Performance Modeling of Computer Systems and Networks

Prof. Vittoria de Nitto Personè

Analytical results

KP further results

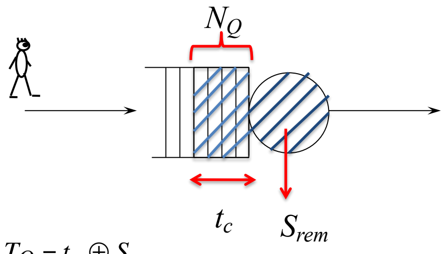
Università degli studi di Roma Tor Vergata

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



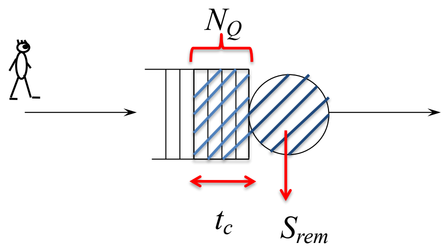
$T_Q = t_c \oplus S_{rem}$

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



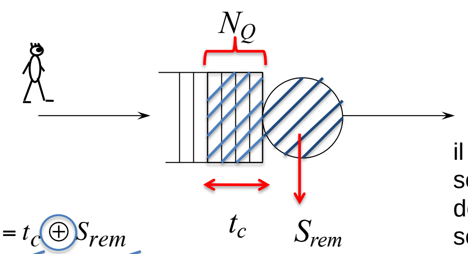
$$E(S_{rem}) = \frac{\lambda}{2}E\left(S^2\right)$$

caratteristica unica dell'esponenziale

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

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

The waiting time and the remaining service time



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

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)

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

legato a Tc

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

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

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

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

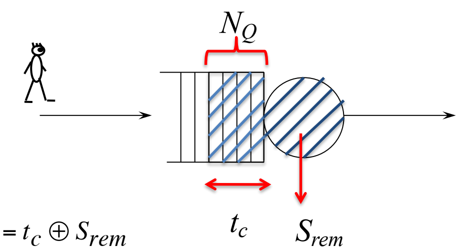
$$= \frac{\lambda E(S)}{2(1-\rho)}\left[\frac{E(S^2)}{E(S)^2}-1+1\right]E(S) =$$

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

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

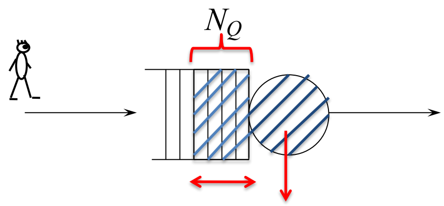


$T_Q = t_c \oplus S_{rem}$

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



$T_Q = t_c \oplus S_{rem}$

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

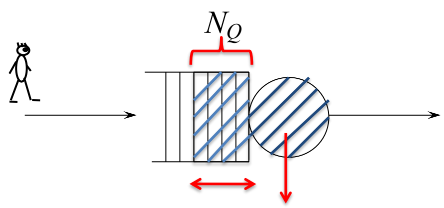
$E(S_{rem})$

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?

Analytical models
M/G/1

The waiting time and the remaining service time



$T_Q = t_c \oplus S_{rem}$

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

$E(S_{rem})$

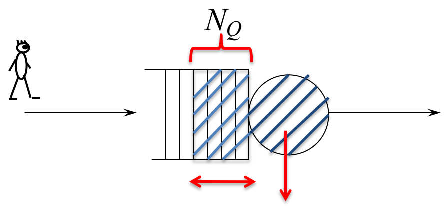
job che arriva prende servizio subito

$E(S_{rem}) = 0$ ⇒ 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.

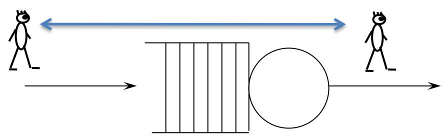
The waiting time and the remaining service time



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

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

The response time



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

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

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.

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

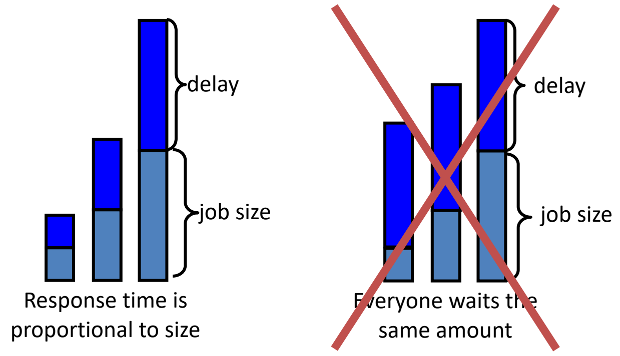
[1]Richard Conway, William Maxwell, and Louis Miller, Theory of Scheduling Addison-Wesley Publishing Company, Inc., 1967. Chapter 8

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

WHAT IS FAIR?



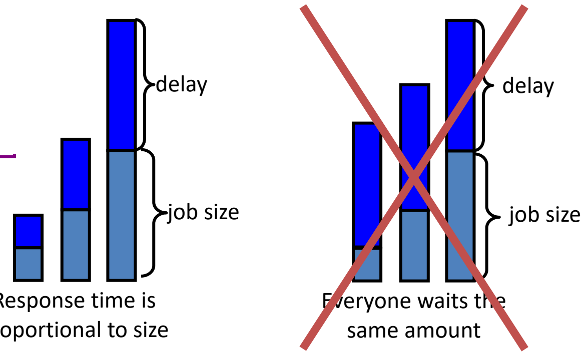
13

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

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

14

WHAT IS FAIR?



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

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

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

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

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

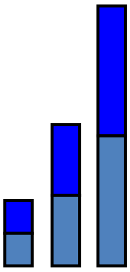
16

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.)

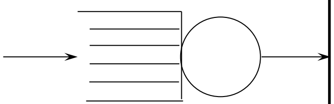
Slowdown tends to be representative of the performance of most jobs – because it is dominated by the performance of the large number of small jobs.

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



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.

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

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 $E(sd)$!

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

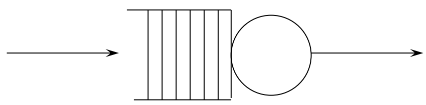
$$E(N_S)^{M/G/1/PS} = \frac{\rho}{1-\rho} = E(N_S)^{M/M/1/FIFO}$$

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

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

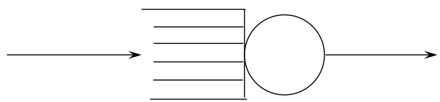
the M/G/1/PS queue is insensitive to the variability of the service time distribution, G

2 arrivi simultanei che richiedono 1 s di servizio



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

E(T_S) **?**



E(T_S)=2 s

PS può essere peggio su alcune sequenze di job!

Processor sharing

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

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

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

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

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

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

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