Performance Modeling of Computer Systems and Networks

Prof. Vittoria de Nitto Personè

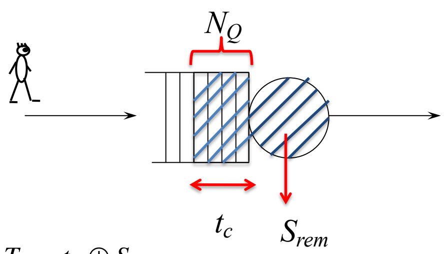
Analytical results
KP further results

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

1

Tempo che spende un job prima di entrare in servizio? (NON CHE ESCE DAL SERVIZIO) tempo attesa di questo job in coda (in tc considero i job in coda che mi precedono incluso il loro completamento) e quanto manca al job servito prima di completare il servizio.

The waiting time and the remaining service time



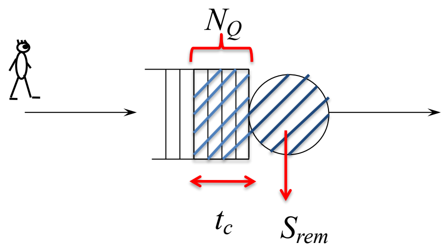
$T_Q = t_c \oplus S_{rem}$

che operazione li lega?

$E(S_{rem}) = \frac{\lambda}{2} E(S^2)$ di una distribuzione QUALSIASI

2

The waiting time and the remaining service time



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

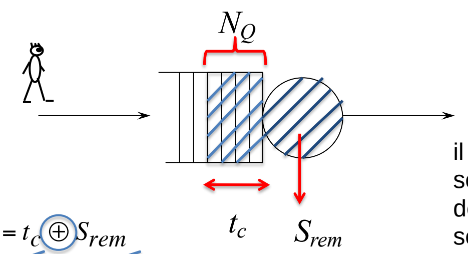
caratteristica unica dell'esponenziale

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

$$\boxed{E(S_{rem}) = \rho E(S)} \qquad \rho = \lambda E[S]$$

3

The waiting time and the remaining service time



il job, prima di entrare in servizio,
deve aspettare il job in servizio attualmente,
più quelli in coda che lo precedono (attesa in coda + loro che vengono serviti)

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

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

legato a Tc

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

4

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

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

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

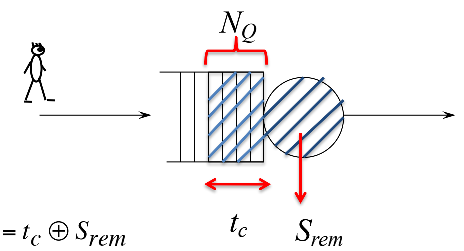
$$= \frac{\lambda E(S)}{2(1-\rho)}\left[\frac{E(S^2)}{E(S)^2}-1+1\right]E(S) =$$

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

formuline che sicuramente vedrò, stai sereno.

Anche se C^2 = 25, avrei $\frac{C^2+1}{2}$ = 13, cioè tempo attesa in coda è 13 volte quello in servizio.

The waiting time and the remaining service time

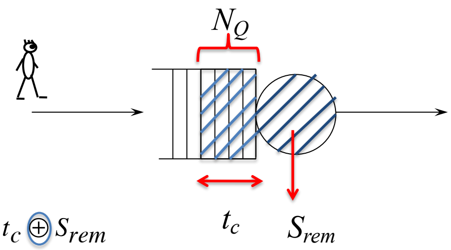


$T_Q = t_c \oplus S_{rem}$

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

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

The waiting time and the remaining service time



$T_Q = t_c \oplus S_{rem}$

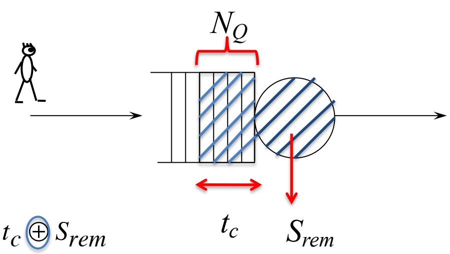
$\frac{1}{1-\rho}$

$E(S_{rem})$

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

possiamo dare un significato a questi due valori?

The waiting time and the remaining service time



$T_Q = t_c \oplus S_{rem}$

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

$E(S_{rem})$

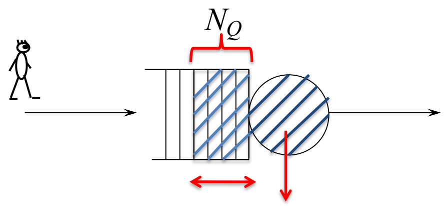
job che arriva prende servizio subito

$E(S_{rem}) = 0$ ⇒ The queue cannot be ⇒ The incoming job does not wait, but it is immediately served

$\frac{1}{1-\rho} = 1$ ⇒ 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

trovo solo job in servizio con tempo piccolo, come se non aspettassi.
rho infatti è molto piccolo.

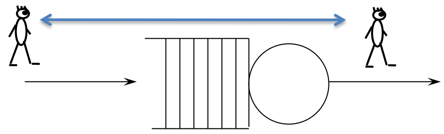
The waiting time and the remaining service time



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

$$E(T_Q) = \frac{E(S_{rem})}{1-\rho} = \frac{\frac{\lambda}{2}E(S^2)}{1-\rho} \qquad \text{M/G/1}$$

The response time



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

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

servizio esponenziale, in pratica si semplifica la formula della M/G/1, il numeratore del rapporto.

Nell'analisi, i parametri sono lambda e mu, posso vedere E[Ts] come: $\frac{1}{\mu - \lambda}$
se rho < 1

Non-preemptive abstract scheduling

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.

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.

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

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

11

Il teorema 1 ci dice che: Tutti gli scheduling non-preemptive e non usino la size dei job (quindi scheduling astratti, basati su fattori indipendenti da quanto chiedono, come divisione dei flussi etc) hanno la stessa DISTRIBUZIONE DELLA POPOLAZIONE NEL SISTEMA Ns.
Non solo stessa media e varianza, bensì proprio stessa forma della distribuzione, cioè tutti i momenti.

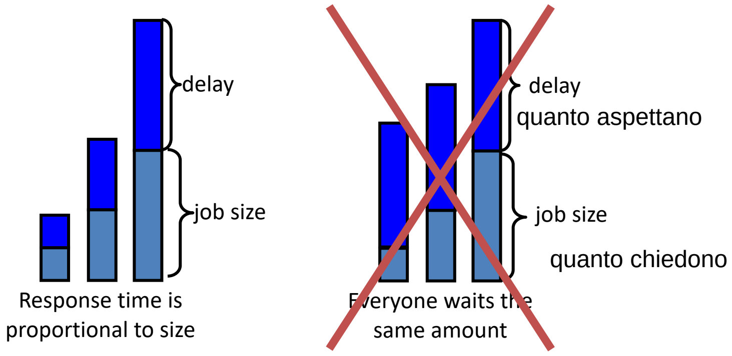
Non-preemptive abstract scheduling

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

which is very high when $E(S^2)$ is high

12

WHAT IS FAIR?



13

stiamo parlando delle non preemptive astratte!

Per misurare tali aspetti si è introdotto un nuovo indice: SLOWDOWN, misura il rallentamento che un servizio riceve in virtù dell'attesa che deve sperimentare.

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

slowdown medio per i job di taglia 'x'

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

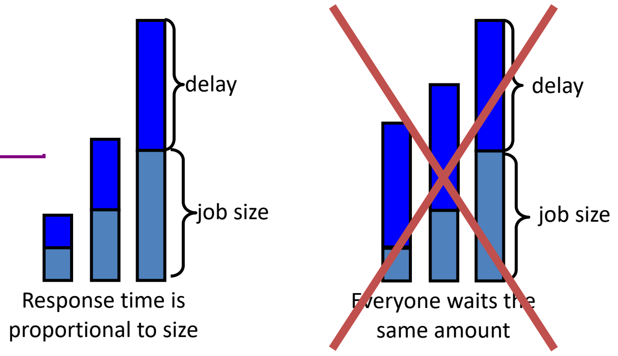
E[Tq] uguale per tutti!

Non valutato più sulla media, bensì sulla richiesta 'x' precisa.
Nella formula sopra, non ho tempo risposta medio per TUTTI, ma per tutti quelli che chiedono esattamente 'x'.

Errore grave sommare E[S] invece di 'x', perchè E[S] è la media!

14

WHAT IS FAIR?



Define the slowdown for a job of size x as $E(sd(x)) = \frac{E(T_S(x))}{x}$

Rallentamento che subisce un job rispetto alla PROPRIA size

Slowdown for jobs of size x

per scheduling astratti non preemptive

Def.

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

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

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

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

job piccoli tendono ad avere slowdown più grande!

Response time:
ho 10k job, prendo ciascun tempo da quando un job arriva a quando esce, sommo i tempi, e divido per 10k. Dentro questa media pesano molto i tempi di risposta grandi, influiscono maggiormente. Con molti job piccoli, e pochi job grandi, il tempo di risposta rappresenta maggiormente le prestazioni di questi job grandi.

Slowdown: è più significativo per i job piccoli, che sono anche la maggior parte dei job.

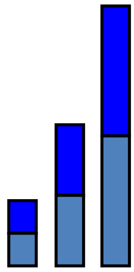
Cosa ci rappresentano questi indici?

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

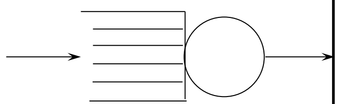


17

Processor sharing

perchè lo vorrei 'fair'

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.

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

Nella PS, quando arriva nuovo job, devo rischedulare per fare in modo che anche questo nuovo job possa essere subito in esecuzione, con mu = 1/(n+1)

18

Anche senza conoscere size dei job, io do un pò di servizio a tutti, in modo che i job piccoli possano uscire più velocemente dal centro.

Processor sharing

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 $E(sd)$!

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

A sinistra abbiamo distribuzioni generali, quindi potremmo avere variabilità molto alta.

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

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

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

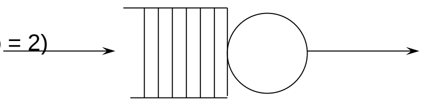
perchè nella Fifo le prestazioni degradano (anche Pareto), in PS si mantengono al livello dell'esponenziale.

the M/G/1/PS queue is insensitive to the variability of the service time distribution, G

Esistono particolari sequenze in cui potrebbe andare peggio!

2 arrivi simultanei che richiedono 1 s di servizio

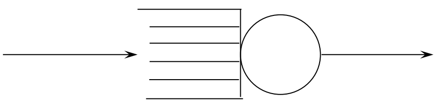
il primo arriva, ci mette 1 con sistema vuoto,
il secondo ci mette 2 (1s di attesa, 1s di servizio = 2)



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

$E(T_S)$?

Entrambi arrivano, vanno in servizio insieme, ricevono metà della capacità operativa, ovvero 0.5, quindi ci mettono 2s



$E(T_S)=2$ s

PS può essere peggio su alcune sequenze di job!

Processor sharing

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

E[S] = x perchè
sto vedendo
i job che chiedono x

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

indipendente dalla size,
infatti dividendo per 'x'
si toglie 'x'.

all jobs have same slowdown: PS as “FAIR” scheduling

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

non preemptive!

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

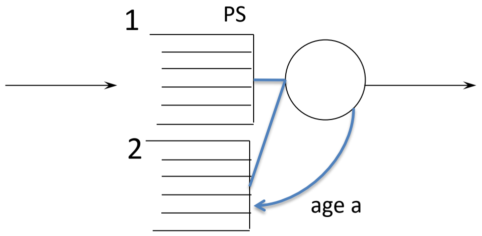
But how can we give preference to the smaller jobs if we don’t know job size?

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

(heavy tail: leggere par. 20.7 !)

Scheduling in Unix:
Foreground-Background scheduling



Lo scheduling di Unix prevede due code distinte in PS. Il sistema parte dalla prima coda per un certo tempo. Un job nella prima coda, superato un periodo 'age a' viene mandato nella coda due.
Il sistema continua a servire i job in coda uno. Il sistema serve quindi i job di age<a, che è quindi un parametro critico. Anche il tempo di servizio tra le due code è un parametro
che può essere definito. In questo modo tutti i job piccoli se ne vanno.