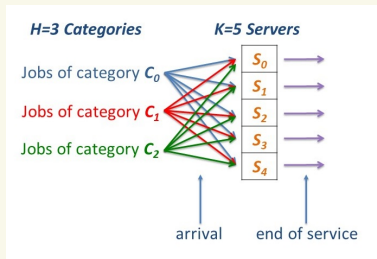


PROBLEMA: simulare lo scheduling di job in un sistema multiserver.
tramite la *Simulazione Discreta a Eventi*

Parametri del sistema:

- K server: S_0, S_1, \dots, S_{K-1}
- H categorie di job: C_0, C_1, \dots, C_{H-1}
- Tempi di interarrivo dei job di C_r : *distribuzione esponenziale di parametro $\lambda_r^{\text{arrival}} > 0$ (media $1/\lambda_r^{\text{arrival}}$)*.
- Tempi di servizio dei job di C_r : *distribuzione esponenziale di parametro $\lambda_r^{\text{service}} > 0$ (media $1/\lambda_r^{\text{service}}$)*.
- Scheduling policy.



Progetto di Programmazione Facoltativo

Quando un nuovo job J arriva, viene assegnato a un server S selezionato in base a una specifica **scheduling policy**. Se S è occupato, J è messo in attesa nella coda FIFO associata ad S .

Eventi simulati:

- Arrivo di un job J nel sistema
- Fine dell'esecuzione di un job J in un server S

Metriche di interesse: Sia A l'insieme dei primi N job arrivati.

- **End time $ET(A)$:** ultimo tempo di fine esecuzione di un job di A
- **Average Queuing Time $AQT-all(A)$:** tempo medio di attesa in coda di un job di A
- **Average Queuing Time $AQT(A,r)$** tempo medio di attesa in coda di un job di A di categoria C_r .

Progetto di Programmazione Facoltativo

La simulazione consiste in una sequenza di iterazioni che simulano i vari eventi nell'ordine temporale in cui occorrono, utilizzando una priority queue Q di appoggio, le cui entry sono coppie $(t(e), e)$, dove e è un evento e $t(e)$ (la chiave) è il tempo di occorrenza di quell'evento.

Inizializzazione: inserisci in Q gli arrivi dei primi job di ciascuna categoria (stessa distribuzione dei tempi di interarrivo)

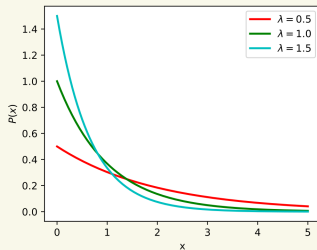
Generica iterazione:

- Estrai da Q l'evento e con $t(e)$ minimo.
- Simula e . In funzione della tipologia di evento, si generano eventuali altri eventi da inserire e si aggiornano le metriche di interesse.

Simulazione di un evento $e = (t(e), e)$:

- Arrivo di un job J di categoria C_r :
 - Si aggiunge a Q l'arrivo del prossimo job di C_r al tempo $t(e) + X$, con X valore random con distribuzione $\text{esponenziale}(\lambda_r^{\text{arrival}})$.
 - Si sceglie un server S per J secondo la scheduling policy. Se S è libero, si mette in esecuzione J in S aggiungendo a Q la fine dell'esecuzione di J al tempo $t(e) + Y$, con Y valore random con distribuzione $\text{esponenziale}(\lambda_r^{\text{service}})$. Altrimenti si inserisce J nella coda FIFO di S .
- Fine esecuzione di un job J di categoria C_r nel server S :
 - Se la coda FIFO di S non è vuota, si mette in esecuzione in S il primo job J' nella coda e si aggiunge a Q la fine della sua esecuzione (come descritto sopra).

Distribuzione Esponenziale: density function



Un valore random X con distribuzione **esponenziale**(λ) è generato così:

$$X = -\frac{1}{\lambda} \ln(1 - \alpha)$$

con α valore random in $[0, 1)$ con distribuzione uniforme, e \ln logaritmo naturale.

Il valore α può essere generato con il metodo `nextFloat` della classe `Random` di `java.util` (usando un seed diverso per ognuno dei **2H** generatori richiesti).

Progetto di Programmazione Facoltativo

Scheduling policy: la scheduling policy da usare è indicata da un parametro P passato in input:

- ($P = 0$, policy obbligatoria): l' i -esimo job arrivato, con $1 \leq i \leq N$, viene schedulato per l'esecuzione nel server S_x , dove

$$x = i \bmod K$$

- ($P = 1$, policy facoltativa): l'obiettivo è di ridurre $ET(A)$

Grading:

- 2 punti: basta la policy obbligatoria, ma il programma deve essere corretto ed efficiente
- 1 punto extra: con la policy facoltativa, mantenendo correttezza ed efficienza.

Progetto di Programmazione Facoltativo

Input:

- Command-line argument: nome di un file di testo contenente i parametri di input.
- Nel file di testo:
 - Linea 1: K, H, N, R, P separati da virgole
 - Linea $2 + r$, $0 \leq r < H$: $\lambda_r^{\text{arrival}}$, $\lambda_r^{\text{service}}$, $\text{seed}_{r, \text{arr}}$, $\text{seed}_{r, \text{ser}}$ separati da virgole

Output:

- K, H, N, R, P separati da virgole
- Se $R = 1$, $N \leq 10$ e $P = 0$, le triple t_e, s_e, c_e , ordinate per t_e , per ciascun evento (t_e, v_e) relativo a un job J , dove: s_e = service time di J , se e = fine esecuzione di J , e 0 altrimenti; e c_e = la categoria di J .
- ET(A) mediato su R run.
- AQT-all(A) mediato su R run.
- $N_r, \text{AQT}(A, r), \text{average service time per } C_r$, mediati su R run.

ESEMPIO: K=2 Server e H=3 Categorie

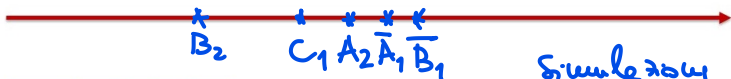
	Arrivo i-esimo job, $i=1,2,\dots$	Fine esecuzione i-esimo job, $i=1,2,\dots$
Categoria 0	A_i	\bar{A}_i
Categoria 1	B_i	\bar{B}_i
Categoria 2	C_i	\bar{C}_i



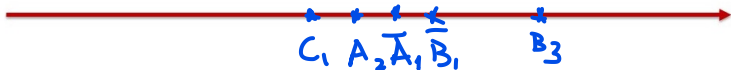
	S0	S1
Esecuz.		
Coda		



	S0	S1
Esecuz.	B_1	
Coda		



	S0	S1
Esecuz.	B_1	A_1
Coda		



	S0	S1
Esecuz.	B_1	A_1
Coda	B_2	



	S0	S1
Esecuz.	B_1	A_1
Coda	B_2	C_1



	S0	S1
Esecuz.	B_1	A_1
Coda	B_2 A_2	C_1

Simulazione con A_2



	S0	S1
Esecuz.	B_1	C_1
Coda	B_2 A_2	


* * * *

\overline{B}_2 A_3 B_3 \overline{C}_1


	S0	S1
Esecuz.	B_2	C_1
Coda	A_2	

	S0	S1
Esecuz.		
Coda		


	S0	S1
Esecuz.		
Coda		




	S0	S1
Esecuz.		
Coda		




	S0	S1
Esecuz.		
Coda		




	S0	S1
Esecuz.		
Coda		



	S0	S1
Esecuz.		
Coda		



	S0	S1
Esecuz.		
Coda		



	S0	S1
Esecuz.		
Coda		