UNIVERSITÉ CÔTE D'AZUR

L2 Informatique, 14 mars 2022

TP-1 Réseaux : Simulation de la file M/M/1

File M/M/1 :

- arrivées poissonniennes de taux λ
- durée de service exponentielle de taux μ
- 1 seul serveur
- buffer d'attente de taille infinie
- discipline de service FIFO

1 Objectifs pédagogiques

L'objectif de ce TP est de simuler l'évolution du nombre de clients dans une M/M/1 en utilisant le langage de programmation `Python`. Nous étudierons l'influence du facteur de charge $\rho = \lambda/\mu$. Nous caractériserons également la distribution de probabilité du délai de traversée du système (attente + service) dans le cas de la M/M/1.

2 Consignes

General : Lisez TOUT l'énoncé avant de taper une seule ligne de code.

Evaluation : A la fin de chaque séance, il y aura une evaluation QCM sur Moodle d'une durée de 15 minutes sur le sujet du TP.

3 Quelques formules

En plus des formules déjà étudiées dans le CM, vous aurez besoin des suivantes afin de réussir ce TP.

Probabilité que le temps T entre deux arrivées consécutives soit au moins t

$$\mathbb{P}(T > t) = e^{-\lambda t} \tag{1}$$

Probabilité que le temps T jusqu'à le prochain depart (c.-à.-d., la durée de service) soit au moins t

$$\mathbb{P}(T > t) = e^{-\mu t} \tag{2}$$

Probabilité que le temps T entre deux événements consécutifs (quel que soit l'événement) soit au moins t

$$\mathbb{P}(T > t) = e^{-(\lambda+\mu)t} \tag{3}$$

Probabilité que le prochain événement soit une arrivée

$$\mathbb{P}(arrivee) = \frac{\lambda}{\lambda + \mu} \tag{4}$$

Probabilité que le prochain événement soit un départ

$$\mathbb{P}(depart) = \frac{\mu}{\lambda + \mu} \tag{5}$$

4 Simulation à Evénements Discrets

La Simulation à Evénements Discrets (SED) nous permet de modéliser des buffers des noeuds des commutation et de simuler leurs fonctionnements. La SED est un type de simulation qui fait progresser l'horloge en étapes discrètes, souvent de taille irrégulière. Ces étapes discrètes correspondent à l'intervalle de temps ente deux événements consécutifs. Donc, la simulation est dirigée par des événements :

— A chaque étape, l'horloge avance vers l'événement suivant planifié dans une file d'attente d'événements, et l'événement est traité.
— Étant donné que ce sont les événements les seuls qui peuvent entraîner un changement de l'état de la simulation, il n'y a aucun intérêt à faire avancer l'horloge dans des pas de temps plus petits que les intervalles entre les événements.

Les simulations à événements discrets sont utilisées de façon classique pour modéliser des problèmes d'accès concurrents à une ressource partagée, c.-à-d. les problèmes des files d'attente. Dans notre cas, lorsque plusieurs paquets de données (les clients de notre système) essaient d'accéder à la bande passante (la ressource partagée) du lien de sortie. Donc, on aura affaire à deux types d'événements :

— **Arrivée** du paquet au système et entrée dans la file d'attente (suivi, si possible, par le début du service).
— **Départ** du paquet du système, après la fin du service (suivi, si possible, par le début du service d'un paquet mis en attente au préalable dans le buffer).

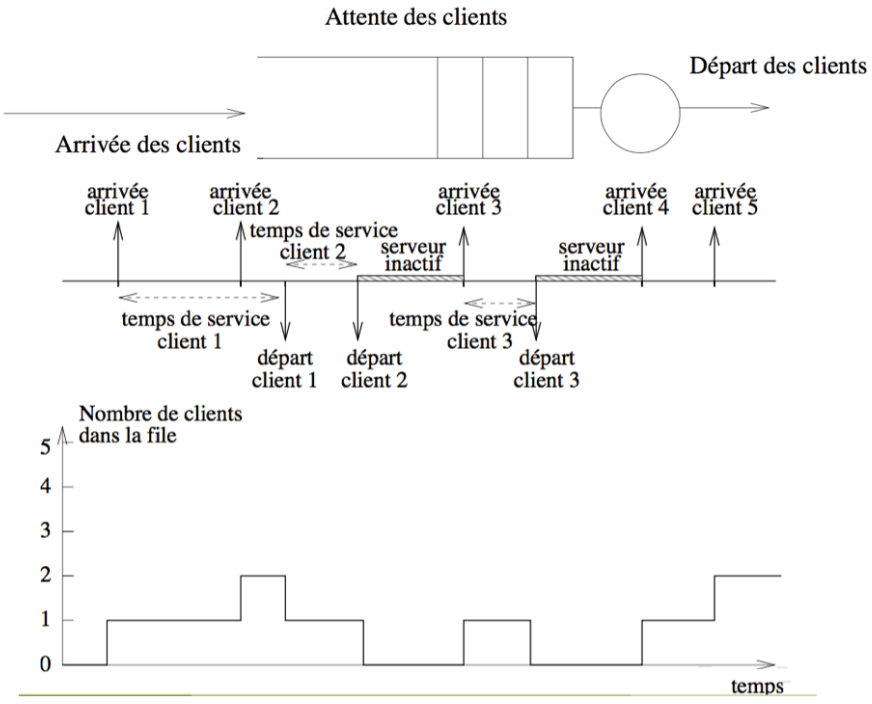

FIGURE 1 – Modèle de file d'attente

5 Le lien de sortie d'un noeud comme une file M/M/1

— Les paquets de données correspondent aux clients.
— Le serveur (ou service) corresponde à un lien de transmission avec un débit r (bps) par lequel des paquets de données sont transmis

— Le temps de service moyen (t_s) correspond à la durée de transmission (d_{trans}) sur le lien de sortie : t_s = s (bits) / r (bps) = 1 / μ (s).
— Le file d'attente est infinie.
— Le processus des arrivées de paquets est un processus de Poisson de paramètre λ. Cela veut dire que la durée des inter-arrivées (temps entre deux arrivées consécutives) suit une loi exponentielle de paramètre λ, où λ représente le nombre moyen d'arrivées par unité de temps (unité : paquets/seconde) et, en consequence, la durée moyenne des inter-arrivées vaut 1/ λ (unité : seconde).
— La durée de service suit une loi exponentielle de paramètre μ, où μ représente le nombre moyen de départs (terminaisons de services) par unité de temps (unité : paquets/seconde) et, en consequence, la durée moyenne de service vaut 1/ μ (unité : seconde).

6 Programme `Python`

Ecrivez un programme `Python` qui simule l'évolution au cours du temps du nombre N de clients dans le système, en suivant les étapes proposées. En bref, le programme aura une boucle principale pour avancer d'un événement vers le suivant (*simulation à événements discrets*). A chaque iteration de la boucle, il faudra faire deux tâches : (i) d'abord, identifier la nature du prochain événement (arrivée ou départ), et (ii) ensuite, calculer le temps jusqu'au prochain événement. Mais, vous avez deux manières de les implémenter : (i) soit vous avez une *liste d'événements à venir* (où les prochains événements, arrivés et departs, sont connus à l'avance), (ii) soit vous n'avez pas cette liste et vous simplement considérez que le prochain événement est un événement arbitraire non identifié (arrivée ou depart). L'approche avec *liste d'événements à venir* est, peut-être, plus intuitive mais plus longue à programmer. Sentez-vous libres pour essayer la première ou la deuxième approche.

6.1 Approche avec *liste d'événements à venir*

Dans cette approche, on a deux *listes* qui constituent la base du simulateur : (i) la *liste des événements* (arrivées et departs) ; et (ii) la *file d'attente* où on stocke les paquets en attente d'accéder au service (dans notre cas, le lien de sortie du noeud)
Dans la boucle principale, à chaque iteration, on avance jusqu'au prochain événement (arrivée ou départ). Donc, la boucle doit parcourir la *liste des événements* Notez que deux situations se posent en fonction de type d'événement (arrivée ou départ) qui est le premier dans la liste. Si l'événement est une arrivée de paquet, on doit planifier **deux** événements :

— *la prochaine arrivée :* on sait que les interarrivées suivent une loi exponentielle avec un temp moyen de 1/λ seconds, on peut calculer le moment d'arrivé du suivant client et le mettre dans la *liste des événements*.
— *la sort du client arrivant* : Deux cas apparaissent en fonction de l'état du système (s'il y a déjà des clients en attente) :
  — Si le système est vide (il n'y aucun client dans le lien de sortie ni implicitement dans le buffer) : le paquet qui arrive peut accéder directement au lien de sortie sans besoin d'attendre dans la buffer. On planifie le depart de ce paquet (loi exponentielle avec un temp moyen de 1/μ seconds) et on le met dans la *liste des événements*.
  — Si le système n'est pas vide (au moins, il y a un client dans le service) : on met le client qui vient d'arriver dans la file d'attente. Aucun événement *départ* est planifié puisque le paquet ne peut pas encore accéder au service (lien de sortie).

6.2 Approche sans *liste d'événements à venir*

Dans ce cas, on n'a pas besoin d'avoir une liste d'événements où on programme des arrivés ou des departs. Simplement, on progresse d'un événement arbitraire (arrivée ou départ) vers le suivant événement (arrivée ou départ) arbitraire

6.2.1 Fonctionnes auxiliaires

Le temps jusqu'au prochain événement (arrivée, départ ou événement arbitraire) suit une loi exponentielle de paramètre λ, μ ou $\lambda + \mu$, respectivement, comme on a vu avant. Programmez une fonction auxiliaire de nom `temps_suivant` qui calcule ce temps-là à partir du paramètre d'une loi exponentielle.
Ecrivez une une fonction auxiliaire de nom `evenement` qui determine si le prochain événement est une arrivée ou un départ à partir de λ et μ. Vous devez utiliser les formules de la probabilité que le prochain événement soit une arrivée (ou un départ). Si l'événement es une arrivée, le fonction doit retourner un '1'. Si l'événement es un depart, le fonction doit retourner un '-1'.

6.2.2 Boucle principal

Ecrivez le boucle principal de la simulation. Dans chaque iteration, on avance jusqu'au prochain événement (arrivée ou départ). Donc, le boucle doit calculer a chaque fois le temps jusqu'à le prochain événement et determiner son type (arrivée ou départ). Attention, si on est dans la toute première itération, le prochain événement est forcement une arrivée. La simulation doit garder la trace de l'état du système, c.-à.-d., de l'évolution au cours du temps du nombre N de clients dans le système (p. ex. avec deux listes `Python` : une pour les instants temporels où se passe quelque chose ; et, une autre pour le nombre N de clients à ces instants). Tracer un graphe représentant cette évolution.

Question 1 : Tester l'influence du facteur de charge $\rho = \lambda/\mu$. Prendre une valeur de $\rho < 1$ (par exemple, λ = 10 et μ = 20). Puis prendre une valeur de $\rho > 1$. Que se passe-t-il dans chacun des 2 cas ?

6.3 Délais de traversée

Maintenant, on va étudier les délais de traversée des clients, c.-à.-d, le temps total de séjour dans le système de chaque client : temps d'attente dans la file plus temps de service. Pour faire cela, on doit rajouter une nouvelle liste `Python` pour stocker les les délais de traversée de chaque client, c.-à.-d., la difference entre le moment de départ et le moment d'arrivée de chaque client. Ensuite, tracer un histogramme des délais de traversée (prendre le cas d'une file stable, c.-à.-d. $\rho < 1$, pour 10000 iterations).

Question 2 : Quelle est la fonction de densité de probabilité qui suit les délais de traversée (de séjour) des clients ? Quel est le délai moyen ? Est-ce qu'il corresponde à la valeur prevue dans le CM ?

7 Etude du temps moyen de traversée du système en fonction de ρ

Evaluer, à l'aide de vos simulations `Python`, le temps moyen de traversée du système. Tracer ensuite la valeur du délai moyen de traversée en fonction du facteur de charge ρ (ρ varie entre 0 et 1, μ est fixé)

Question 3 : Si on compare l'allure de la courbe avec la formule du délai de traversée du système, a quoi correspondent la region de la courbe lorsque $\rho << 1$ et la region de la courbe lorsque $\rho \to 1$?