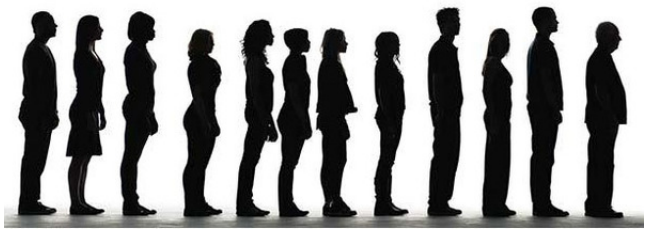
Institut national de statistique et
d'économie appliquée



Simulation of a bank branch made up of two counters

M'chichi Boutayna

10 octobre 2021

MOTS CLÉS

- Simulation
- Réseaux
- Phénomènes aléatoires
- Files d'attentes
- Performance
- Python
- Horizon

TABLE DES MATIÈRES

LISTE DES FIGURES

PRÉAMBULE

Le développement de la théorie des files d'attentes remonte à plus d'un siècle. Initialement, le concept a été examiné dans le but de maximiser les performances du téléphone des centres d'opérations. Cependant, on s'est rendu compte assez tôt que les problèmes dans ce domaine qui étaient solubles à l'aide de modèles mathématiques pourraient survenir dans d'autres domaines de la vie quotidienne ; en l'occurence les services financiers et sanitaires.

Les modèles mathématiques, qui, servent à décrire certains phénomènes, correspondent assez souvent entre eux, quel que soit le domaine spécifique pour lequel ils ont été développés à l'origine : que ce soit les centres d'opérations téléphoniques, la planification et la gestion des services médicaux d'urgence, la description de l'ordinateur l'exploitation, les services bancaires, les systèmes de transport ou d'autres domaines.

Les caractéristiques communes dans ces domaines est que les demandes et les services se produisent avec des contenus variés en fonction des questions posées.

Au cours de la modélisation, même si la signification de la demande et du service dans le système modélisé peut changer, on ne traite que des moments et des intervalles de temps. Ainsi peut-on conclure que, malgré la diversité des problèmes, un bagage théorique en stochastique en s'appuyant d'une boîte à outils mathématique assure une résolution efficace des différents problèmes auquels on peut faire face.
Il convient de noter comme un aspect intéressant que le début du développement de la théorie des files d'attente est étroitement liée à l'apparition des centres d'exploitation téléphonique il y a plus d'un siècle, comme on a décrit précédemment.
Néanmoins, il joue toujours un rôle important dans la planification, la modélisation et l'analyse des réseaux de télécommunications complétés par des méthodes de simulation de pointe et procédures.

CHAPITRE 1

INTRODUCTION

Les files d'attentes font un phénomène incontournable dans notre vie quotidienne :

- faire la queue pour se vacciner (à la lumière du contexte pandémique que nous vivons) ;
- faire la queue pour se servir ;

Donc,faire la queue est une facette omniprésente dans la vie de chacun d'entre nous. Il s'agit d'un phénomène relevant du verbe « attendre » ; ce qui n'est pas- par nature humaine - sympathique.

Par ailleurs, la suppression totale de l'attente n'est pas une possibilité envisageable, ni réalisable ; puisque le coût d'installation et le fonctionnement d'un "vaccinodrôme",par exemple, peut être prohibitif.

Nous remarquons donc la nécéssité des disciplines d'ordre mathématiques tel que la Recherche Oppérationnelle (Operations Research) qui s'octroie la lourde responsabilité d'étudier les sytèmes sujets à ce phénomène, et donc, garantir des sytèmes répondant le mieux possible aux besoins humains minimisant ainsi les temps d'attentes des clients au sein d'un système ou un réseaux.

Dans ce rapport, nous nous attarderons principalement, sur **l'étude d'un réseau de files d'attentes particulier**, comprenant deux guichets dont nous étalerons :

- la formalisation
- l'étude théorique
- la simulation

Et ce,en s'armant principalement des acquis de la simulation à évènements discrets et des compétences en programmation. Nous allons travailler principalement avec le langage de programmation python,notre allié, pour mener à bien cette simulation. Enfin,nous allons terminer notre travail par une examination des performances de ce système.

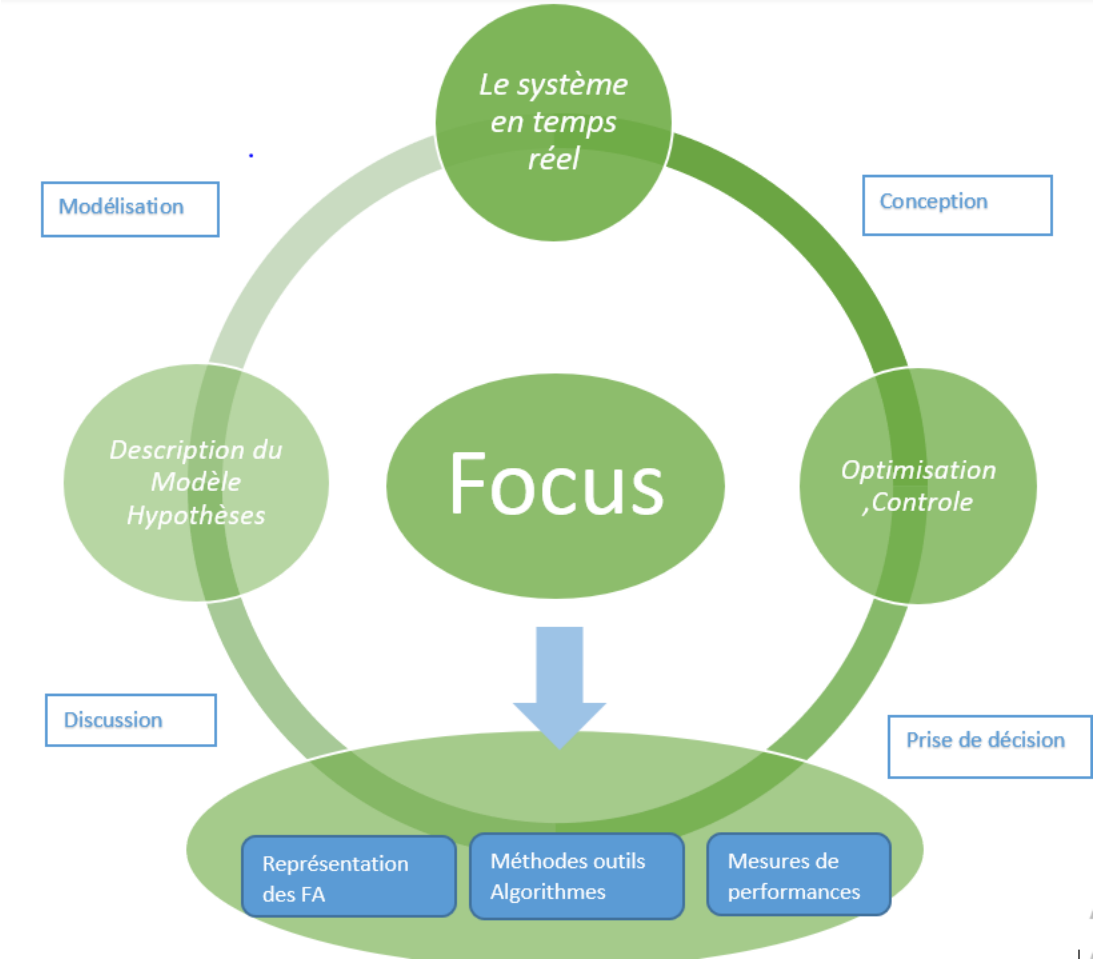


FIGURE 1.1 – Schématisation de la démarche de l'article

CHAPITRE 2

APPROPRIATION DU PROBLÈME

1 Brève description

Considérons un bureau de poste ayant deux guichets, chaque guichet a son propre employé offrant un service particulier en fonction du type du client, ces derniers sont réparties en deux catégories A et B de telle façon que :

- Le client de type A s'engage dans la file d'attente du guichet 1, puis à l'issue de son service, il rejoint la file 2 l'engageant dans le second guichet.
- Le client de type B s'engage dans la file d'attente du guichet 2 en premier lieu, puis à l'issue de son service il rejoint la file l'engageant dans le premier guichet.

Il faut préciser que la capacité des deux files est infinie, et aussi que chaque client qui termine son service, il va rejoindre l'autre file en prenant la dernière place vacante.

Il s'agit d'un réseau ouvert si tout client présent ou entrant dans le système peut le quitter. Il s'agit donc d'un réseau de Jackson composé de n files exponentielles. les files d'attente comportent chacune un ou plusieurs serveurs identiques mj,fournissant des services de durée exponentielle (le taux de service est noté µj. Il est de capacité infnie,en utilisant une discipline de service FIFO.

2 Notations

Gi: Guichets i∈ {1,2}.

Ei : Employés i∈ {1,2}.

$\boldsymbol{\mu_i}$: Taux de service des employés i∈ {1,2}.

$\mathbf{p_1}$: Probabilité de quitter le guichet 1 par le client A.

$\mathbf{1-p_1}$ **:** Probabilité de rejoindre le guichet 2 par le client A.

$\mathbf{p_2}$: Probabilité de quitter le guichet 2 par le client B.

$\mathbf{1-p_2}$ **:** Probabilité de rejoindre le guichet 1 par le client B.

FIFO: Réfère à la discipline de la file acronyme de « **F**irst **I**n **F**irst **O**ut »

C : Représente la capacité de la file dans ce cas-là on prend (C=∞)

$\boldsymbol{\lambda_i}$: Le paramètre du temps inter-arrivé i∈ {1,2}.

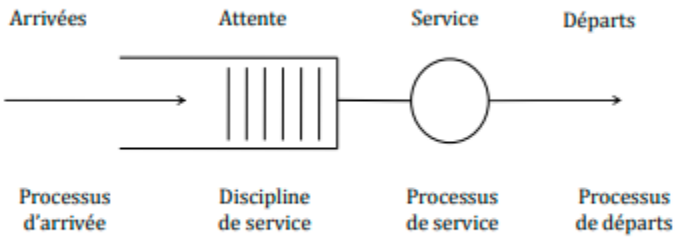


FIGURE 2.1 – Formalisation de la modélisation d'une file d'attente

3 Une image vaut mille mots

L'image ci-dessous illustre le parcours empreinté par chaque type de client en effet :

Le client de type A est matérialisé par les flèches surlignées en turquoise, par ailleurs comme mentionné précédement :

le parcours du client de type A commence par :

1) Il rentrée dans la file 1
2) Il se sert par le guichet E1
3) Il se dirige en prenant la dernière place dans la deuxième file.

Prenons t=0 :
On suppose que le premier client venue est de type A, celui-ci la rentre dans la file et devient A1.

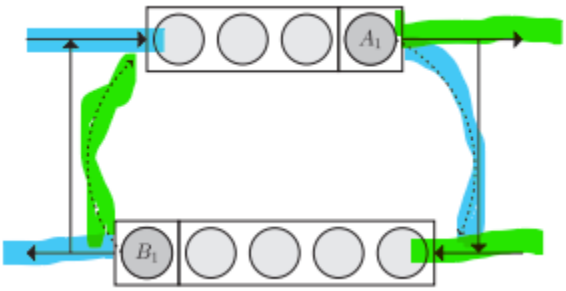
Le client de type B est matérialisé par les flèches surlignées en vert, par ailleurs comme mentionné précédement,le parcours du client de type B commence par :

1) Une rentrée dans la deuxième file B, il se sert par E2.
2) il se dirige prenant la dernière place dans la première file.

Prenons t=0 :

On suppose que le premier client venue est de type B celui là rentre dans la file et devient B1.

Ainsi de suite...



4 Le Logigramme

Ci-dessous le logigramme du système à etudier comprenant la listes des événements retenus.

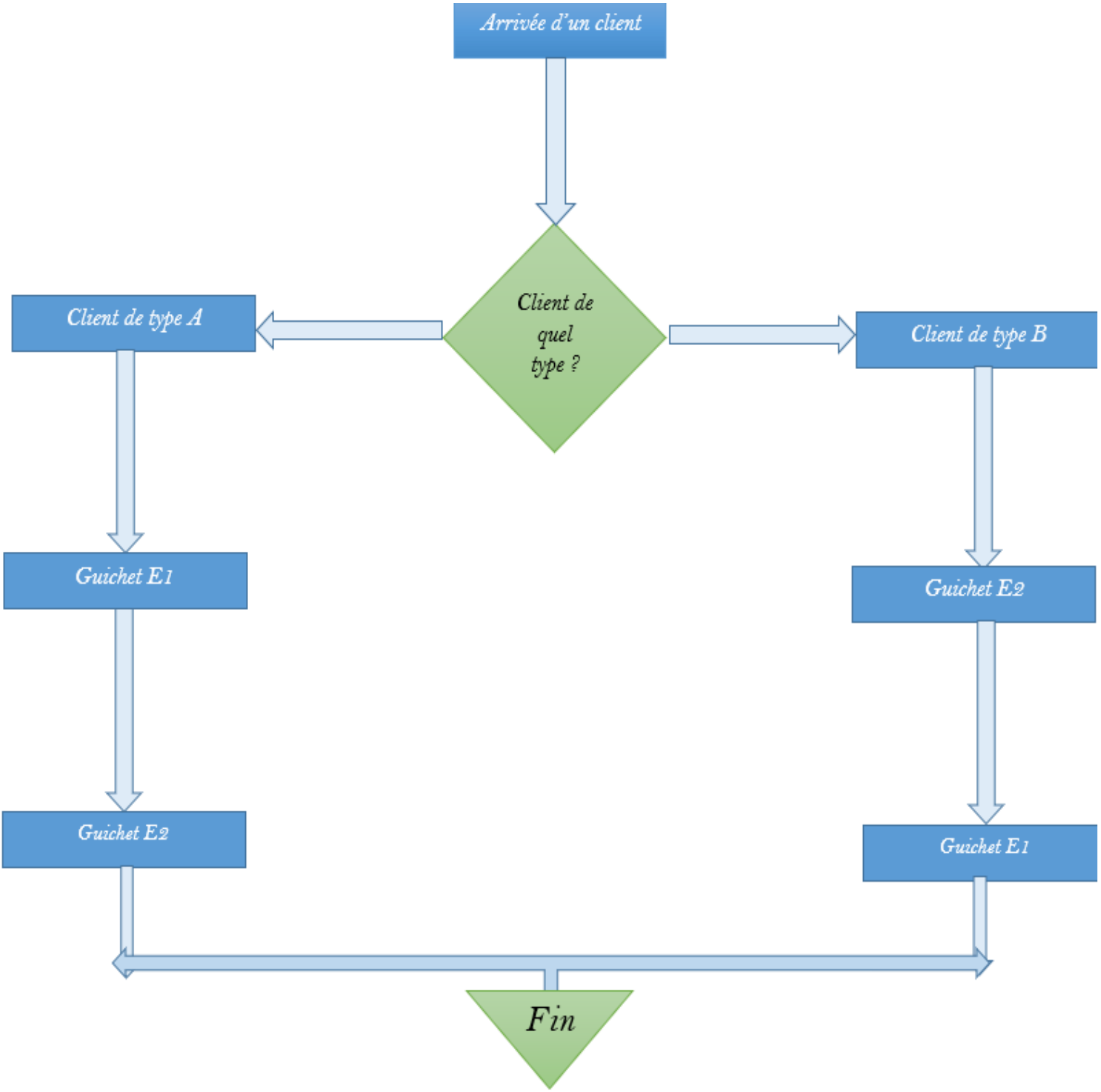


FIGURE 2.2 – Logigramme du système

CHAPITRE 3

FORMULATION

1 Définitions

- Réseaux fermé : si les clients ne peuvent pas quitter le système. il convient de signaler que dans un réseau fermé, le nombre de clients est fixé et ces derniers sont présents dans le système dès le début de son évolution.

- Réseaux ouvert : si tout client présent ou entrant dans le système peut le quitter.

- Réseaux mixte : s'il est ouvert pour certains clients et fermé pour d'autres.

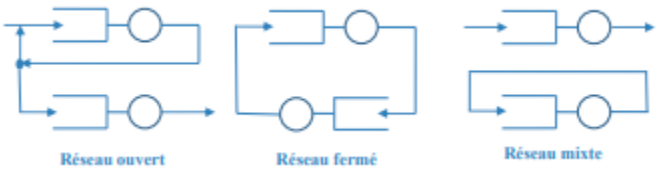


FIGURE 3.1 – Les résaux de files d'attentes

- Réseaux de Jackson : Soit un réseaux de Jackson composée de de n files comportant chacune un ou plusieurs serveurs identiques (mj pour la file j), fournissant des services de durée exponentielle (le taux de service de la file j est noté µj) de capacité infinie,avec une discipline de service FIFO. Les clients (appartenant tous à la même classe) arrivent dans le système selon des processus de Poisson indépendants ;ces conditions sont bien remplies par notre réseau.

- Matrice de probabilité de routage : Dans un réseau de files d'attentes, les clients se déplacent d'une file à une autre avec de probablités définies. On rassemble ces probabilités dans ce qu'on appelle un vecteur des probabilités de sortie.

CHAPITRE 4

SIMULATION

1 Intérêt et perspectives

La simulation à événements discrets repose sur plusieurs techniques qui, lorsqu'elles sont appliquées à la l'étude d'un système dynamique, elle génère des séquences appelées chemins d'échantillonnage qui caractérisent son comportement. La collection comprend :

- Concepts de modélisation pour résumer les caractéristiques essentielles d'un système dans un ensemble cohérent de relations mathématiques entre ses éléments.
- Logiciel spécialement conçu pour convertir ces relations en code exécutable afin de générer les données de chemin d'échantillonnage requise.
- Procédures de conversion de ces données en estimations des performances du système.
- Méthodes pour évaluer dans quelle mesure ces estimations se rapprochent d'un système vrai.

La modélisation de systèmes complexes est devenue un mode de vie dans de nombreux domaines, plus particulièrement dans l'ingénierie, la santé, la gestion, les mathématiques, le militaire, le social, les télécommunications, et sciences des transports. Il fournit un moyen relativement peu coûteux pour recueillir des informations pour la prise de décision. Étant donné que la taille et la complexité des systèmes réels dans ces domaines rend difficile la résolution,la simulation à événements discrets est devenu la méthode de choix. Comme pour tous les outils de recherche scientifique, le succès dans l'application de la simulation à événements discrets dépend de :

- La profondeur et l'étendue du modèle sous-jacent en tant qu'approximation du système
- L'habileté du simulateur à appliquer sa collection de techniques pour percer les mystères du problème.

Pour mettre la simulation à événements discrets dans son contexte en tant qu'outil d'analyse, nous allos décrire la manière dont elle se rapporte à la modélisation en général.

1) La première étape de l'étude consiste à utiliser des connaissances accumulées pour construire un modèle.

2) Un modèle peut être une représentation formelle basée sur théorie ou un compte rendu détaillé basé sur l'observation empirique. Habituellement, il combine les deux.

Un modèle sert à plusieurs fins. En particulier, il :

1. Il permet à un chercheur d'organiser ses croyances théoriques et ses observations empiriques sur un système et en déduire les implications logiques de cette organisation.
2. Il conduit à une meilleure compréhension du système.
3. Il met en perspective le besoin de détail et de pertinence
4. Il accélère la vitesse à laquelle une analyse peut être effectuée
5. Il fournit un cadre pour tester l'opportunité des modifications du système.
6. Il est plus facile à manipuler que le système.
7. Il permet de contrôler plus de sources de variation que l'étude directe d'un système.
8. Il est généralement moins coûteux que l'étude directe du système

2 Présentation du langage python

Le langage de programmation Python a été créé en 1989 par Guido van Rossum, aux Pays-Bas. Le nom Python vient d'une série télévisée Monty Python's Flying Circus dont G. van Rossum est fan. La première version publique de ce langage a été publiée en 1991. La dernière version de Python est la version 3. Plus précisément, la version 3.7 aui a été publiée en juin 2018. La version 2 de Python est désormais obsolète et cessera d'être maintenue après le 1er janvier 2020. Dans la mesure du possible, il faut éviter de l'utiliser. Ce langage est caractérisé par les avantages suivant :

- Interprété
- Portable
- Orienté objet
- Interactif
- Interfacé
- Open source
- Facile à comprendre et à utiliser

On a opté pour python pour sa simplicité et les multiples options qu'il offre notamment une aisance à la visualisation,mais, le talon d'Achille de ce langage reste dans son impuissance a générer des calcul complexes rapidement ce qui n'est pas le cas du langage C. Or,ce dernier à son tour représente des difficultés colossales dans l'élaboration du code.

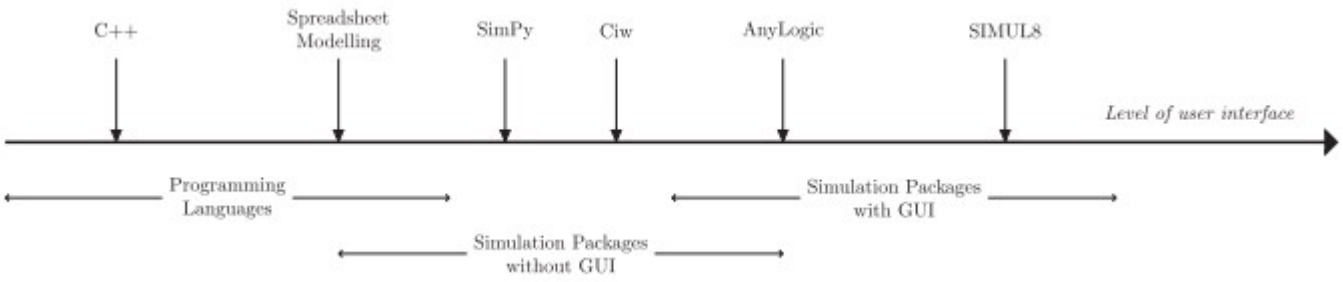


FIGURE 4.1 – Une suggestion de spectre correspondant à l'interface utilisateur, avec un positionnement illustratif pour six options de simulation

3 Bibliothèques utilisées

3.1 NumPy

Acronyme de Numerical Python une librairie open source permettant de manier les tableaux facilement. Elle permet un accès jusqu' à 50 fois plus rapide que les listes, tuples et stocke les valeurs dans un espace mémoire continu.

FIGURE 4.2 – La bibliothèque NumPy

3.2 Pandas

La richesse des fonctionnalités de la librairie pandas est une des raisons, si ce n'est la principale, d'utiliser Python pour extraire, préparer, éventuellement analyser, des données.

FIGURE 4.3 – La bibliothèque Pandas

3.3 Matplotlip

Il s'agit sûrement de l'une des bibliothèques python les plus utilisées pour représenter des graphiques en 2D. Elle permet de produire une grande variété de graphiques et ils sont de grande qualité. Le module pyplot de matplotlib est l'un de ses principaux modules. Il regroupe un grand nombre de fonctions qui servent à créer des graphiques et les personnaliser (travailler sur les axes, le type de graphique, sa forme et même rajouter du texte). Avec lui, nous avons déjà de quoi faire de belles choses.

FIGURE 4.4 – La bibliothèque Matplotlip

3.4 Ciw

Le nom Ciw est le mot gallois pour une file d'attente.

Ciw est une bibliothèque de simulation d'événements discrets pour les réseaux de files d'attente ouverts. Ses fonctionnalités principales incluent la possibilité de simuler des réseaux de files d'attente, plusieurs classes de clients et la mise en œuvre du blocage de type I pour les réseaux restreints.

Un certain nombre d'autres fonctionnalités sont également implémentées, notamment les priorités, le blocage, les horaires et la détection des blocages.

FIGURE 4.5 – La bibliothèque CiW

Remarque :

Ciw est actuellement pris en charge et régulièrement testé sur les versions Python 3.6, 3.7 3.8 et PyPy3.

4 Editeur de code

Jupyter Notebook est la dernière évolution de cet environnement interactif. En effet, avec Jupyter Notebook, on pourra fusionner du code exécutable, du texte, des formules, des images, et des animations dans un seul document Web. Ceci est utile à de nombreuses fins telles queprésentations, tutoriels, débogage, etc.

FIGURE 4.6 – L'éditeur JupyerNotebook

5 Corps du code de la simulation

5.1 Installation importation des bibliothèques

```
Entrée [1]: pip install ciw

Collecting ciw
  Downloading Ciw-2.2.0.tar.gz (93 kB)
Requirement already satisfied: PyYAML>=5.1 in c:\users\lenovo\anaconda3\lib\site-packages (from ciw) (5.4.1)
Requirement already satisfied: networkx>=2.3 in c:\users\lenovo\anaconda3\lib\site-packages (from ciw) (2.5)
Collecting tqdm==4.14.0
  Downloading tqdm-4.14.0-py2.py3-none-any.whl (46 kB)
Requirement already satisfied: decorator>=4.3.0 in c:\users\lenovo\anaconda3\lib\site-packages (from networkx>=2.3->ciw) (5.0.
6)
Building wheels for collected packages: ciw
  Building wheel for ciw (setup.py): started
  Building wheel for ciw (setup.py): finished with status 'done'
  Created wheel for ciw: filename=Ciw-2.2.0-py3-none-any.whl size=66617 sha256=35805edbe2f84fb34f22698c975c65d3309ec0de319243a4
738931c3aa8cc6d6
  Stored in directory: c:\users\lenovo\appdata\local\pip\cache\wheels\ec\d7\72\47a229f40a7b3efc35b9016fc8eb6568bd8650720f955fb1
ec
Successfully built ciw
Installing collected packages: tqdm, ciw
  Attempting uninstall: tqdm
    Found existing installation: tqdm 4.59.0
    Uninstalling tqdm-4.59.0:
      Successfully uninstalled tqdm-4.59.0
Successfully installed ciw-2.2.0 tqdm-4.14.0
Note: you may need to restart the kernel to use updated packages.
```

FIGURE 4.7 – Installation de Ciw

```
Entrée [2]: #importations des bibliothèques
            import pandas as pd
            import numpy as np
            import matplotlib.pyplot as plt
            %matplotlib inline
            import math
            from math import *
            from random import *
            import matplotlib.pyplot as plt
            import numpy as n
            import ciw
```

FIGURE 4.8 – Importation des bibliothèques

5.2 Création d'un réseau de files d'attentes

Avant d'étaler le code de ce réseaux typique, il convient de reprendre le système sujet a l'étude : ici, on remarque qu'il y a deux classes :catégorie A et B de clients différents.

Ces derniers au bout d'un certain moment, ils changent leurs classes. Ciw nous permet de coder ce type de réseau de files d'attentes ce qui est très connu sous sa documentation par **« Dynamic Customer Classes »** .

Par ailleurs, Ciw permet aux clients de changer de classe de manière probabiliste après le service, c'est-à-dire qu'après le service au guichet E1, un client de classe A deviendra de classe B avec probabilité P(C=A|C=B,Ei=E1).
Ces probabilités sont entrées dans le système via le mot-clé **class change matrices**.

Ici la matrice de routage est défini comme suit :

$$\begin{pmatrix} p1 & 1-p1 \\ p2 & 1-p2 \end{pmatrix}$$

Le code utilisé est illustré comme suit :

```
Entrée [26]: N = ciw.create_network(
                 #Classe 0 désigne les clients de type A
                  #Classe 1 désigne les clients de type B
                  arrival_distributions={'Class 0': [ciw.dists.Exponential(8)],
                                         'Class 1': [ciw.dists.Exponential(17)]},
                  service_distributions={'Class 0': [ciw.dists.Exponential(20)],
                                         'Class 1': [ciw.dists.Exponential(30)]},
                 routing={'Class 0': [[1.0]],
                          'Class 1': [[1.0]]},
                 #Vecteur des probailités de changements
                 class_change_matrices={'Node 1': [[0.5, 0.5],
                                                   [0.75, 0.25]]},
                 #Chaque guichets possède un seul employé
                 number_of_servers=[1])
             #ici on a pas mentionné la capacité c'est par défaut infinie en cas de besoin ci dessous la syntaxe munie d'un exemple
             #queue_capacities=[5, float('inf'), float('inf'), 10]
```

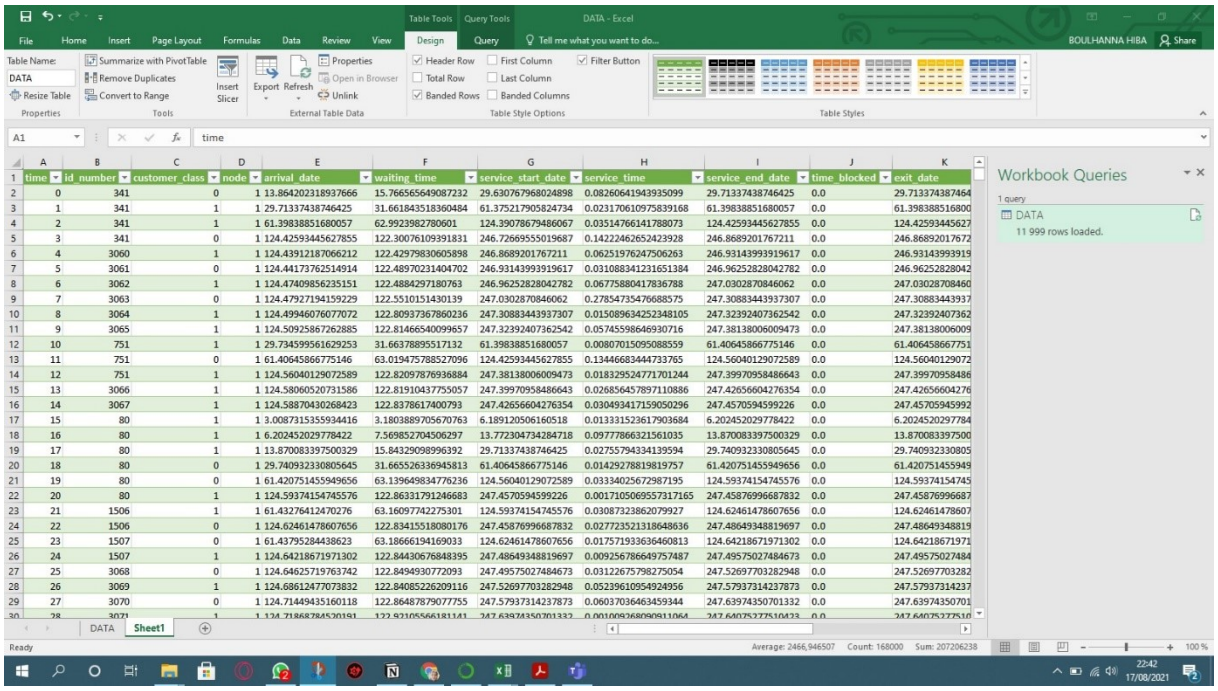
FIGURE 4.9 – Création du réseau des files d'attentes sujet à l'etude

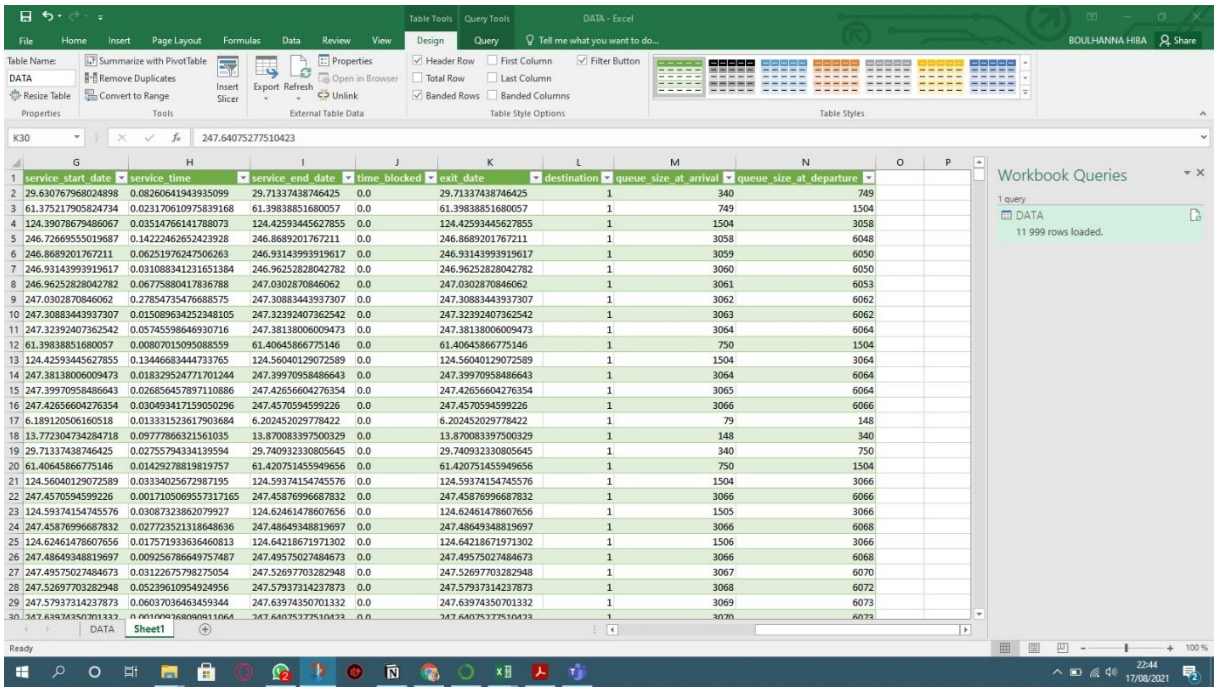
5.3 Lancement de la simulation

```
ciw.seed(1)
Q = ciw.Simulation(N)
#Horizon de simulations
#Q.simulate_until_max_time(10000.0)
Q.simulate_until_max_time(500.0)
```

5.4 Collecte et restitution des output de la simulation

Le listing de la simulation est comme suit :





On remarque la puissance de la bibliothèque CIW en matière des différents évènements retenus. Par ailleurs, la richesse de python nous permettra par la suite de générer assez facilement des données structurées pouvant nous servir dans l'études et le dimensionnement des systèmes sujets à des phénomènes stochastiques.

CHAPITRE 5

EXEMPLES ET DISCUSSIONS

1 Exemple illustratif

Pour cet exemple, et Vu que python prend une durée de temps exhaustive, on s'est limité a un horizon de 500 unité de temps.

2 Calcul des métriques exhibant la performance

Dans cette partie, nous allons etudier la stabilité du réseaux. Pour ce,on va calculer les taux d'ocuppation respectifs des deux files d'attentes via python :

2.1 Stabilité du système

```
Entrée [18]: #taux d'ocupation noté gho
             print("le taux d'ocuppation de la file du guichet 1 est : ",8/20)

             le taux d'ocuppation de la file du guichet 1 est :  0.4

Entrée [19]: print("le taux d'ocuppation de la file du guichet 2 est : ",17/30)

             le taux d'ocuppation de la file du guichet 2 est :  0.5666666666666667
```

FIGURE 5.1 – Stabilité du système

Nous remarquons que les deux taux sont inférieurs à 1 donc **le réseaux est stable.**

2.2 Temps de séjour d'un client dans le système

Méthode : 1

```
Entrée [34]: df_system_times=df["service_end_date"]-df["arrival_date"]

Entrée [35]: df_system_times

Out[35]: 0         15.849172
         1         31.685014
         2         63.027546
         3        122.442986
         4        122.492318
                     ...
         11994    253.133887
         11995    122.330462
         11996    253.149321
         11997    253.236001
         11998    253.224306
         Length: 11999, dtype: float64

Entrée [71]: y=df_system_times.mean()

Entrée [72]: print("Le temps moyen de séjour d un client est :",y)

             Le temps moyen de séjour d un client est : 126.34692848149247
```

Méthode : 2

```
Entrée [68]: x=(df["service_time"]+df["waiting_time"]).mean()

Entrée [70]: print("Le temps moyen de séjour d un client est :",x)

             Le temps moyen de séjour d un client est : 126.34692848149247
```

Comparaison

```
Entrée [77]: print("la différence entre les résultats des deux :")
             print(x-y)

             la différence entre les résultats des deux :
             0.0
```

FIGURE 5.2 – Durée moyenne de séjour d'un client selon deux méthodes

Nous avons réalisé ceci a l'aide de deux méthode qui vont comme suit :
La manipulation du listing par le biais de la bibliothèque pandas nous donne une durée moyenne de : 126.34692848149247 unité de temps

2.3 La durée moyenne de séjour de chaque catégorie dans le système

- Pour un client de type A :

```
Entrée [18]: temp_de_sejour_A=(client_A["service_time"]+client_A["waiting_time"]).mean()

Entrée [19]: temp_de_sejour_A

    Out[19]: 125.74759762946427
```

FIGURE 5.3 – Temps de séjour d'un client de type A

nous avons trouvé : 125.74759762946427 unités de temps

- Pour un client de type B :

```
Entrée [22]: temp_de_sejour_B=(client_B["service_time"]+client_B["waiting_time"]).mean()

Entrée [23]: temp_de_sejour_B

    Out[23]: 126.87547888628559
```

FIGURE 5.4 – Temps de séjour d'un client de type B

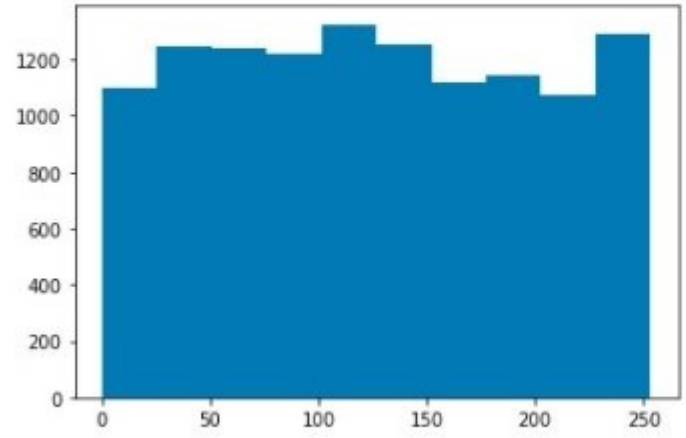
Nous avons trouvé :126.87547888628559 unités de temps

On conclue donc qu'en moyenne, les client de type B séjournent plus que les client de type A dans le système. Ce qui est normal ; puisque le taux d'arivée des clients de type B est plus grand que celui de type A sans oublier que le taux de service des clients de type B est plus grand que celui des clients de type A.

Evaluons le pourcentage d'activité des serveurs : Comme le montre le graphique suivant, la proportion d'activité des serveurs atteint :0.9996392723397263 soit : 99,96 %

```
Entrée [27]: plt.hist(system_times);
             plt.show()

             print ("Pourcentage d'activité des serveurs :  "+str(Q.transitive_nodes[0].server_utilisation))
```




```
Pourcentage d'activité des serveurs :  0.9999639272339726
```

FIGURE 5.5 – Représentation du pourcentage d'activité des serveurs

3 Visualisation de l'évolution du nombre des clients des deux catégories clients

On a prit 4 unités de temps pour visualiser l'évolution des clients.

A l'aide de Matplotlib, on aboutit à ces des représentations :

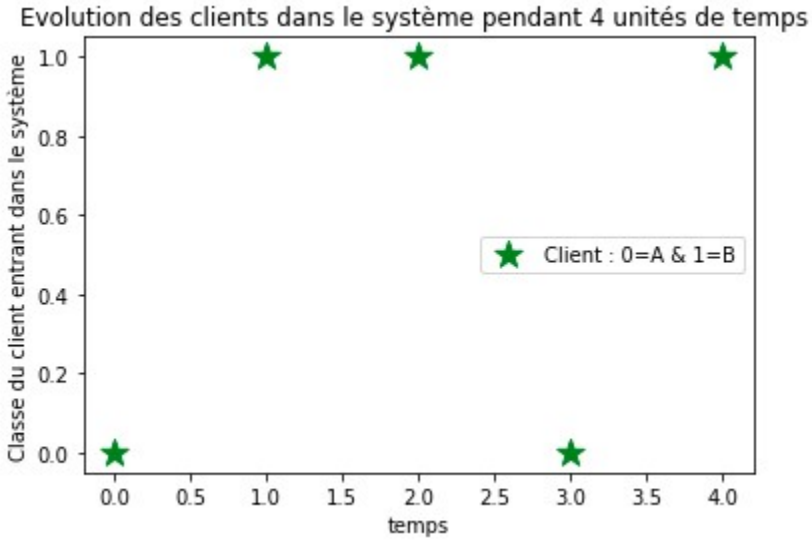


FIGURE 5.6 – Evolution des clients dans le premier serveur

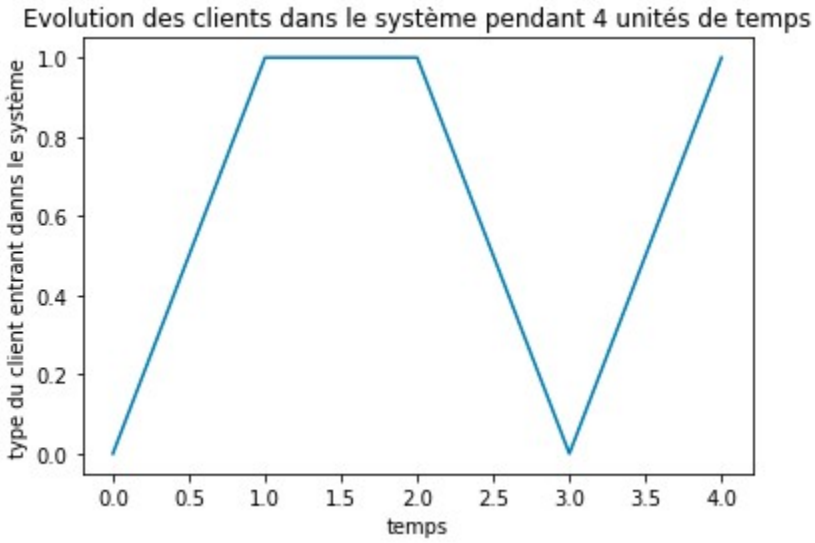


FIGURE 5.7 – Evolution des clients dans le serveur 2

On remarque que pendant 4 unités de temps, les clients de type B viennent plus que ceux de type A, ce qui est expliqué par le taux d'arrivée qui est plus important chez cette catégorie.

CONCLUSION

Au terme de cette étude à la fois théorique et pratique, on a entamé notre étude par un état de l'art de la problématique puis on s'est servi du langage python pour approcher la réalité.

Ceci a été courroné par la génération d'une donnée fiable qui nous a permis par la suite d'évaluer les différentes performances de notre sytème.

Afin de voir toute les documentations et les sources utilisée veuillez vous référer à la bibiliographie.

Enfin,pour avoir une idée claire sur le cheminement de la simulation veuillez voir les annexes présentés à la fin de ce document ; vous trouverez le codes et les étapes du projet commentées.

BIBLIOGRAPHIE

beginitemize[label = •]

- [1] :Introduction to Queueing Systems with Telecommunication Applications
- [2] :Recherche opérationnelle pour ingénieurs, Volume 2
- [3] : Discrete-Event Simulation Modeling, Programming, and Analysis by George S. Fishman
- [4] : Bibliothèque de Ciw [Lien]
- [5] : Tutoriel de Python [Lien]
- [6] :Excel Data Analysis : Modeling and Simulation Hector Guerrero
- [7] :Dynamic Systems : Modeling, Simulation, and Control Craig A. Kluever [Lien]
- [8] :Description de la bibliothèque Ciw avec python [Lien]
- [9] :Ciw, An open-source discrete event simulation library Geraint I. Palmer, Vincent A. Knight, Paul R. Harper Asyl L. Hawa [Lien de l'article]

Annexe A :

1.Importation des bibliothèques

ciw bibliothèque de SED des files d'attentes

pandas pour manipuler le listing

matplotlib pour représentation graphiques

numpy pour les calcules numériques

math réalisation des opérations Arithmétiques

seaborn Bibliothèque de la Data_Visualisation

Entrée [8]:

```
import ciw
import pandas as pd
from pandas import DataFrame
import matplotlib as plt
import numpy as np #Calcules numériques
import matplotlib.pyplot as plt#Représentation Graphiques
%matplotlib inline
import math #Réalisation des opération mathématiques
import matplotlib.pyplot as plt
import seaborn as sns #Bibliothèque de la Data_Visualisation
from collections import Counter
```

2.Création du réseaux des files d'attentes

Classe 0 désigne les clients de type A

Classe 1 désigne les clients de type B

Chaque guichet a un seul employé

La matrice de routage est aussi mentionnée

Ici on a pas mentionné la capacité c'est par défaut infinie en cas de besoin ci dessous la syntaxe munie d'un exemple

queue_capacities=[5, float('inf'), float('inf'), 10]**

***Il convient de noter qu'il existe toutes les autres lois dans la bibliothèques Ciw :**

Entrée [9]:

```
N = ciw.create_network(
    #Classe 0 désigne les clients de type A
     #Classe 1 désigne les clients de type B
     arrival_distributions={'Class 0': [ciw.dists.Exponential(8)],
                            'Class 1': [ciw.dists.Exponential(17)]},
     service_distributions={'Class 0': [ciw.dists.Exponential(20)],
                            'Class 1': [ciw.dists.Exponential(30)]},
    routing={'Class 0': [[1.0]],
             'Class 1': [[1.0]]},
    #Vecteur des probailités de changements
    class_change_matrices={'Node 1': [[0.5, 0.5],
                                      [0.75, 0.25]]},
    #Chaque guichet possède un seul employé
    number_of_servers=[1])
```

3.Lancement de la simulation

Horizon de la simulation est fixé a 10000 unités

Vue que l'exécution prends beaucoup de temps on a décidé de :

On changera cet horizon demandée par 500 unités

Entrée [10]:

```
ciw.seed(1)
Q = ciw.Simulation(N)
#Horizon de simulations
#Q.simulate_until_max_time(10000.0)
Q.simulate_until_max_time(500.0)
```

4.Collecter la Data communément appelé "Listing"

Entrée [11]:

```
DATA= Q.get_all_records()
```

```
DATA
```

Out[12]:

```
[Record(id_number=341, customer_class=0, node=1, arrival_date=13.864202318
937666, waiting_time=15.766565649087232, service_start_date=29.63076796802
4898, service_time=0.08260641943935099, service_end_date=29.7133743874642
5, time_blocked=0.0, exit_date=29.71337438746425, destination=1, queue_siz
e_at_arrival=340, queue_size_at_departure=749),
 Record(id_number=341, customer_class=1, node=1, arrival_date=29.713374387
46425, waiting_time=31.661843518360484, service_start_date=61.375217905824
734, service_time=0.023170610975839168, service_end_date=61.3983885168005
7, time_blocked=0.0, exit_date=61.39838851680057, destination=1, queue_siz
e_at_arrival=749, queue_size_at_departure=1504),
 Record(id_number=341, customer_class=1, node=1, arrival_date=61.398388516
80057, waiting_time=62.9923982780601, service_start_date=124.3907867948606
7, service_time=0.03514766141788073, service_end_date=124.42593445627855,
time_blocked=0.0, exit_date=124.42593445627855, destination=1, queue_size_
at_arrival=1504, queue_size_at_departure=3058),
 Record(id_number=341, customer_class=0, node=1, arrival_date=124.42593445
627855, waiting_time=122.30076109391831, service_start_date=246.7266955501
9687, service time=0.14222462652423928, service end date=246.868920176721
```

5.Restituer la donnée aquise

Comme vous remarquez en haut le listing fourni en une liste de tuples ce qui n'est pas réutilisable par la suite

La bibliothèque pandas nous permettra d'organiser les données dans un tableau format CSV qu'on va convertir en format xls a l'aide Power Query

Entrée [13]:

```
df = pd.DataFrame(DATA, columns =['id_number',"customer_class","node",
                                  "arrival_date","waiting_time","service_start_date",
                                  "service_time","service_end_date","time_blocked",
                                  "exit_date","destination","queue_size_at_arrival",
                                  "queue_size_at_departure"])
```

```
df
```

Out[14]:

	id_number	customer_class	node	arrival_date	waiting_time	service_start_date	service_
0	341	0	1	13.864202	15.766566	29.630768	0.08
1	341	1	1	29.713374	31.661844	61.375218	0.02
2	341	1	1	61.398389	62.992398	124.390787	0.03
3	341	0	1	124.425934	122.300761	246.726696	0.14
4	3060	1	1	124.439122	122.429798	246.868920	0.06
...	...	...	...	...	...	...	
11994	6045	0	1	246.708942	253.118210	499.827151	0.01
11995	3059	1	1	124.396233	122.224179	246.620412	0.10
11996	3059	1	1	246.726696	253.116133	499.842829	0.03
11997	6046	1	1	246.758080	253.117936	499.876016	0.11
11998	6047	1	1	246.773794	253.220287	499.994081	0.00

11999 rows × 13 columns

6. Afficher le Dataframe

```
print(df)
```

```
       id_number  customer_class  node  arrival_date  waiting_time  \
0            341               0     1     13.864202     15.766566
1            341               1     1     29.713374     31.661844
2            341               1     1     61.398389     62.992398
3            341               0     1    124.425934    122.300761
4           3060               1     1    124.439122    122.429798
...          ...             ...   ...           ...           ...
11994       6045               0     1    246.708942    253.118210
11995       3059               1     1    124.396233    122.224179
11996       3059               1     1    246.726696    253.116133
11997       6046               1     1    246.758080    253.117936
11998       6047               1     1    246.773794    253.220287

       service_start_date  service_time  service_end_date  time_blocked  \
0               29.630768      0.082606         29.713374           0.0
1               61.375218      0.023171         61.398389           0.0
2              124.390787      0.035148        124.425934           0.0
3              246.726696      0.142225        246.868920           0.0
4              246.868920      0.062520        246.931440           0.0
...                   ...           ...               ...           ...
11994          499.827151      0.015678        499.842829           0.0
11995          246.620412      0.106283        246.726696           0.0
11996          499.842829      0.033187        499.876016           0.0
11997          499.876016      0.118064        499.994081           0.0
11998          499.994081      0.004019        499.998100           0.0

        exit_date  destination  queue_size_at_arrival  queue_size_at_departu
re
0       29.713374            1                    340                     7
49
1       61.398389            1                    749                    15
04
2      124.425934            1                   1504                    30
58
3      246.868920            1                   3058                    60
48
4      246.931440            1                   3059                    60
50
...           ...          ...                    ...
...
11994  499.842829            1                   6044                   122
85
11995  246.726696            1                   3058                    60
44
11996  499.876016            1                   6044                   122
88
11997  499.994081            1                   6045                   122
90
11998  499.998100            1                   6046                   122
90

[11999 rows x 13 columns]
```

7.Exporter les données

```
df.to_csv ('DATA.csv', index = False , header=True)
df
df.to_csv('DATA.csv')
```

8.Aficher le Dataframe

Entrée [17]:

```
df=df.rename(columns={"Unnamed: 1":"time"})
```

Entrée [18]:

```
df
```

Out[18]:

	id_number	customer_class	node	arrival_date	waiting_time	service_start_date	service_
0	341	0	1	13.864202	15.766566	29.630768	0.08
1	341	1	1	29.713374	31.661844	61.375218	0.02
2	341	1	1	61.398389	62.992398	124.390787	0.03
3	341	0	1	124.425934	122.300761	246.726696	0.14
4	3060	1	1	124.439122	122.429798	246.868920	0.06
...	...	...	...	...	...	...	
11994	6045	0	1	246.708942	253.118210	499.827151	0.01
11995	3059	1	1	124.396233	122.224179	246.620412	0.10
11996	3059	1	1	246.726696	253.116133	499.842829	0.03
11997	6046	1	1	246.758080	253.117936	499.876016	0.11
11998	6047	1	1	246.773794	253.220287	499.994081	0.00

11999 rows × 13 columns

9.Nombre total des clients réparties en deux à l'issue de la simulation

Entrée [19]:

```
Counter([client.customer_class for client in DATA])
```

Out[19]:

```
Counter({0: 5623, 1: 6376})
```

```
print("On a 5623 clients de type A \n On a 6376 clients de types B ")
```

```
On a 5623 clients de type A
 On a 6376 clients de types B
```

10.Etude de la stabilité du système

Entrée [21]:

```
#taux d'ocupation noté gho
print("le taux d'ocuppation de la file du guichet 1 est : ",8/20)
```

```
le taux d'ocuppation de la file du guichet 1 est :  0.4
```

Entrée [22]:

```
print("le taux d'ocuppation de la file du guichet 2 est : ",17/30)
```

```
le taux d'ocuppation de la file du guichet 2 est :  0.5666666666666667
```

On conclue donc que le réseaux est stable puisque les deux taux d'occupation sont inférieurs à 1

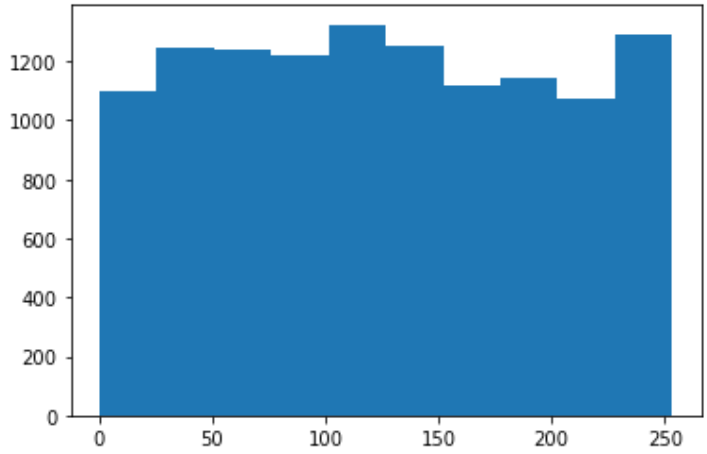
Entrée [26]:

```
system_times=[client.service_end_date-client.arrival_date for client in DATA]
moyenne_system_times=sum(system_times)/len(system_times)
```

12.Représentation du pourcentage d'activité des serveurs

```
plt.hist(system_times);
plt.show()

print ("Pourcentage d'activité des serveurs :  "+str(Q.transitive_nodes[0].server_utilisati
```




```
Pourcentage d'activité des serveurs :  0.9999639272339726
```

13 Calcul de la durée Moyenne de séjour d'un client dans le système

Méthode : 1

Entrée [28]:

```
df_system_times=df["service_end_date"]-df["arrival_date"]
```

Entrée [29]:

```
df_system_times
```

Out[29]:

```
0         15.849172
1         31.685014
2         63.027546
3        122.442986
4        122.492318
            ...
11994    253.133887
11995    122.330462
11996    253.149321
11997    253.236001
11998    253.224306
Length: 11999, dtype: float64
```

Entrée [30]:

```
y=df_system_times.mean()
```

Entrée [31]:

```
print("Le temps moyen de séjour d un client est :",y)
```

```
Le temps moyen de séjour d un client est : 126.34692848149247
```

Méthode : 2

Entrée [32]:

```
x=(df["service_time"]+df["waiting_time"]).mean()
```

Entrée [33]:

```
print("Le temps moyen de séjour d un client est :",x)
```

```
Le temps moyen de séjour d un client est : 126.34692848149247
```

Comparaison

Entrée [34]:

```
print("la différence entre les résultats des deux méthode :")
print(x-y)
print("Puisque la différence est nulle on conclue que les deux méthodes sont equivalentes "
```

```
la différence entre les résultats des deux méthode :
0.0
Puisque la différence est nulle on conclue que les deux méthodes sont equiva
lentes
```

14. Le nombre moyen de clients dans chaque catégorie

Entrée [36]:

```
N_A=(0.4/0.6)
print("le nombre moyen de client dans la file 1 est :",N_A )
```

```
le nombre moyen de client dans la file 1 est : 0.6666666666666667
```

Entrée [37]:

```
N_B=(0.56/0.44)
print("le nombre moyen de client dans la file 1 est :",N_B )
```

```
le nombre moyen de client dans la file 1 est : 1.272727272727273
```

Annexe B :

Entrée [1]:

```
import ciw
import pandas as pd
from pandas import DataFrame
import matplotlib as plt
import numpy as np #Calcules numériques
import matplotlib.pyplot as plt#Représentation Graphiques
%matplotlib inline
import math #Réalisation des opération mathématiques
import matplotlib.pyplot as plt
import seaborn as sns #Bibliothèque de la Data_Visualisation
from collections import Counter
```

Entrée [9]:

```
df=pd.read_csv(r'C:\Users\BOULHANNA\Desktop\INSEA S4\s4\SIMULATION\simulation_projet\output
```

Entrée [10]:

```
df
```

Out[10]:

	time	id_number	customer_class	node	arrival_date	waiting_time	service_start_date
0	0	341	0	1	13.864202	15.766566	29.630768
1	1	341	1	1	29.713374	31.661844	61.375218
2	2	341	1	1	61.398389	62.992398	124.390787
3	3	341	0	1	124.425934	122.300761	246.726696
4	4	3060	1	1	124.439122	122.429798	246.868920
...	...	...	...	...	...	...	...
11994	11994	6045	0	1	246.708942	253.118210	499.827151
11995	11995	3059	1	1	124.396233	122.224179	246.620412
11996	11996	3059	1	1	246.726696	253.116133	499.842829
11997	11997	6046	1	1	246.758080	253.117936	499.876016
11998	11998	6047	1	1	246.773794	253.220287	499.994081

11999 rows × 14 columns

Entrée [11]:

```
df.set_index('time')
```

Out[11]:

time	id_number	customer_class	node	arrival_date	waiting_time	service_start_date	service_
0	341	0	1	13.864202	15.766566	29.630768	0.08
1	341	1	1	29.713374	31.661844	61.375218	0.02
2	341	1	1	61.398389	62.992398	124.390787	0.03
3	341	0	1	124.425934	122.300761	246.726696	0.14
4	3060	1	1	124.439122	122.429798	246.868920	0.06
...	...	...	...	...	...	...	
11994	6045	0	1	246.708942	253.118210	499.827151	0.01
11995	3059	1	1	124.396233	122.224179	246.620412	0.10
11996	3059	1	1	246.726696	253.116133	499.842829	0.03
11997	6046	1	1	246.758080	253.117936	499.876016	0.11
11998	6047	1	1	246.773794	253.220287	499.994081	0.00

11999 rows × 13 columns

Entrée [12]:

```
client_A=df[df["customer_class"]==0]
```

Entrée [13]:

```
client_A
```

Out[13]:

	time	id_number	customer_class	node	arrival_date	waiting_time	service_start_date
0	0	341	0	1	13.864202	15.766566	29.630768
3	3	341	0	1	124.425934	122.300761	246.726696
5	5	3061	0	1	124.441738	122.489702	246.931440
7	7	3063	0	1	124.479272	122.551015	247.030287
11	11	751	0	1	61.406459	63.019476	124.425934
...	...	...	...	...	...	...	...
11987	11987	1505	0	1	124.390787	122.145543	246.536329
11988	11988	1505	0	1	246.572842	253.052003	499.624845
11990	11990	3058	0	1	124.395734	122.177108	246.572842
11992	11992	6043	0	1	246.656917	253.113193	499.770111
11994	11994	6045	0	1	246.708942	253.118210	499.827151

5623 rows × 14 columns

Entrée [18]:

```
temp_de_sejour_A=(client_A["service_time"]+client_A["waiting_time"]).mean()
```

Entrée [19]:

```
temp_de_sejour_A
```

Out[19]:

```
125.74759762946427
```

Entrée [20]:

```
client_B=df[df["customer_class"]==1]
```

```
1 client_B
```

Out[21]:

	time	id_number	customer_class	node	arrival_date	waiting_time	service_start_date
1	1	341	1	1	29.713374	31.661844	61.375218
2	2	341	1	1	61.398389	62.992398	124.390787
4	4	3060	1	1	124.439122	122.429798	246.868920
6	6	3062	1	1	124.474099	122.488430	246.962528
8	8	3064	1	1	124.499461	122.809374	247.308834
...	...	...	...	...	...	...	...
11993	11993	6044	1	1	246.698625	253.102845	499.801470
11995	11995	3059	1	1	124.396233	122.224179	246.620412
11996	11996	3059	1	1	246.726696	253.116133	499.842829
11997	11997	6046	1	1	246.758080	253.117936	499.876016
11998	11998	6047	1	1	246.773794	253.220287	499.994081

6376 rows × 14 columns

Entrée [22]:

```
temp_de_sejour_B=(client_B["service_time"]+client_B["waiting_time"]).mean()
```

Entrée [23]:

```
temp_de_sejour_B
```

Out[23]:

```
126.87547888628559
```

Entrée [26]:

```
Exemple_4_unit_time=df[:5]
```

```
Exemple_4_unit_time
```

Out[27]:

	time	id_number	customer_class	node	arrival_date	waiting_time	service_start_date	servic
0	0	341	0	1	13.864202	15.766566	29.630768	0.
1	1	341	1	1	29.713374	31.661844	61.375218	0.
2	2	341	1	1	61.398389	62.992398	124.390787	0.
3	3	341	0	1	124.425934	122.300761	246.726696	0.
4	4	3060	1	1	124.439122	122.429798	246.868920	0.

Entrée [28]:

```
Exemple_4_unit_time.set_index("time")
```

Out[28]:

	id_number	customer_class	node	arrival_date	waiting_time	service_start_date	service_t
time							
0	341	0	1	13.864202	15.766566	29.630768	0.082
1	341	1	1	29.713374	31.661844	61.375218	0.023
2	341	1	1	61.398389	62.992398	124.390787	0.035
3	341	0	1	124.425934	122.300761	246.726696	0.142
4	3060	1	1	124.439122	122.429798	246.868920	0.062

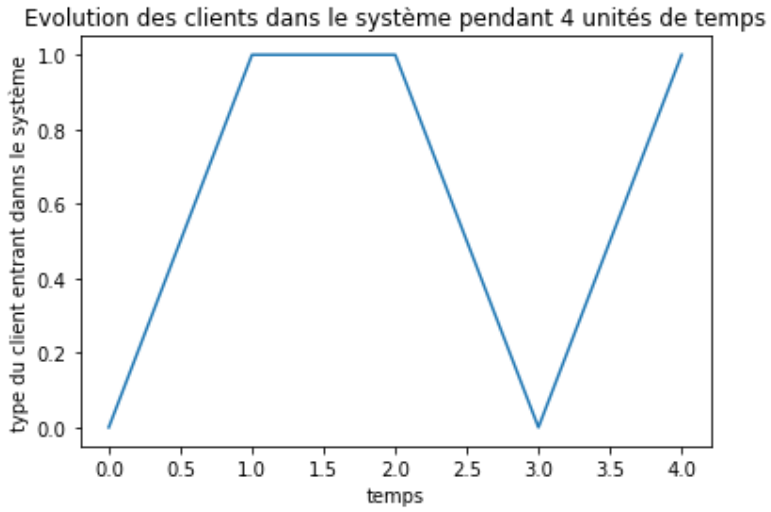
```
# x axis values
x =Exemple_4_unit_time["time"]
# corresponding y axis values
y = Exemple_4_unit_time["customer_class"]

# plotting the points
plt.plot(x, y)

# naming the x axis
plt.xlabel('temps')
# naming the y axis
plt.ylabel('type du client entrant danns le système')

# giving a title to my graph
plt.title('Evolution des clients dans le système pendant 4 unités de temps')

# function to show the plot
plt.show()
```



```
import matplotlib.pyplot as plt



# x axis values
x =Exemple_4_unit_time["time"]
# corresponding y axis values
y = Exemple_4_unit_time["customer_class"]

# plotting points as a scatter plot
plt.scatter(x, y, label= "Client : 0=A & 1=B", color= "green",
            marker= "*", s=200)

# x-axis label
plt.xlabel('temps')
# frequency label
plt.ylabel('Classe du client entrant dans le système')
# plot title
plt.title('Evolution des clients dans le système pendant 4 unités de temps ')
# showing legend
plt.legend()

# function to show the plot
plt.show()
```