Performance Modeling of Computer Systems and Networks

Prof. Vittoria de Nitto Personè

Analytical results

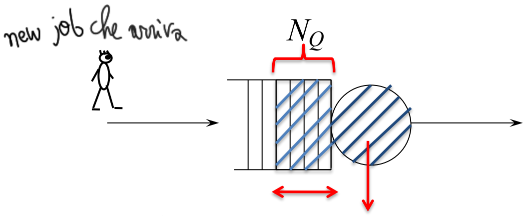
KP further results

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

1

P 31 slide,

The waiting time and the remaining service time



t_c S_{rem}

tempo per completare i job in coda

combinazione

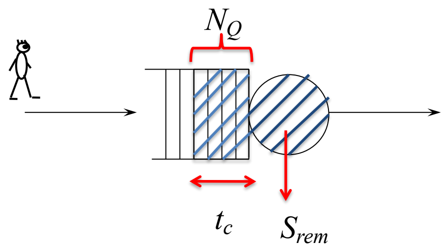
$T_Q = t_c \oplus S_{rem}$

Tq in funzione di: ?

$E(S_{rem}) \doteq \frac{\lambda}{2} \underbrace{E(S^2)}_{\text{media del quadrato del servizio}}$ ∀ distribuzione

2

The waiting time and the remaining service time



$$E(S_{rem}) = \frac{\lambda}{2}E(S^2)$$

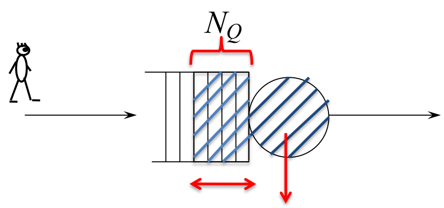
$\rho = \lambda E(S)$

nel caso exponential $\xrightarrow{\text{ho:}}$ $E(S^2) = 2E(S)^2 \longrightarrow E(S_{rem}) = \frac{\lambda}{\cancel{2}}\cancel{2}E(S)^2$

$$\boxed{E(S_{rem}) = \rho E(S)}$$

Prob che ci sia qualcuno, è esponenziale.

The waiting time and the remaining service time



$T_Q = t_c + S_{rem}$ t_c S_{rem}

dalla tabella $E(T_Q) = \frac{\rho E(S)}{1-\rho} = \frac{E(S_{rem})}{1-\rho} = \frac{1}{1-\rho}E(S_{rem})$

fattore, e' un numero puro, non tempo!

exponential $E(S_{rem}) = \rho E(S)$

caso generale

$$E(T_Q) = \frac{\rho}{1-\rho}\frac{C^2+1}{2}E(S) =$$

$$= \frac{\rho}{2(1-\rho)}\left[\frac{\sigma^2(S)}{E(S)^2}+1\right]E(S) =$$

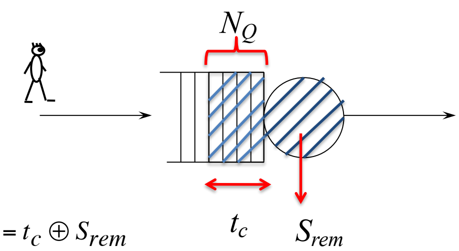
$$= \frac{\rho}{2(1-\rho)}\left[\frac{E(S^2)-E(S)^2}{E(S)^2}+1\right]E(S) =$$

$$= \frac{\lambda E(S)}{2(1-\rho)}\left[\frac{E(S^2)}{E(S)^2}-1+1\right]E(S) =$$

$$= \frac{\lambda}{2(1-\rho)}\left[\frac{E(S^2)}{E(S)^2}\right]E(S)^2 = \frac{\frac{\lambda}{2}E(S^2)}{1-\rho}$$

5

The waiting time and the remaining service time



$T_Q = t_c \oplus S_{rem}$

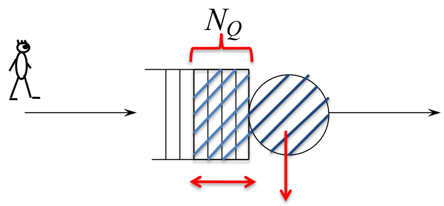
$\frac{1}{1-\rho}$ represents the mean time to complete the jobs in the queue at the arrival instant

• Rappresenta il fattore legato a tc MA NON è un tempo

$\rho \cdot E(S) = E(S_{rem})$ is the mean time to complete the job in service at the arrival instant

6

The waiting time and the remaining service time



$T_Q = t_c \oplus S_{rem}$ $\quad t_c \quad S_{rem}$

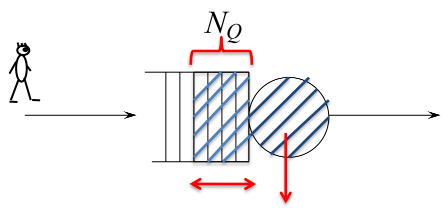
$\frac{1}{1-\rho}$

$\rho \cdot E(S) = E(S_{rem})$

the two terms are compounded through the multiplication:
- if one is 0, the result is 0
- if one is 1, it does not give contribution

7

The waiting time and the remaining service time



$T_Q = t_c \oplus S_{rem}$ $\quad t_c \quad S_{rem}$

Non c'è coda, new job subito servito

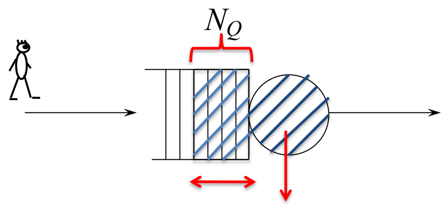
$\frac{1}{1-\rho}$ $\quad E(S_{rem}) = 0$ ⇨ The queue cannot be ⇨ The incoming job does not wait, but it is immediately served

$E(S_{rem})$ $\quad \frac{1}{1-\rho} = 1$ ⇨ The component does not give contribution, the incoming job finds just the job in service with remaining service so small that it does not wait

coda così piccola che il new job trova solo job in esecuzione che ha "quasi finito"

8

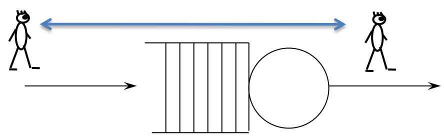
The waiting time and the remaining service time



$T_Q = t_c \oplus S_{rem}$ $\quad t_c \quad S_{rem}$

$$E(T_Q) = \frac{E(S_{rem})}{1-\rho} = \frac{\overbrace{\frac{\lambda}{2}E(S^2)}^{\text{tempo servizio rimanente}}}{\underbrace{1-\rho}_{\text{fattore}}}$$ M/G/1

The response time



$$E(T_S) = E(T_Q) + E(S) = \frac{\frac{\lambda}{2}E(S^2)}{1-\rho} + E(S)$$ M/G/1

$$E(T_S) = \frac{\rho E(S)}{1-\rho} + E(S) = \frac{E(S)}{1-\rho}$$ M/M/1

P 32 Slide

Non-preemptive abstract scheduling

(ho risultati più forti di quelli della KP)

Def. 1

A policy is *preemptive* if a job may be stopped part way through its execution and then resumed at a later point in time from the same point where it was stopped. A policy is *non-preemptive* if jobs are always run-to-completion.

Preemptive se fermo Job e dopo lo riprendo da dove si ero fermato

Def. 2

A *work-conserving* scheduling policy is one which always performs work on some job when there is a job in the system.

Theorem 1 (Conway, Maxwell, Miller[1]).

All non-preemptive service orders that do not make use of job sizes have the same distribution on the number of jobs in the system.

NON SOLO le medie!

$E(N_S)$ $E(T_S)$ $E(N_Q)$ $E(T_Q)$

[1]Richard Conway, William Maxwell, and Louis Miller, Theory of Scheduling Addison-Wesley Publishing Company, Inc., 1967. Chapter 8

11

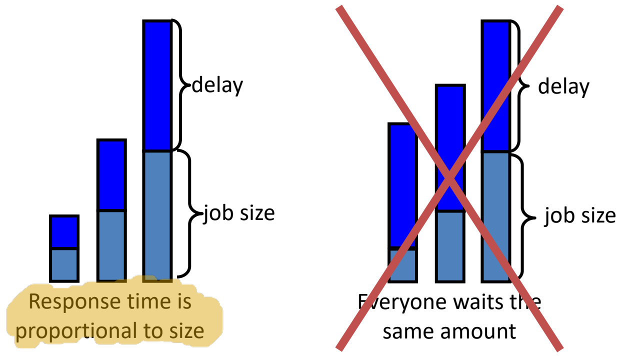
Non-preemptive abstract scheduling

$$E(T_Q) = \frac{\frac{\lambda}{2} E(S^2)}{1-\rho}$$

which is very high when $E(S^2)$ is high (anche se ρ piccolo!)

12

WHAT IS FAIR?



13

Let us consider the mean time in system for a job of size x

dipende da taglia 'x'

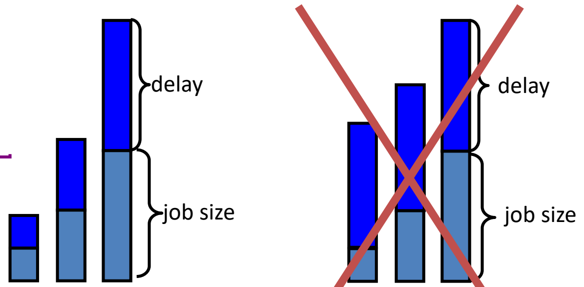
$$E(T_S(x)) = E(x + T_Q(x)) = x + E(T_Q) = x + \frac{\frac{\lambda}{2}E(S^2)}{1-\rho}$$

(non dipende da size (in $\frac{\lambda}{2}\frac{E(S^2)}{1-\rho}$ non c'è x))

non è $E(S)$, ma ciò che chiedo!

14

WHAT IS FAIR?



Response time is proportional to size

Everyone waits the same amount

Define the slowdown for a job of size x as

condizionato

$$E(sd(x)) = \frac{E(T_S(x))}{x}$$

Misura fairness

15

Slowdown for jobs of size x

Def.
The mean slowdown for jobs of size x is the observed mean response time in respect of their size, that is

$$E(sd(x)) = \frac{E(T_S(x))}{x} = \frac{[\text{tempo risposta size}]}{[\text{quanto chiede}]}$$

metto KPe divido per 'x'

$$1 < E(sd(x)) = 1 + \frac{\frac{\lambda}{2}E(S^2)}{x(1-\rho)}$$

inverso a x

Se fosse =1 non attendo nulla!
(chiedo e spendo x)

Sono in sist non-preemptive, (es FIFO), con T_Q e $E(S)$, normalmente ha senso pensare che job più piccoli siano svantaggiati nell'aspettare big jobs.

es: se P_1 e P_2 aspettano $T_q = 30s$, $x_1 = 5s$ e $x_2 = 35s$, ovviamente P_1 è svantaggiato
(Tipo andare alle poste, 2ore di fila per operazione veloce)

Note that small jobs have a higher expected slowdown than do big jobs.

Cosa succede a quelli grandi? sono pochi ma ci sono, se passano davanti i job piccoli, i grandi vanno in starvation?

16

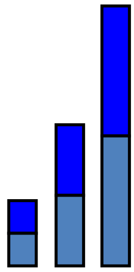
Slowdown vs Response time

Response Time tends to be representative of the performance of just a few jobs — the bigger ones
tends to emphasize the performance of the really big jobs, since they count the most in the mean, since their response time tends to be the greatest (emphasized for heavy-tail distr.)

Slowdown tends to be representative of the performance of most jobs – because it is dominated by the performance of the large number of small jobs.

we would like to make $E(T_S(x))$ smaller for small x

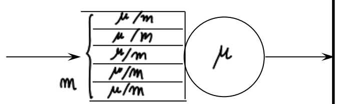
(se alle parte op veloce, vorrei aspettare poco!)



33 slide, 508 Performance

Processor sharing

we would like to make $E(T_S)$ smaller for small x



How do we do this if we DON'T know job sizes?

two reasons historically why CPU scheduling is (approximately) processor-sharing

1. in a multi-resource system (including a CPU, disk, memory, etc.) it is useful to have many jobs running simultaneously (rather than just one job at a time) because jobs requiring different resources can be overlapped to increase throughput. (se ∀ job do quanto di tempo, little job finiscono, big jobs no.)

2. PS is a good way to get small jobs out fast, given that we don't know the size of the jobs.

Analytical models
PS scheduling

Processor sharing

tempi risposta medi

should be better than FIFO with respect to $E(T_S)$, because PS gets small jobs out faster, and PS should be a lot better than FIFO with respect to $E(sd)$!

Processor Sharing ammortizza variabilità

distribuzione $$Pr\{N_S = n\}^{M/G/1/PS} = \rho^n(1-\rho) = Pr\{N_S = n\}^{\overbrace{M/M/1}^{\text{esponenziale}}/FIFO}$$

popolazione globale $$E(N_S)^{M/G/1/PS} = \frac{\rho}{1-\rho} = E(N_S)^{M/M/1/FIFO}$$

tempo risposta $$E(T_S)^{M/G/1/PS} = \frac{E(S)}{1-\rho} = E(T_S)^{M/M/1/FIFO}$$

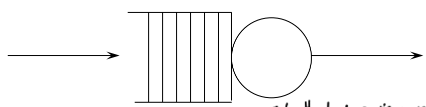
PS is better then FIFO when $C^2>1$

the M/G/1/PS queue is insensitive to the variability of the service time distribution, G (variabilità non compare nelle formule)

Quanto appena detto è vero sempre, o esistono casi particolari?

2 arrivi simultanei che richiedono 1 s di servizio

Fifo

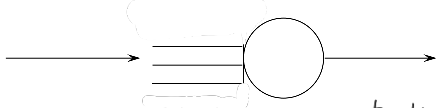


↳ un Job attende 1 e viene servito per un altro s, l'altro subito servito

$E(T_S)=(1+2)/2=1.5$ s

$E(T_S)$?

PS



Adesso J_1 e J_2 accedono insieme, ma capacità server è la metà: entrambi hanno $\frac{1}{2}$ capacità a testa per secondo.

$E(T_S)=2$ s

$E(T_S) = 2\,(0.5\ j/s + 0.5\ j/s) = 2$

PS può essere peggio su alcune sequenze di job!

Processor sharing

$$E(T_S(x))^{M/G/1/PS} = \frac{x}{1-\rho}$$

non c'è la size

$$E(sd(x))^{M/G/1/PS} = \frac{1}{1-\rho}$$

PS è con prelazione

all jobs have same slowdown: PS as "FAIR" scheduling , equo, rallenta tutti ugualmente

Fifo: $$E(sd(x))^{M/G/1/abstract} = 1 + \frac{\frac{\lambda}{2}E(S^2)}{x(1-\rho)}$$

all the preemptive non-size-based scheduling policies produce the same mean slowdown for all job sizes

$$E(sd(x))^{M/G/1/\,preemp-non-size-based} = \frac{1}{1-\rho}$$

We would like to get lower slowdowns for the smaller jobs

But how can we give preference to the smaller jobs if we don't know job size?

we do know a job's age (CPU used so far), and age is an indication of remaining CPU demand

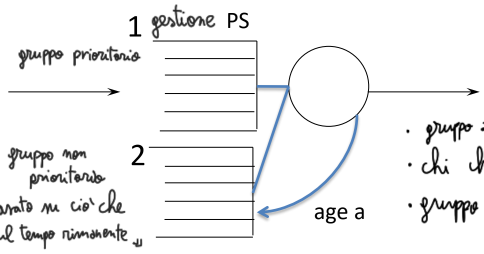
If the job size distribution has DFR (e.g. Pareto distribution) then the greater the job's age, the greater its expected remaining demand
→ give preference to jobs with low age (younger jobs) and this will have the effect of giving preference to jobs which we expect to be small

In pratica: se job nuovo "scommetto" rimanga poco (age bassa). Se age alta scommetto vi rimanga tanto (Decreasing Failure Rate)

(heavy tail: leggere par. 20.5 !)
Performance's Book

35 slide, 516 Perf

Scheduling in Unix:
Foreground-Background scheduling (≐ Multi Level Processor Sharing)



"L'ordine è basato su ciò che chiedono e sul tempo rimanente"

- gruppo 1, dopo 'age a', passa a gruppo 2.
- chi chiede poco se ne va prima.
- gruppo 2 servito se gruppo 1 vuoto.

Coda 1 principale, se vuota passo alla 2. Un job passa da coda 1 a coda 2 superando un age threshold. (se job più vecchio di tale parametro, va in coda 2). Appena arriva job in coda 1, servo lui dandogli precedenza. Mi baso sull'age (NOTA), poiché NON conosco la SIZE.