

## TP Final Investigación Operativa

Integrantes: Molina Giuliana, Pinto E. Marcos, Witomski Nicole

**Contexto del problema:** Cada vez que se ponen a la venta entradas para ver a la Selección Argentina o un concierto popular, miles de usuarios ingresan simultáneamente a la web. La mayoría esperan en una fila virtual, y muchas veces el sistema se satura, provocando frustración y abandonos (pérdida de ventas). Las empresas que gestionan las plataformas (por ejemplo, Ticketek o EntradaUno) tienen el desafío de definir la cantidad de servidores de forma eficiente, para reducir los tiempos de espera al menor costo posible.

Vamos a simular una fila virtual para la venta de entradas online

## ✓ 1. DEFINICIÓN DE PARÁMETROS

### ✓ PARÁMETROS DEL SISTEMA

```
import numpy as np
import math

#Tasa inicial de arribos (personas/min)
tasa_inicial_llegada = 10

#Factor para escalar la demanda (así nos aseguramos que haya una fila)
factor_demanda = 100

#Es el parámetro que define qué tan rápido cae la llegada en minutos.
t_caída = 30

#Cantidad de servidores en paralelo:
s = 2

#Tiempo de servicio
#tiempo promedio ≈ 4 minutos p/compra
#--> suponemos que una persona tarda entre 3 y 5 minutos en el proceso de compra
#(desde que entra al servicio, elige lugar, y va al checkout)
mu_base = 0.25 #(1/4)

#RESTRICCIÓN: capacidad máxima del sistema
#basándonos en la capacidad del Movistar Arena

capacidad_max= 20000
```

### ✓ PARÁMETROS DE SIMULACIÓN

```
#Duración total en minutos de la venta
T = 480 #(8 horas)

#Paso del tiempo (segundos / minutos) --> cada cuánto se actualiza el sistema
dt = 1/60

#Número de simulaciones
n_sim = int(T/dt)
```

## ✓ 2.FUNCIONES

```
#Nuestra tasa de arribos será una función exponencial decreciente
#A medida que pasa el tiempo, va entrando menos gente a la página.
def tasa_arribos(t):
    return factor_demanda * tasa_inicial_llegada * np.exp(-t / t_caída)
```

```
#Abandono: función sigmoide
personas_adelante = 8000 # a partir de 8000 personas hay 50% chances que abandonen
pendiente = 0.002 # qué tan empujado crece (paciencia)
```

```
#Probabilidad de irse dado la cantidad de personas que tengo adelante en la fila
def tasa_abandono(posicion_en_fila):
    return 1.0 / (1.0 + np.exp(-pendiente * (posicion_en_fila - personas_adelante)))
```

### 3.SIMULACIÓN

Una vez ya definidos todos los parámetros y funciones del problema, armamos la simulación del proceso de la fila virtual.

```
#Aseguramos con la semilla de siempre generar las mismas llegadas y tiempos de servicio
rng = np.random.default_rng(42)

#La simulación recibe una tasa de servicio (mu) y devuelve la cantidad de personas que compraron,
#abandonaron y los que llegaron en total al sistema.

def simulacion(mu):
    servidores = np.zeros(s) # 0 = libre, > 0 = ocupado --> [0. 0.]
    fila = [] # guarda los tiempos de llegada de cada persona

    total_arribos = 0
    total_vendidas = 0
    total_abandono = 0

    for i in range(1, n_sim):
        t = i * dt #actualiza el tiempo actual

        # 1) ARRIBOS (Poisson con lam(t) * dt)
        lam = tasa_arribos(t)

        #cuántas personas llegaron en este segundo
        n_arribos = rng.poisson(lam * dt)
        total_arribos += n_arribos

        #capacidad total actual
        en_servicio = np.sum(servidores > 0)
        en_fila = len(fila)
        en_sistema = en_servicio + en_fila

        #cuántos usuarios nuevos puedo aceptar
        espacio = max(capacidad_max - en_sistema, 0)
        if n_arribos <= espacio:
            fila.extend([t] * n_arribos)

        # 2) ABANDONO en la fila (no afecta a los que ya están en el servicio)
        if fila:
            fans = []
            for i, t_llegada in enumerate(fila):
                # i = personas adelante en la fila
                prob_churn = tasa_abandono(i)
                if rng.random() < prob_churn:
                    total_abandono += 1
                else:
                    fans.append(t_llegada)
            fila = fans

        # 3) ACTUALIZAR SERVIDORES
        prob_serv = 1 - np.exp(-mu * dt)
        for j in range(s):
            if servidores[j] > 0:
                # decidir si termina en este dt con prob_serv
                if rng.random() < prob_serv:
                    servidores[j] = 0
                    total_vendidas += 1

        # 4) ASIGNAR NUEVOS CLIENTES A SERVIDORES LIBRES
        for j in range(s):
            if servidores[j] == 0 and fila:
                # saco al primero de la fila (FIFO)
                fila.pop(0)
                servidores[j] = 1 # cualquier número > 0 para marcar ocupado

    return total_vendidas, total_abandono, total_arribos
```

## ✓ 4.CORREMOS LA SIMULACIÓN

```
def correr_sim(mu, etiqueta, corridas=3):
    ventas = []
    abandonos = []
    llegadas = []

    for _ in range(corridas):
        ven, aba,lle = simulacion(mu)
        ventas.append(ven)
        abandonos.append(aba)
        llegadas.append(lle)

    ventas_m = np.mean(ventas)
    abandonos_m = np.mean(abandonos)
    llegadas_m = np.mean(llegadas)

    print(f"\n=== {etiqueta} ===")
    print(f"Llegadas promedio:          {llegadas_m:,.0f}")
    print(f"Compras exitosas promedio:      {ventas_m:,.0f}")
    print(f"Abandonos promedio:              {abandonos_m:,.0f}")
    print(f"% que compran:                   {100*ventas_m/llegadas_m:5.2f}%")
    print(f"% que abandonan:                 {100*abandonos_m/llegadas_m:5.2f}%")

    return {
        "ventas": ventas_m,
        "abandonos": abandonos_m,
        "llegadas": llegadas_m
    }

# Escenario base (velocidad actual)
res_base = correr_sim(mu_base, "Base (mu)", corridas=2)

# Experimento: impacto de la cantidad de servidores
print("\n=== Impacto de aumentar servidores ===")

#valores_s = [2, 4, 8]
valores_s = [50,100,200]
#valores_s = [100,300,500]

for servidores in valores_s:
    s = servidores # Seteo la nueva cantidad de servidores
    res = correr_sim(mu_base, f"{servidores} servidores", corridas=2)

# === Escenario especial: dimensionar s para vender ~20.000 entradas ===

entradas_totales = capacidad_max # mismas que definiste arriba (20.000)

# Guardamos el valor original de s para restaurarlo después
s_original = s

# Calculamos la cantidad de servidores necesarios para poder vender 20k en T minutos
s_necesarios = math.ceil(entradas_totales / (mu_base * T))
print(f"\nServidores necesarios (teóricos) para vender {entradas_totales:,} entradas en {T} minutos: {s_necesarios}")

# Usamos ese valor de s para este escenario especial
s = s_necesarios

res_20k = correr_sim(mu_base, f"{s_necesarios} servidores (escenario 20k)", corridas=2)

# Restauramos s para no modificar el resto de los experimentos
s = s_original

=== Base (mu) ===
Llegadas promedio:      29,776
Compras exitosas promedio:  10,197
Abandonos promedio:      19,578
% que compran:           34.25%
% que abandonan:         65.75%

=== Impacto de aumentar servidores ===
```

```

=== 50 servidores ===
Llegadas promedio:      29,868
Compras exitosas promedio: 5,670
Abandonos promedio:     24,198
% que compran:          18.98%
% que abandonan:        81.02%

```

```

=== 100 servidores ===
Llegadas promedio:      30,002
Compras exitosas promedio: 7,473
Abandonos promedio:     22,530
% que compran:          24.91%
% que abandonan:        75.09%

```

```

=== 200 servidores ===
Llegadas promedio:      29,986
Compras exitosas promedio: 10,248
Abandonos promedio:     19,738
% que compran:          34.18%
% que abandonan:        65.82%

```

Servidores necesarios (teóricos) para vender 20,000 entradas en 480 minutos: 167

```

=== 167 servidores (escenario 20k) ===
Llegadas promedio:      30,150
Compras exitosas promedio: 9,413
Abandonos promedio:     20,737
% que compran:          31.22%
% que abandonan:        68.78%

```

```

import matplotlib.pyplot as plt

# Cantidad de servidores evaluados
servidores = [4, 8, 50, 100, 200, 500, 1334]

# Tasas de abandono (%) obtenidas en cada escenario
tasas_abandono = [79.11, 78.78, 81.02, 75.09, 65.82, 45.95, 12.88]

# Paleta de colores (uno por barra)
colores = ["tab:red", "tab:orange", "tab:green", "tab:purple",
           "tab:pink", "tab:cyan", "tab:blue"]

plt.figure(figsize=(12,6))
barras = plt.bar([str(s) for s in servidores], tasas_abandono, color=colores)

# Etiquetas y título
plt.xlabel("Cantidad de Servidores")
plt.ylabel("Tasa de Abandono (%)")
plt.title("Impacto de la Cantidad de Servidores en la Tasa de Abandono")
plt.ylim(0, 100)
plt.grid(axis='y', linestyle='--', alpha=0.4)

# Mostrar valor arriba de cada barra
for barra, tasa in zip(barras, tasas_abandono):
    plt.text(barra.get_x() + barra.get_width() / 2,
             tasa + 2,
             f"{tasa:.1f}%",
             ha='center', fontsize=10)

plt.show()

```

