Performance Modeling p.5-20

Performance Modeling of Computer Systems and Networks

Prof. Vittoria de Nitto Personè

Introduction to modeling Modello Analitico

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

1

Queueing theory (teoria delle code)
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, banda, energia, VM, disk, etc.) and jobs which require these resources. Contention occurs when several jobs simultaneously require a resource, which means some jobs must wait.

Risorse insufficienti → generano attesa → ritardo → influiscono sulle prestazioni

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

↓

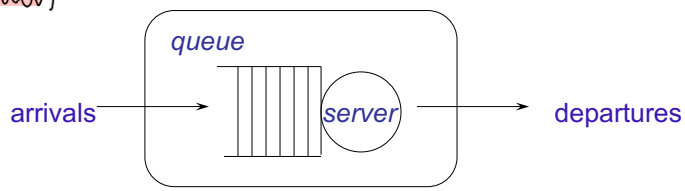
So improving system performance!

2

Centro servizio servente singolo
(normalmente a capacità infinita)

conceptual model

Single server queue

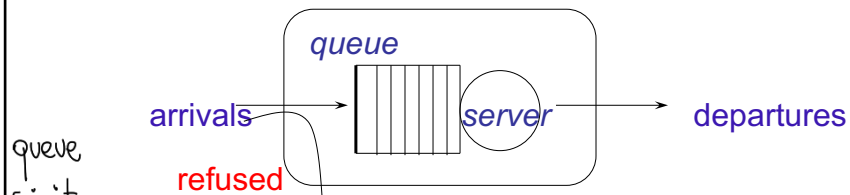


Service node

- *def. 1* a *single server service node* consists of a *server* plus its *queue*

3

conceptual model



queue finita →

refused job /loss

Service node

Assumption:
for the finite *capacity case*,
only the accepted jobs are considered

BUT
the *loss probability* could be of interest!
(voglio minimizzarla)

4

come gestire l'attesa?

def. 2 *queue discipline (scheduling / service order)*:
the algorithm used when a job is selected
from the queue to enter service

FIFO – first in, first out
LIFO – last in, first out
random – serve in random order
Priority – tipically shortest job first (SJF)
PS – processor sharing (interessanti per analisi priorità)

5

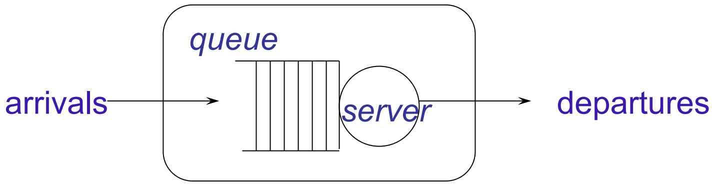
- FIFO (/ FCFS):
  - The order of arrival and departure are the same
  - A job cannot start service if the "previous" job has not left the node; this observation can be used to simplify the simulation
  - Unless otherwise specified, assume FIFO with infinite queue capacity
- service is *non-preemptive*
  - Once initiated, service of a job will continue until completion (non ho SNAPSHOT = foto stato attuale), Job rinizierebbe da ø
- service is *conservative*
  - server will never remain idle if there is one or more jobs in the service node (server fa SEMPRE qualcosa)

• In alcuni casi la NON-conservazione può avere senso!
(es: aspetto processo Liv.1, se partisse il liv.2 dovrei bloccarlo!)

6

Discrete-event simulation
Trace-driven simulation

Single server queue



Consider *n*=10 job and given arrival and service times:

Arrival times: (quando arrivano)

15 47 71 111 123 152 166 226 310 320

Service times: (quanto durano)

43 36 34 30 38 40 31 29 36 30

Prof. Vittoria de Nitto Personè 7

7

$$\sum_{i=1}^{10} (S_i - \text{mean})^2$$

Service times:

S_1 S_2 S_3 S_4 S_5 ...

										mean	variance	resp time
43	36	34	30	38	40	31	29	36	30	34,7	20,21	26,70
63	16	54	10	18	60	51	9	56	10	34,7	504,21	32,70
9	10	10	16	18	51	54	56	60	63	=	=	12,80
63	60	56	54	51	18	16	10	10	9	=	=	77,70

+20 -20 aumento la variabilità, media uguale

ordino in modo ↗ e ↙ i service times, lo scheduling NON INCIDE sulla VARIANZA e MEDIA, ma sul tempo di risposta.
La variabilità impatta la varianza (ovviamente)

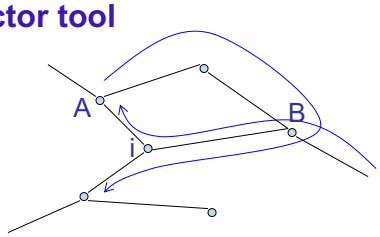
Prof. Vittoria de Nitto Personè 8

8

Modelling power: Predictor tool

consider that we're given

- a network
- some particular packet routes
- the packet arrival rates
- the transmission times
- wire lengths



una rete (routing Pkt) ben si presta alla teoria delle code (attesa media, distribuzione...)

by queueing theory

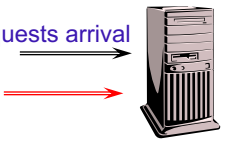
- the mean time packets spend waiting at a particular router i,
- the distribution on the queue buildup at router i
- the mean overall time to get from point A to point B in the network

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.

non mi interessa aumentarlo

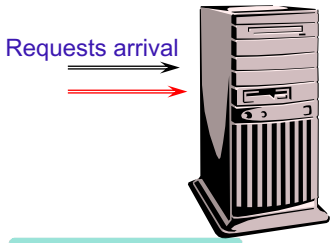
Question: By how much should you increase the CPU speed in order to maintain the same mean response time?

voglio stesso response time con arrival rate doppio!

Answer: Less than double!

Modelling power: Design tool

Design Example 1: doubling arrival rate



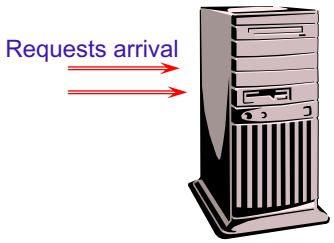
doubling CPU speed together with doubling the arrival rate will result in cutting the mean response time in half!

CPU doppia → 1s di ORIGINAL SYST diventa 0.5 s nel NUOVO (clock doppio)

11

Modelling power: Design tool

Design Example 1: doubling arrival rate



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

12

Modelling power: Design tool

Design Example 1: doubling arrival rate

Does the answer change? (cioè se non lavoro in Fifo, cambia?)

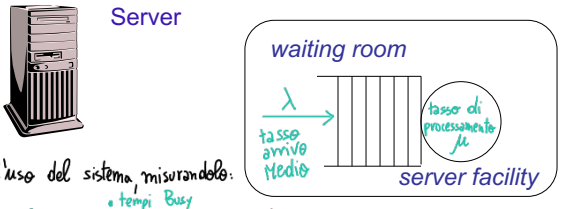
No!

13

Modelling power: Design tool

Design Example 2 sistema batch

Resource



Vorrei sapere l'uso del sistema misurandolo:
- tempi Busy
- tempi Vuoti (fascio nullo)
- intervallo da a_i a a_{i+1}

Queue

$$\rho \doteq \lim_{\substack{n\to\infty \\ T\to\infty}} \frac{B}{T} \to \frac{\sum_{i=1}^{n} s_i}{T} \to \frac{\sum_{i=1}^{n} s_i}{\sum_{i=1}^{n} n_i} \simeq \frac{E\left(\sum_{i=1}^{n} s_i\right)}{E\left(\sum_{i=1}^{n} n_i\right)} = \frac{\sum_{i=1}^{n} E(s_i)}{\sum_{i=1}^{n} E(n_i)} = \frac{n \cdot E(s)}{n \cdot E(n)} = \frac{\lambda}{\mu} \doteq \text{Prob}\{N_s > 0\} = 1 - P(N_s = 0)$$

def: $E(s) \doteq \frac{1}{\mu}$; $E(n) \doteq \frac{1}{\lambda}$

chi è in coda + chi in esecuz.

PROB che sia BUSY

$$P \doteq \begin{cases} \text{Prob. Busy} \\ \text{Popolazione media} \end{cases} \leftarrow \text{2 DEF. correlate}$$

(Se popol. media = 0 → P(Busy) = 0 perché non c'è nessuno)

14

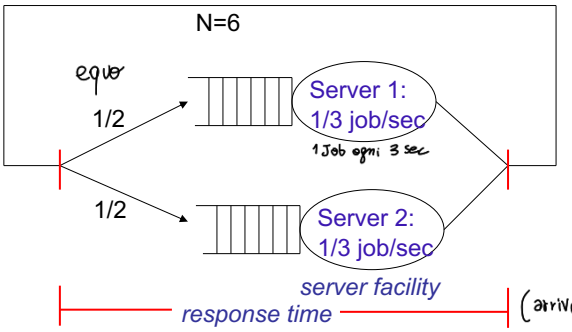
servente SINGOLO, coda infinita

$\rho = 1 \cdot \text{prob}\{N_s = 1\} + 1 \cdot \text{prob}\{N_s = 2\} + \ldots$ = calcolo popolazione media nel server (μ) → servente singolo conservativo: ↔ 1 persona / ø persona (non ho 2, 3, ...)

Modelling power: Design tool

Design Example 2 (closed system)

A batch system model



due server identici

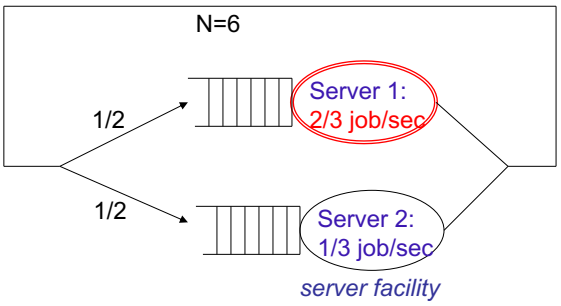
(arrivo job → completamento)

- The service time distribution is irrelevant
- ad un certo punto cambio le capacita':

15

Modelling power: Design tool

Design Example 2



- Is the response time improved?
- Is the throughput improved?

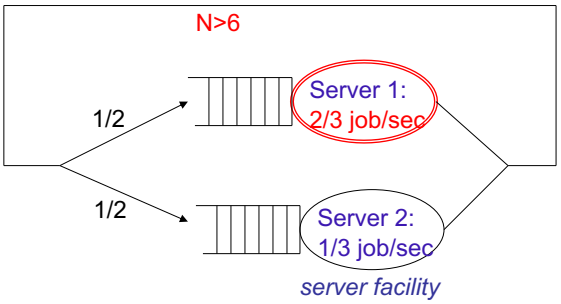
Not really!

sometimes "improvements" do nothing

16

Modelling power: Design tool

Design Example 2: sometimes "improvements" do nothing



bottleneck identification

- Is the response time improved?
- Is the throughput improved?

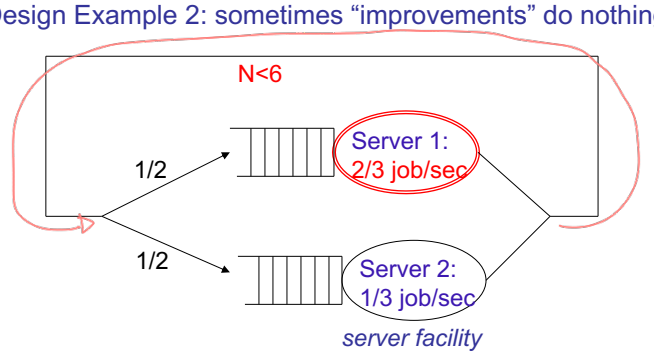
The already negligible effect goes to zero as N increases!

. N cresce → miglioramenti si notano di meno (tende a 0),
il server 2 è collo di bottiglia (bottle neck)

17

Modelling power: Design tool

Design Example 2: sometimes "improvements" do nothing



- Is the response time improved?
- Is the throughput improved?

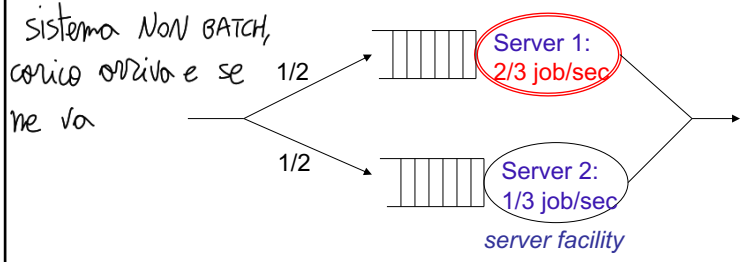
It is possible!
If N is sufficiently low, then the "improvement" helps.
Consider, for example, the case N = 1.

N decresce, ho miglioramenti !

18

Modelling power: Design tool

Design Example 2: sometimes "improvements" do nothing



- Is the response time improved?
- Is the throughput improved?

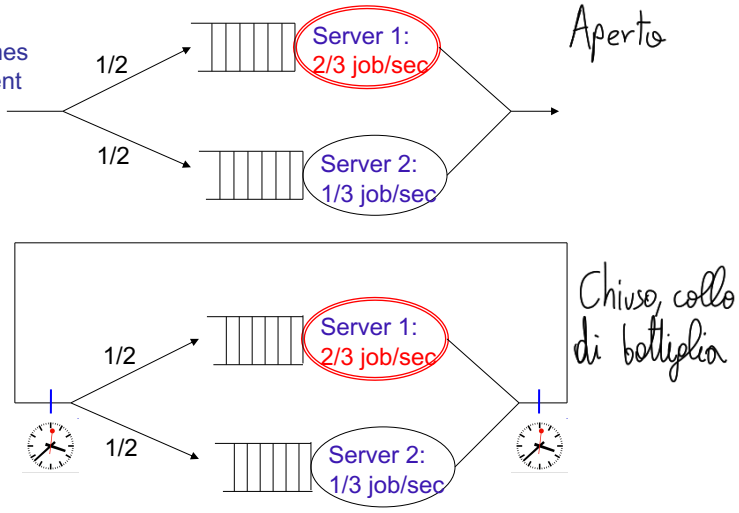
Absolutely!

now arrival times are independent of service completions

19

Modelling power: Design tool

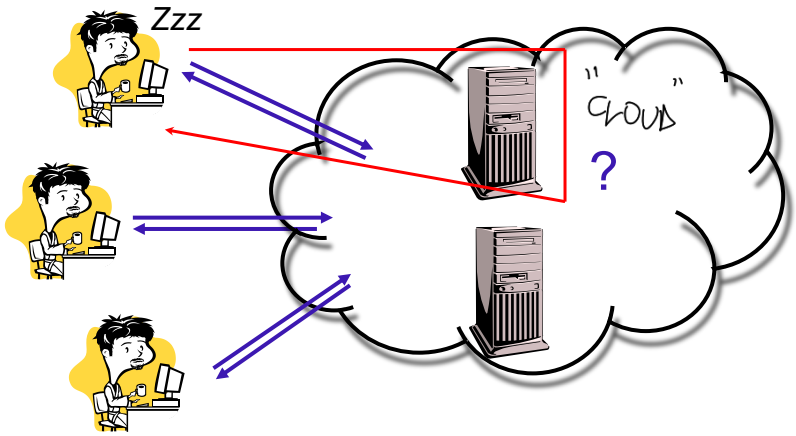
now arrival times are independent of service completions



20

Modelling power: Design tool

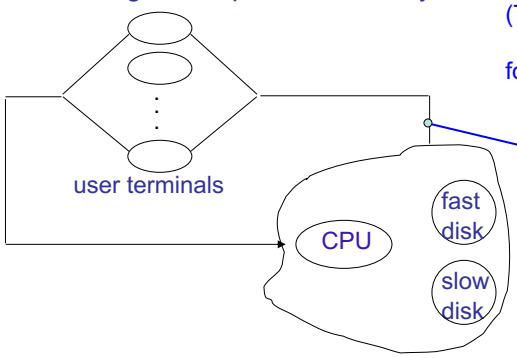
Design Example 3: case study



21

Modelling power: Design tool

Design Example 3: case study



During the observation interval (T=900sec) (osservazione)

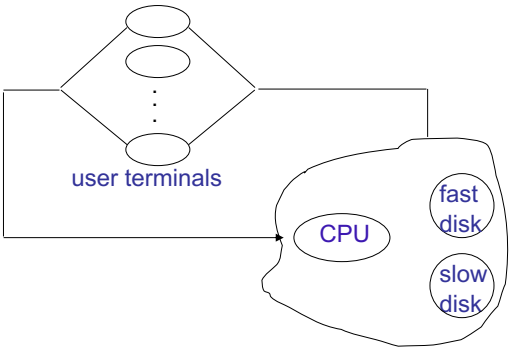
for ech device MISURO
- busy time
- n° of completions
- n° of system completions (risposte/tempo)
- mean user "think time" (tempo dell'utente vecchia risposta - nuova domanda)

obiettivo studio: come comportarsi rispetto ai 3 elementi? Quale miglioramento è più efficace?

22

Modelling power: Design tool

Design Example 3: case study



Possibili miglioramenti:

- CPU twice fast
- load balance between disks
- a second fast disk ?

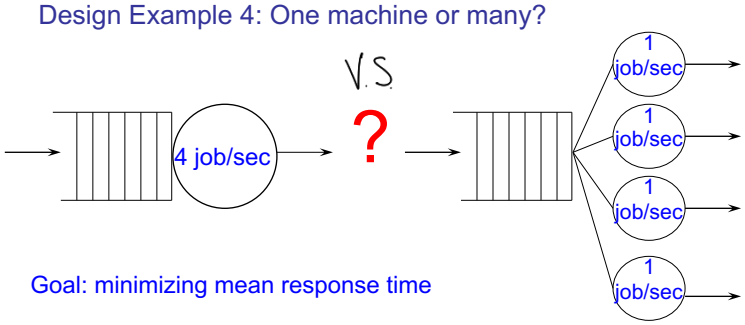
The answer is *counter-intuitive!*

Operational Analysis
- Easy
- Independent on distributional assumptions
- Do not need to know the full network topology

Which of the following changes is most effective in increasing throughput?

Modelling power: Design tool

Design Example 4: One machine or many?



Goal: minimizing mean response time

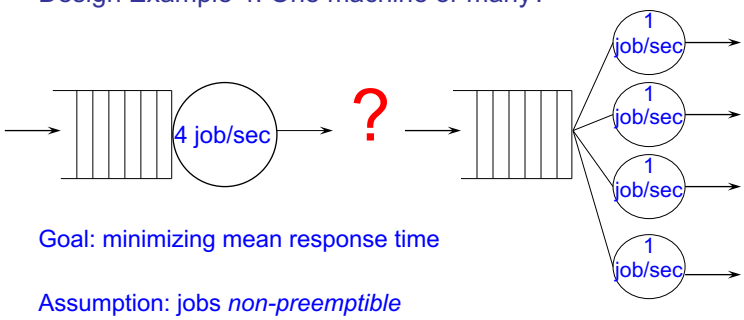
Assumption: jobs *non-preemptible* (non interrompibili)
each job must be run to completion

hint: "it depends on the workload." (caratteristiche del carico ?)

depends on the variability of the job size distribution

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

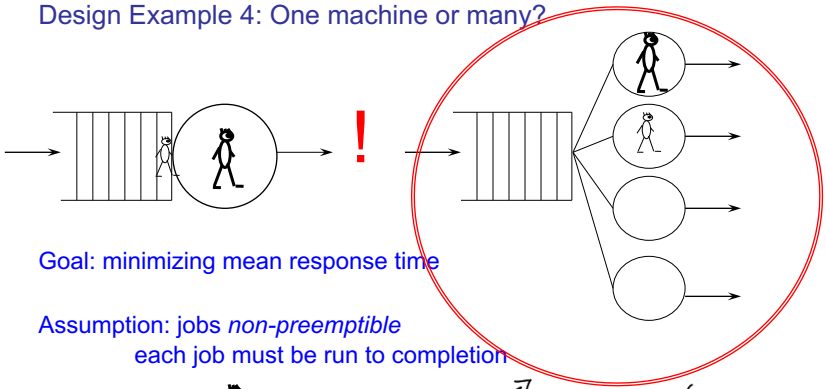
high variability 

(dipende anche dalla frequenza
alcuni server potrebbero essere vuoti
• se mediamente tutti pieni, cambio poco

25

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 

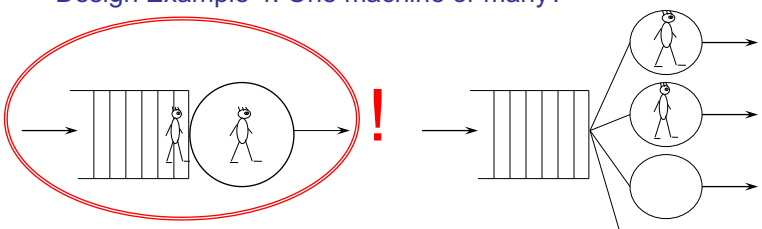
∃ periodi in cui
posso avere 2 job su 4 di media,
altri in cui è vuoto, spreco capacità
(lo tengo $\frac{1}{4}$ di secondo)

Meglio più server "lenti", con
singolo server potrei avere job
lungo che blocca quelli brevi!

26

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

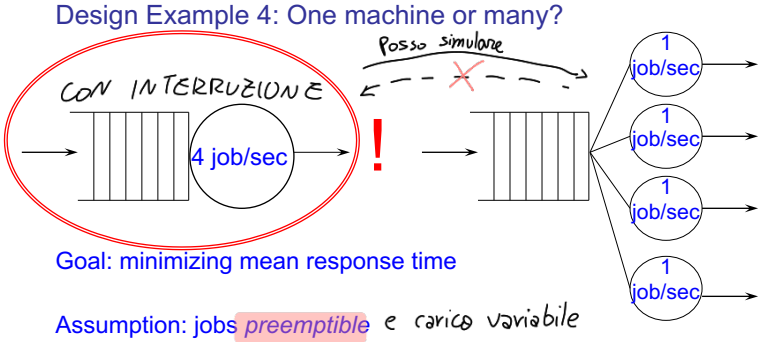
low variability 

(Lo tengo 1 secondo)

27

Modelling power: Design tool

Design Example 4: One machine or many?



Goal: minimizing mean response time

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

28

Modelling power: Design tool

Design Example 4: One machine or many?

huge applicability — *resource* → CPU, power, bandwidth, …

Resource allocation

- power management in data centers[1]
- 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

[1] 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.

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

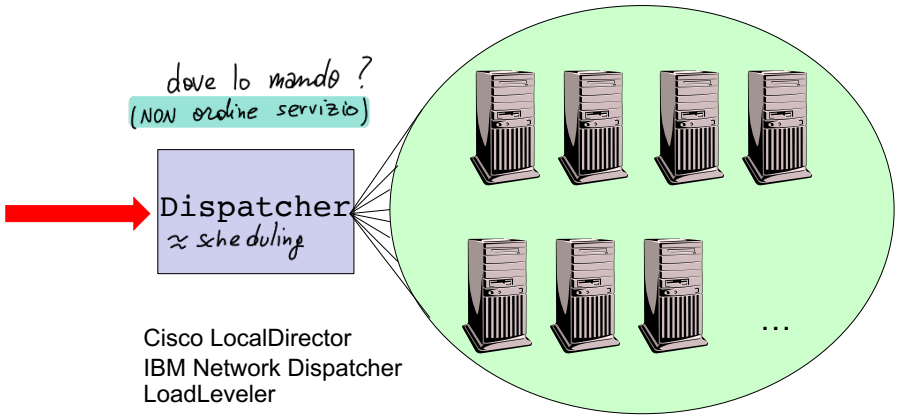
Consumo di energia dell'universo digitale, emissioni di CO2:
2% 2008→3,7% 2020→8,5% 2025 →14% 2040

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

Modelling power: Design tool

Design Example 5: Task Assignment in a Server Farm

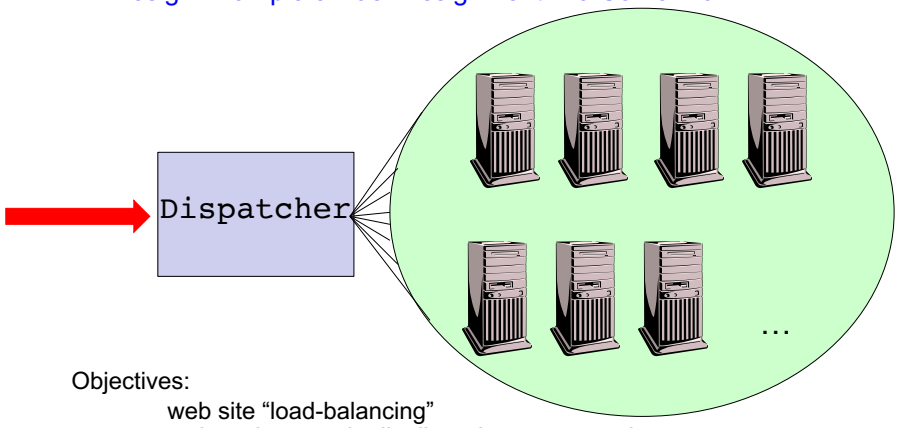


Cisco LocalDirector
IBM Network Dispatcher
LoadLeveler
…

31

Modelling power: Design tool

Design Example 5: Task Assignment in a Server Farm

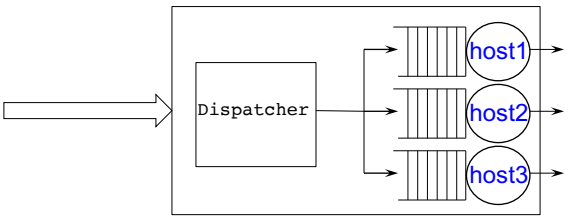


Objectives:
web site “load-balancing”
task assignment in distributed supercomputing servers

32

Modelling power: Design tool

Design Example 5: Task Assignment in a Server Farm

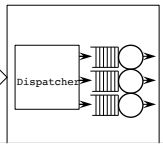


Assumption: homogeneous (host uguali)
single resource for each job
FIFO non-preemptible

33

Modelling power: Design tool

Design Example 5: Task Assignment in a Server Farm



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

Round-Robin: The ith job goes to host $i \bmod n$, where n is the number of hosts, and hosts are numbered $0, 1, \ldots, n-1$.

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

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," and "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.

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. (Ho unica coda)

job size based

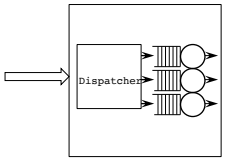
34

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**

high variability → **SITA**

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

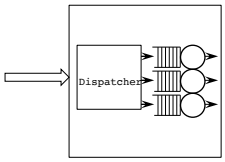
Actually, most task assignment policies do not require knowing the size of jobs.
(It can be proven by induction that LWL is equivalent to Central- Queue.)

Modelling power: Design tool

Design Example 5: Task Assignment in a Server Farm

task assignment policies

L'interrompibilità INCIDE



non-preemptible (supercomputer) →
- No-load balancing
- Different TAPs → different performance
- Well analyzed → POLITICHE
  - ∃ documenti di analisi

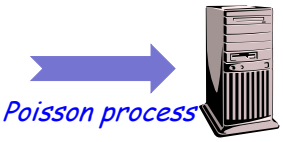
preemptible (web server farm) →
- load balancing to minimize response time
- no-load balancing to minimize mean slowdown (voglio essere equo!)

Open issue
- variable size distribution
- minimizing the variance of response time

4/03/22

Modelling power: Design tool

Design Example 6: Scheduling policies



- 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.

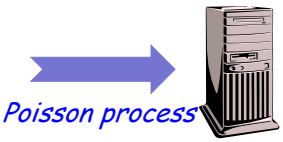
Random When the server completes a job, it starts working on a random job.

These all have the same mean response time !!

37

Modelling power: Design tool

Design Example 6: Scheduling policies



in questo nuovo caso?

~~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

(discretamente)

at least moderately variable → A huge performance improvement

(poco variabile) hardly variable → Up to a factor of 2 worsening

38