo di risposta medio.

Se carico POCO variabile (job chiedono più o meno la stessa cosa), uso single server, poichè al singolo job do un sistema 4 volte più potente dei 4 isolati.



21



22

Se fossero INTERRUPIBILI?

uso direttamente 1 server a 4 j/s, perchè per ogni motivo posso bloccare un job grande, far passare quello piccolo e ripartire, e ognuno ha una buona capacità di calcolo (4 j/s).

**Modelling power: Design tool**

Design Example 4: One machine or many?

Goal: minimizing mean response time

Assumption: jobs *preemptible*  
each job can be stopped and restarted where they left off

Prof. Vittoria de Nitto Personè 23

23

**Modelling power: Design tool**

Design Example 4: One machine or many?

huge applicability      resource → CPU power bandwidth ...

*Resource allocation*

- power management in data centers<sup>1</sup>
- bandwidth partition

*1.5% of the total electricity in the U.S. at a cost of nearly \$4.5 billion*

*Cost vs performance*

- it is often cheaper (financially) to purchase many slow servers than a few fast servers
- many slow servers can in some cases consume more total power than a few fast ones

<sup>1</sup> A. Gandhi, M. Harchol-Balter, R. Das, and C. Lefurgy. Optimal power allocation in server farms. In *ACM Sigmetrics 2009 Conference on Measurement and Modeling of Computer Systems*, 2009.

Prof. Vittoria de Nitto Personè 24

24

*Dispositivi connessi: 1,2 miliardi 2018  
4,4 miliardi 2023*

*Consumo di energia dell'universo digitale, emissioni di CO2:  
2% 2008 → 3,7% 2020 → 8,5% 2025 → 14% 2040*

<https://www.corriere.it/dataroom-milena-gabanelli/emissioni-co2-ambiente-internet-quanto-inquina-nostra-vita-digitale-effetto-serra-consumi-invisibili-streaming-app-video/eb680526-5363-11eb-b612-933264f5acaf-va.shtml>

[https://www.facebook.com/watch/live/?v=148546343701326&ref=watch\\_permalink](https://www.facebook.com/watch/live/?v=148546343701326&ref=watch_permalink)

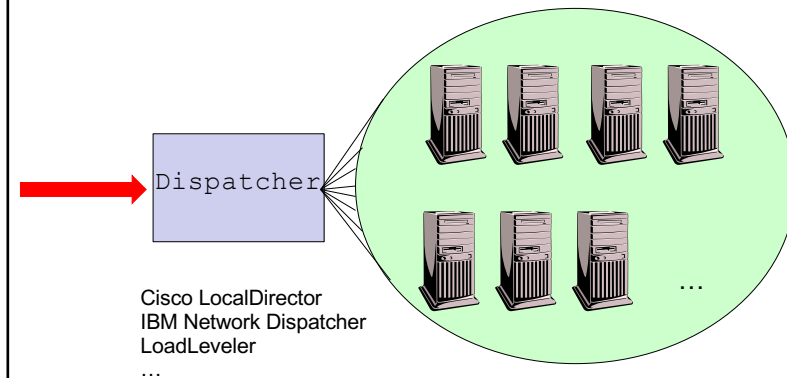
Prof. Vittoria de Nitto Personè

25

25

### Modelling power: Design tool

#### Design Example 5: Task Assignment in a Server Farm



Prof. Vittoria de Nitto Personè

27

27

#### Dispatcher - Task Assignment

Anche questo viene chiamato 'scheduling', ma è meno appropriato. Il dispatcher sceglie a chi mandare il carico, poi ogni cpu schedula il proprio lavoro come deve.

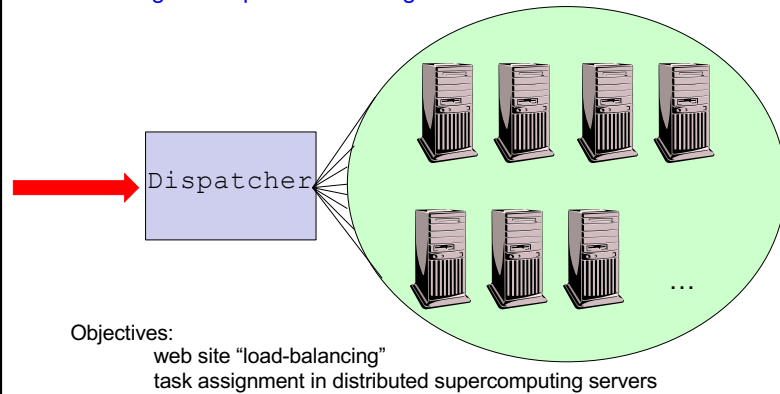
Qui importante è bilanciamento del carico (web site).

come si deve comportare il dispatcher? immaginiamo di avere 3 host, ognuno con propria coda.

Assumiamo che: server omogenei (tutti uguali), ogni host ha un singolo server (non è quad core, è una sola CPU), e la gestione è FIFO non interrompibile.

### Modelling power: Design tool

#### Design Example 5: Task Assignment in a Server Farm



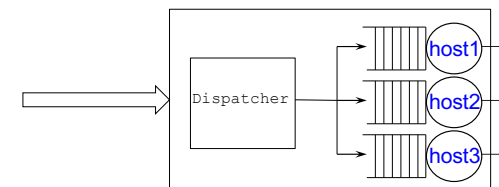
Prof. Vittoria de Nitto Personè

28

28

### Modelling power: Design tool

#### Design Example 5: Task Assignment in a Server Farm



Assumption: homogeneous  
single resource for each job  
FIFO non-preemptible

Prof. Vittoria de Nitto Personè

29

29

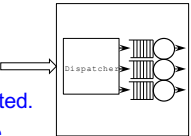
Abbiamo diverse politiche di assegnamento:

- \* random: a caso
- \* round robin: job  $i$  va a host  $i \bmod n$ , è ciclico. gli host vanno da 0 a  $n-1$ .
- \* shortest queue: guarda l'occupazione delle code, lo manda alla più vuota. ( $n^\circ$  job, NON PESO).
- \* central queue: metto una coda unica (non 3), e l'host quando si libera prende la risorsa in modalità FIFO. Dispatcher perde di significato, è l'host che li prende.

Indipendenti da quanto viene chiesto, si basano su criteri astratti, non si interessa alla size del job.

**Modelling power: Design tool**  
Design Example 5: Task Assignment in a Server Farm

*task assignment policies*



**Random** Each job flips a fair coin to determine where it is routed.

**Round-Robin** The  $i$ th job goes to host  $i \bmod n$ , where  $n$  is the number of hosts, hosts are numbered  $0, 1, \dots, n-1$ .

**Shortest-Queue** Each job goes to the host with the fewest number of jobs.

**Central-Queue** Rather than have a queue at each host, jobs are pooled at one central queue. When a host is done working on a job, it grabs the first job in the central queue to work on.

**Size-Interval-Task-Assignment (SITA)** "short" jobs go to the first host, "medium" jobs go to the second host, "long" jobs go to the third host, etc., for some definition of "short," "medium," "long."

**Least-Work-Left (LWL)** Each job goes to the host with the least total remaining work, where the "work" at a host is the sum of the sizes of jobs there.

} **job-size based**

Prof. Vittoria de Nitto Personè 30

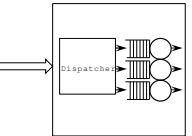
30

variabilità del carico determina moltissimo, cioè a quanti job chiedono, NON agli istanti di arrivo. gli arrivi random corrispondono a tempi di interarrivo esponenziali.

**Modelling power: Design tool**  
Design Example 5: Task Assignment in a Server Farm

*task assignment policies*

Which policies yields the lowest mean response time?



**low variability** → **LWL** Se variabilità bassa → LWL (minor carico, NO lunghezza minore).

**high variability** → **SITA** Se variabilità alta → SITA (partiziono i server).

how important was it that we know the size of jobs?

Actually, most task assignment policies do not require knowing the size of jobs. (It can be proven by induction that LWL is equivalent to Central-Queue.)

Non sempre possiamo conoscere size dei job, per questo ci chiediamo se a volte possiamo fare a meno della size. E' così indispensabile?

Prof. Vittoria de Nitto Personè 31

31

- \* size interval task assignment SITA: un host per job piccoli, un altro per medi, un altro per quelli lunghi. Quindi ogni host ha in affido i job partizionati in base a quanto loro chiedono. Riduco la variabilità, perchè ogni host avrà sempre carichi 'simili' tra loro. Vedremo come la variabilità alta è un gran problema.
- \* least work left LWL: prendiamo shortest queue, se tutti fossero grandi? posso avere 100 job veloci vs 3 pesantissimi, allora con LWL valuto l'host che ha carico minore rimanente, guardando al peso. Se variabilità bassa è simile a SQ, e mi limito a vedere quanti sono. Se variabilità alta ha senso.

Questi ultimi due sono SIZE BASED. Tale distinzione è netta, e queste ultime limitano la variabilità, portando numerosi vantaggi.

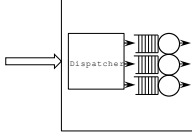
La presenza o meno dell'interrompibilità è un fattore da valutare. Normalmente tutto ciò che ha a che fare coi webserver è interrompibile.

Quando NON interrompibile, il bilanciamento del carico è inutile, può anche peggiorare le prestazioni, non è sempre la soluzione. Sono sensibili alle differenti tecniche, cioè ho prestazioni che cambiano ad ogni scheduler.

Quando interrompibile è utile bilanciare per minimizzare il 'time', ma non 'slowdown' quanto quella richiesta viene rallentata rispetto a ciò che lui chiede.

**Modelling power: Design tool**  
Design Example 5: Task Assignment in a Server Farm

*task assignment policies*




**non-preemptible** (supercomputer) → No-load balancing  
Different TAPs → different performance  
Well analyzed

**preemptible** (web server farm) → load balancing to minimize response time  
no-load balancing to minimize mean slowdown  
Open issue  
variable size distribution  
minimizing the variance of response time

Prof. Vittoria de Nitto Personè 32

32

**Modelling power: Design tool**  
Design Example 6: Scheduling policies



**Poisson process**

- no-assumption on job size distribution
- non-preemptive jobs

Which of these will result in the lowest mean response time?

**First-In-First-Out (FIFO)** When the server completes a job, it starts working on the job which arrived earliest.

**Non-preemptive Last-In-First-Out (LIFO)** When the server completes a job, it starts working on job which arrived last.

**Random** When the server completes a job, it starts working on a random job.

These all have the same mean response time !!

Prof. Vittoria de Nitto Personè 33

33

poisson -> interrarivi esponenziali.  
ci mettiamo in job non interrompibili, no assunzioni sulla distribuzione della size.  
Abbiamo le politiche:  
\* FIFO  
\* LIFO, solo quando server si libera sceglie il job arrivato per ultimo.  
\* Random  
Quale mi da minor tempo tempo di risposta medio? Sono tutte uguali.



Se preemptive?

\* LIFO, appena arriva un nuovo job, sostituisce il job in servizio.



Se c'è un pò di variabilità -> importanti miglioramenti.

Se poco variabile -> può peggiorare fino ad un fattore 2x.

Noi guardiamo sempre la MEDIA, mai il singolo job.

**Modelling power: Design tool**

Design Example 6: Scheduling policies

*Poisson process*

~~Non-preemptive~~ **Last-In-First-Out (LIFO)** When the server completes a job, it starts working on job which arrived last.

Whenever a new arrival enters the system, it immediately preempts the job in service

at least moderately variable ➡ A huge performance improvement

hardly variable ➡ Up to a factor of 2 worsening

Prof. Vittoria de Nitto Personè

34