Trabajo Especial: Modelo de Servidores en Paralelo

Nicolás Benjamín Ocampo
Modelos y Simulación - FaMAF
Email: http://benjamin.ocampo@mi.unc.edu.ar

***Abstract*—Este articulo aborda la simulación de un modelo probabilístico de eventos discretos y la obtención de parámetros de interés a partir de ella. En particular, se va a desarrollar un modelo de servidores en paralelo de atención al publico cuyos arrivos y tiempo de servicio serán en base a variables aleatorias con una cierta distribución. Además, se propondrá posibles implementaciones en python para estos modelos para una o múltiples colas de espera. Finalmente, se analizarán y compararán los resultados obtenidos por cada una de las representaciones.**

1. Introducción

1.1. Descripción del Problema

Considera la siguiente situación afrontada por una sucursal bancaria que tiene a disposición 2 de cajeros automáticos para realizar extracciones y deposito de dinero. El encargado del banco planea poner estos cajeros a disposición del público, donde los clientes que llegan a su local deben formarse en una única fila para acceder ellos y realizar sus transacciones. El además planea atender una cantidad máxima de N clientes. Una vez llegada esa cantidad no se permitiran más ingresos a la sucursal. Notar también que cada vez que hay un arrivo de un cliente ocurre la situación ejemplificada en la Figura 1.

Dado este escenario, es muy probable que al encargado le interese la respuesta a las siguientes preguntas:

- ¿Cual es el tiempo esperado que el cliente pasa en el sistema, es decir desde que llega a la cola de espera hasta que se retira del cajero?
- ¿Cual es la proporción de clientes atendidos por ambos cajeros?
- ¿Y si en vez de una única cola se mantiene una por cada cajero, donde los clientes escogen aquella que sea la más corta?

Con el fin de analizar está situación y proponer respuestas a estas preguntas, primero es necesario realizar concretas suposiciones que conciernan al dominio del problema y plantear un modelo probabilístico. Para este caso en particular (y a lo largo del articulo) se trabajará bajo la siguiente situación:

- El tiempo entre la salida de un cliente y la llegada del siguiente en la cola es instantáneo.

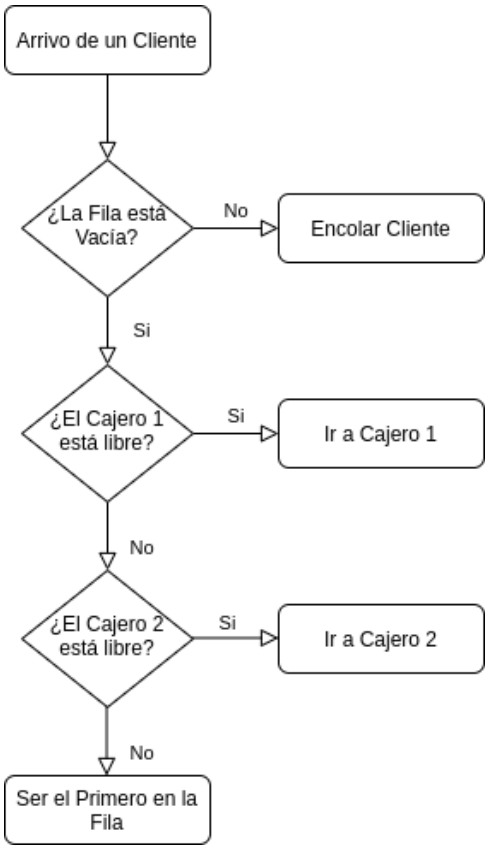

Figure 1. Acción por cada llegada de un cliente a la sucursal

- Los clientes llegan a la sucursal de acuerdo a un *Proceso Poisson Homogéneo* de taza λ.
- Los tiempos de servicio del cajero 1 y 2 son variables aleatorias $TS_1 \sim \varepsilon(\lambda_1)$ y $TS_2 \sim \varepsilon(\lambda_2)$ ambas independientes entre sí y del proceso de arribo.

En particular, haremos un ejemplo de análisis para $\lambda = 6$, $\lambda_1 = 4$, y $\lambda_2 = 3$ pero abarcaremos una implementación general en la sección 2, es decir, para una cantidad de cajeros arbitrarios

1.2. Descripción del Procedimiento

Dado estás suposiciones, podemos proponer una solución por medio de una simulación, es decir, un proced-

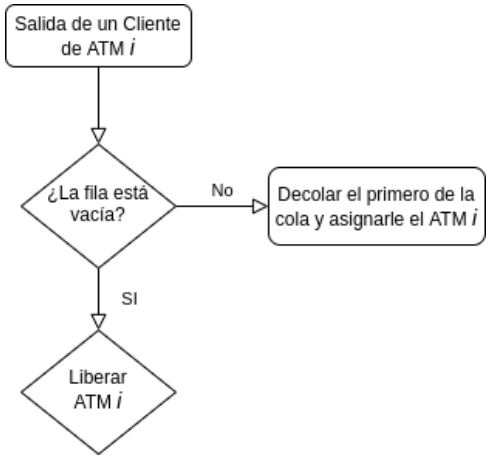

Figure 2. Acción por cada salida de un cliente de algún cajero

imiento que, por medio de variables aleatorias, ejemplifique las posibles ocurrencias del problema.

El procedimiento consistirá principalmente en una cola de prioridades que registre sucesiones de eventos a lo largo de un periodo de tiempo. Cada entrada registrará el tipo de evento y el momento en que ocurrió. Estos eventos son determinados según los datos que queramos obtener. Dado que nos interesa el tiempo de espera de un cliente, tendremos eventos de tipo *ARRIBO* y *SALIDA* (Donde la resta de estos dos por cada cliente determinará su tiempo de espera). Por lo tanto si nos interesan los primeros N usuarios, tendremos en total $2N$ eventos.

Los eventos de tipo *ARRIBO* son generados por medio de un algoritmo que nos obtenga las primeras N llegadas de un *Proceso Poisson Homogéneo*. Por cada arribo realizaremos lo ejemplificado por la Figura 1 y en caso de poder utilizar algunos de los dos cajeros se procederá a encolar su evento de tipo SALIDA, generando algunas de las variables TS_1 o TS_2. El procedimiento terminará al decolar todos los momentos de la cola de prioridades (es decir los $2N$ eventos).

Algo también que nos será de mucha ayuda es mantener en la cola de prioridades registro de que cliente llegó al sistema y por cual cajero salió o fue atendido. Esto con el fin de poder asignarle al primer cliente de la fila (en caso de que no esté vacía) el ATM al cual debe dirigirse. Esto puede verse ejemplificado en la Figura 2.

2. Simulación

2.1. Clases y Variables

Dado que nos interesan los tiempos de arribo-salida y el número de cajero por el cual paso cada cliente, esta información debe ser almacenada en variables para luego consultarla al final de la simulación. Por lo tanto, por cada ocurrencia de un evento debemos hacer registro de estos datos de interés.

En nuestro caso, con el fin de hacer más intuitivo la implementación del algoritmo, lo desarrollaremos en el paradigma orientado a objetos (Una implementación usando programación imperativa puede hallarse en [1]). Por ende, en lugar de variables mantendremos la información almacenada en objetos o instancias de distintas clases.

Además, generalizaremos el problema para implementarlo bajo las siguientes prestaciones:

- La sucursal posee M cajeros funcionando en paralelo.
- El tiempo de servicio del cajero i corresponde a una variable aleatoria $TS_i \sim \varepsilon(\lambda_i)$ para $i \in \{1, ..., M\}$.

Luego los análisis y conclusiones finales los haremos bajo las condiciones descriptas en la sección 1.1.

Una vez dicho esto, las clases (y sus respectivos constructores) que van a representar a un cliente y un cajero serán definidas de la siguiente manera:

```
class ATM:
    def __init__(self, service_rate):
        self.service_rate = service_rate
        self.is_busy = False
        self.attended_customers = 0
```

```
class Customer:
    def __init__(self, arrival_time):
        self.arrival_time = arrival_time
        self.departure_time = None
```

Notar que cada cliente va a almacenar su tiempo de arribo y salida. Similar para los cajeros ATM en cuanto a la cantidad de clientes que atendieron. Además, cada ATM indicará si está siendo ocupado y su parámetro λ_i para poder generar el tiempo de servicio.

Luego mantendremos otra clase que representará la simulación y será definida como.

```
class ATMSimulator:
    Event = Enum('Event', 'ARRIVAL DEPARTURE')
    def __init__(self, λ, λs, nof_arrivals):
        self.λ =λ
        self.λs =λs
        self.nof_arrivals = nof_arrivals
        self.simtime = 0
        self.queue = deque()
        self.atms = []
        self.customers = []
        self.events = []
```

- λ: Parámetro del proceso Poisson.
- λs: Lista de parámetros de tiempo de servicio de todos los cajeros.
- *nof_arrivals*: Cantidad máxima de clientes que se atenderá.
- *simtime*: Cantidad de tiempo simulado que ha ocurrido.
- *atms*: Lista de referencias a los ATM participes de la sucursal.
- *customers*: Lista de referencias de los clientes simulados.
- *queue*: Cola de clientes al tiempo *simtime*.

- *events*: Cola de prioridades de 3-uplas cuyas coordenadas 1, 2, y 3 corresponden al tiempo en que ocurrió un evento, su tipo, y una referencia a un cliente o a un ATM (dependiendo si es un evento de tipo *ARRIBO* o *SALIDA*) respectivamente.

Cabe recalcar que las variables de tiempo y estado del sistema (SS - System State Variables) están contempladas por los atributos **simtime** y **events**. Además, no nos hará falta hacer uso de variables de conteo salvo para los clientes atendidos por cada cajero.

2.2. Descripción del Algoritmo

Como se mencionó en la sección 1.2, vamos a utilizar una cola de prioridad como estructura de datos para almacenar a los eventos. Esto se debe a que estos deben estar ordenados en el tiempo y puede ser complicado implementar el algoritmo para que se satisfaga esta precondición.

Una posibilidad que se comenta en [1] es ir generando llegadas de clientes a medida que la simulación va transcurriendo manteniendo como SS los tiempos del arribo actual y salida de los clientes siendo atendidos por cada ATM. Luego, aquel tiempo que sea menor en las variables de estado, determinará que evento ocurrió. Si bien esto funciona para el caso de 2 cajeros, implementarlo para más cantidades puede tornarse no muy intuitivo.

Otra cosa a considerar es que los tiempos de arrivos y las instancias de los clientes se harán al inicio de la simulación. De todas formas también es posible ir generándolos durante el procedimiento.

Por lo tanto la inicialización de variables serán de la siguiente manera:

```
def set_up(self):
    self.simtime = 0
    self.atms = [ATM(λ_i) for λ_i in self.λs)]
    self.arrival_times = poisson_process(
        self.λ,
        self.nof_arrivals
    )
    self.customers = [
        Customer(time)
        for time in arrival_times
    ]
    self.events = [
        (time, self.Event.ARRIVAL, customer)
        for time, customer in zip(
            arrival_times,
            self.customers
        )
    ]
    # Convierte la cola de eventos
    # en una cola de prioridades
    heapify(self.events)
```

Una vez esto podemos describir el algoritmo como:

```
def run(self):
    self.set_up()
    while len(self.events) > 0:
        time, event, obj = heappop(self.events)
        self.simtime = time
        if event is self.Event.ARRIVAL:
            self.handle_incoming_customer(obj)
        elif event is self.Event.DEPARTURE:
            self.handle_outgoing_customer(obj)
```

A partir de aquí la implementación del algoritmo se reduce a desarrollar los métodos **handle_incoming_customer** y **handle_outgoing_customer**. Estos son nada mas y nada menos que lo mencionado en las Figuras 1 y 2 con la diferencia que en lugar de 2 cajeros tenemos una cantidad arbitraria de ellos.

Notar que el parámetro *obj* que reciben estos métodos puede corresponder a un cliente o a un atm que serán necesarios para realizar estos procedimientos.

2.2.1. Manejo de Arribo de un Cliente. Por un lado, cada vez que una persona llegue a la sucursal, esta se va a formar en la fila tanto si hay gente en ella, o si está vacía pero con todos los cajeros ocupados. Por lo tanto, por cada arribo debemos hacer estos dos chequeos.

Uno pensaría que analizar la premisa del uso de los ATMs es costosa, pero en practica estos no tienden a ser demasiados. De todas formas si se quisiera evitar esto, se puede mantener una variable de la simulación que registre la cantidad de clientes que hay actualmente en la sucursal. Luego la condición de encolado puede chequearse únicamente viendo si la cantidad de clientes al tiempo *simtime* es menor que la cantidad de cajeros. Una solución de está forma puede verse en [1]. Sin embargo, se optó por mantenerlo de la forma más parecida posible a lo que uno haría en la vida real.

En caso de haber algún cajero disponible, se procede a asignarle a este cliente aquel cajero que tenga menor numeración (Se podría también asignar de manera aleatoria entre los cajeros disponibles). La implementación de este método sería de la siguiente manera.

```
class ATM:
# ... Metodo de la clase ATM
def attend_customer(self, simtime, customer):
        customer.departure_time = (
            simtime +
            ϵ(self.service_rate)
        )
        self.is_busy = True
        self.attended_customers += 1

def handle_incoming_customer(self, customer):
    free_atms = [atm for atm in self.atms
        if not atm.is_busy()
    ]
    if len(self.queue) > 0 or len(free_atms) == 0:
        self.queue.append(customer)
    else:
        atm = free_atms[0]
        atm.attend_customer(
            self.simtime,
            customer
        )
        event = (
            customer.departure_time,
            self.Event.DEPARTURE,
            atm
        )
        heappush(self.events, event)
```

Notar que la función **attend_customer** es un método de la clase ATM que asigna el cajero como ocupado, incrementa la cantidad de clientes que pasaron por ella, y asigna el tiempo de salida que tendrá el cliente dado por parámetro a partir del tiempo de simulación *simtime* de acuerdo a su *service_rate*. Es por eso que luego podemos acceder al atributo de dicho cliente y encolar su evento de tiempo de salida.

2.2.2. Manejo de Salida de un Cliente. Por otro lado, cuando una persona termina de usar un cajero, nuevamente nos fijamos en el tamaño de la fila. Si está vacía, entonces el cajero queda en desuso, caso contrario se le es asignado al primero en la cola.

Algo a notar de esta solución es que si hay gente formada, entonces quiere decir que los otros cajeros están ocupados. Pues si hubiera alguno disponible, este no estaría encolado debido a la condición en el método **handle_incoming_customer**.

Una posible implementación de este método es la siguiente:

```
def handle_outgoing_customer(self, atm):
    if len(self.queue) > 0:
        customer = self.queue.popleft()
        atm.attend_customer(
            self.simtime,
            customer
        )
        event = (
            customer.departure_time,
            self.Event.DEPARTURE,
            atm
        )
        heappush(self.events, event)
    else:
        atm.is_busy = False
```

Una descripción gráfica completa del algoritmo puede verse en la Figura 3.

2.3. Obtención de Resultados

Como se dijo al principio, la información post simulación estará almacenada en los objetos que representaron a los clientes y los cajeros. Es decir, en las listas *customers* y *atms*.

2.4. Simulación con Múltiples Colas

Si se quisiera usar una cola por cajero, donde por cada arribo de un cliente, este se dirige a la que tenga longitud más corta solo es necesario hacer una ligera modificación en los métodos **handle_incoming_customer** y **handle_outgoing_customer**.

Cada vez que hay una llegada, ni bien se entra a la sucursal debemos encontrar la cola del cajero en la que haya menos gente. Esto puede obtenerse facil a través del siguiente método.

```
def shortest_queue(self):
    min_queue_id = argmin([
```

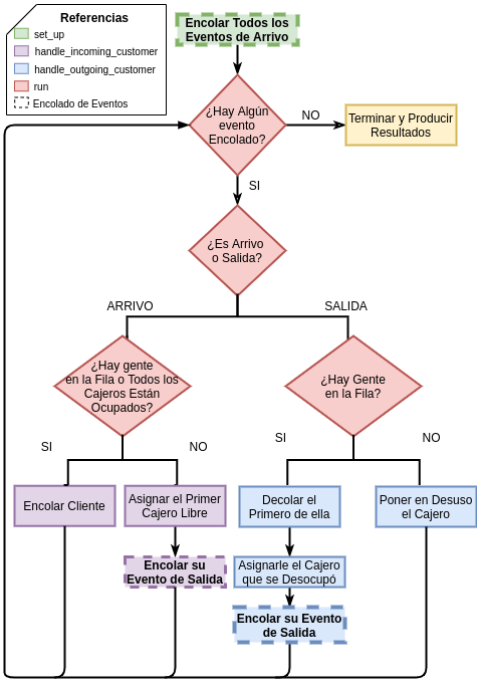

Figure 3. Pasos del Algoritmo

```
        len(queue)
        for queue in self.queues
    ])
    shortest_queue = self.queues[min_queue_id]
    atm = self.atms[min_queue_id]
    return shortest_queue, atm
```

Donde la función *argmin* es un algoritmo de la librería *numpy* que devuelve el índice del mínimo elemento de un arreglo. Si hay varias que tienen la misma longitud, devuelvé aquella cuya numeración sea la más chica, es decir, la primera ocurrencia.

También nos hará falta conocer el ATM asociado a esa fila para saber si el cajero está siendo utilizado o no (En caso de que no se prosigué a asignarle a dicho cliente el cajero). Luego el procedimiento prosigue de igual manera al caso al algoritmo descripto en la Figura 3, donde en vez de verificar si todos los cajeros están ocupados lo consultamos sobre aquel cajero que el cliente eligió. Por lo tanto solo es cuestión de cambiar lo siguiente.

```
    queue, atm = self.shortest_queue()
    if len(queue) > 0 or atm.is_busy:
        # Encolar cliente
    else:
        # Atender y encolar evento de salida
```

Cada vez que hay una salida, sabemos cual es el cajero por el cual el cliente fue atendido y por ende su fila asociada (esto en implementación se puede resolver agregando como atributo un id o la referencia a la instancia de tal fila). Una vez que obtenemos su fila asociada, realizamos exactamente lo mismo que para el caso de una única cola.

3. Resultados

Como se describió en la sección 1.1 se analizó este problema para el caso en que la taza de arrivos del proceso Poisson es $\lambda = 6$, y la cantidad de cajeros a disposición es 2, cuyos tiempos de servicio son v.a. $TS_1 \sim \varepsilon(\lambda_1 = 4)$ y $TS_2 \sim \varepsilon(\lambda_2 = 3)$.

3.1. Simulación con Única Cola

Realizando la ejecución de la simulación para las primeras 10000 llegadas se obtuvieron las siguientes estimaciones.

nsim	Media	Desvio Estandar	Proporción ATM 1
10000	0.889min	0.755min	0.589

Esto nos muestra que el tiempo promedio que un cliente pasa en el sistema es aproximadamente 53.34 segundos. Algo también a notar es que la proporción de clientes atendidos por el cajero 1 es mayor que la del cajero 2. Esto es de esperarse pues:

$$\begin{aligned} P(TS_1 > TS_2) &= \int_0^\infty \int_0^x \lambda_1 e^{-\lambda_1 x} \lambda_2 e^{-\lambda_2 y} dy dx \\ &= \int_0^\infty \lambda_1 e^{-\lambda_1 x} (\int_0^x \lambda_2 e^{-\lambda_2 y} dy) dx \\ &= \int_0^\infty \lambda_1 e^{-\lambda_1 x} F_{TS_2}(x) dx \\ &= \int_0^\infty \lambda_1 e^{-\lambda_1 x} (1 - e^{-\lambda_2 x}) dx \\ &= \int_0^\infty \lambda_1 e^{-\lambda_1 x} - \int_0^\infty \lambda_1 e^{-(\lambda_1+\lambda_2)x} dx \\ &= 1 - \int_0^\infty 4e^{-7x} dx \\ &= 1 - \frac{4}{7} \\ &\approx 0.42857 \end{aligned} \tag{1}$$

Es decir, es más probable que el tiempo de servicio del cajero 2 supere al del cajero 1 y por ende demore más en atender a los clientes. Notar además que este resultado es muy similar a la proporción de clientes atendidos por el cajero 2 (y por lo tanto $P(TS_1 \leq TS_2) = \frac{4}{7}$ se asemejaría a la proporción de clientes atendidos por el cajero 1).

Podemos rápidamente realizar un test de hipótesis y comprobar si la v.a. X: "El cliente es atendido por el cajero 1" es una Bernoulli de parámetro $p = \frac{4}{7}$.

Dado que en la muestra de tamaño 10000 se registraron las frecuencias observadas.

X = 0	X = 1
4110	5890

Usando un test de χ^2 de pearson, el valor del estadístico nos quedaría:

$$t_0 = \frac{(4110 - 10000\frac{3}{7})^2}{10000\frac{3}{7}} + \frac{(5890 - 10000\frac{4}{7})^2}{10000\frac{4}{7}} \approx 12.6075$$

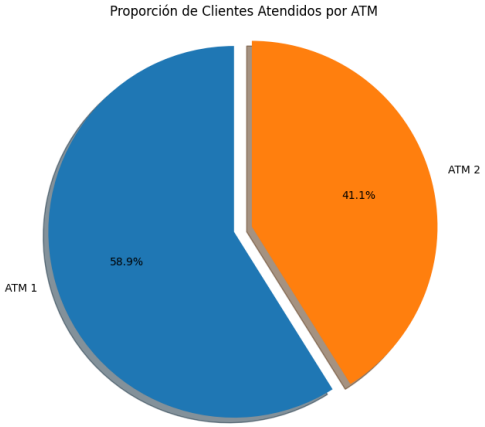

Figure 4. Única Cola: Proporción de Clientes atendidos por cajero.

Luego testeando con una distribución chi cuadrado tenemos que:

$$p - valor \approx P(\chi_1^2 \geq 12.6075) \approx 0.000384$$

Para un nivel de confianza del 95% la hipótesis nula se rechaza. Sin embargo, se analizó ejecutando 1000 veces la simulación verificando si el resultado era consistente y se obtuvo que un 44.2% de las veces se rechazo a la hipótesis nula.

Por otro lado, si prestamos atención al histograma de los datos en la Figura 5 vemos que claramente los tiempos de espera de un cliente más frecuentes se centran en el intervalo [0, 1]. Es decir, la mayoría de clientes realizaba sus transacciones en menos de 1 minuto. Además, podemos notar que la forma se asimila a la de una variable aleatoria exponencial. En particular se nota mayor similitud con una exponencial de parámetro $\lambda = 1$ como se muestra en la Figura 6.

Por lo tanto, se utilizó un test de *Kolmogorov Smirnov* para validar está aseveración. Sin embargo, el método rechazó la hipótesis nula el 100% de las veces que se lo ejecutó obteniendo como $p - valor$ estimado el valor de 0. Desafortunadamente, se desconoce la razón de porque pudo verse está gran similitud en el histograma y haber obtenido estos resultados en el test de hipótesis. Se intentó hacer una afirmación más débil afirmando que los datos provenían de una v.a. exponencial y estimando el parámetro λ por medio de su estimador de máxima verosimilitud. También se analizó si la muestra provenían de una v.a. con distribución gamma y estimando sus parámetros debido a la forma de su función de distribución. Sin embargo, ninguno de estos intentos obtuvieron resultados distintos al test anterior.

3.2. Simulación con Múltiples Colas

Por último obtuvimos los resultados del mismo problema pero está vez usando una cola por cajero. Para este caso se

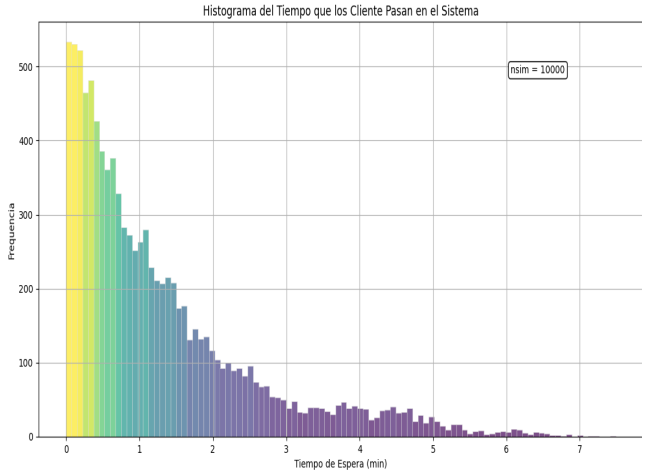

Figure 5. Única Cola: Histograma de los primeros 10000 arrivos.

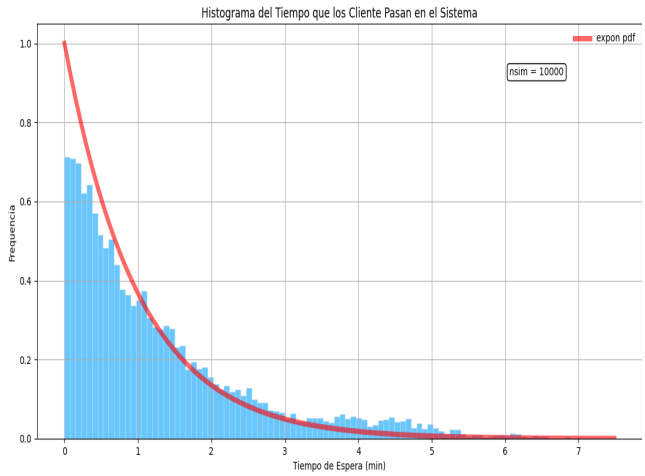

Figure 6. Única Cola: Histograma normalizado y comparación con v.a. exponencial con $\lambda = 1$

obtuvo lo siguiente:

nsim	Media	Desvio Estandar	Proporción ATM 1
10000	1.04min	0.919min	0.614

En el caso del tiempo medio, pareciera que a priori se comportase ligeramente mejor el caso anterior. Lo que si pareciese influenciar es en la frecuencia que el cajero 1 es utilizado. Dado que tenemos dos colas distintas, un cliente utiliza el cajero 1 si y solo si su fila es la más corta al momento de formarse. Como este cajero es más probable de desocuparse antes, entonces tiende a tener una fila de longitud más corta. Además, se probó realizar la prueba de hipótesis que realizamos en la sección 3.1, con la

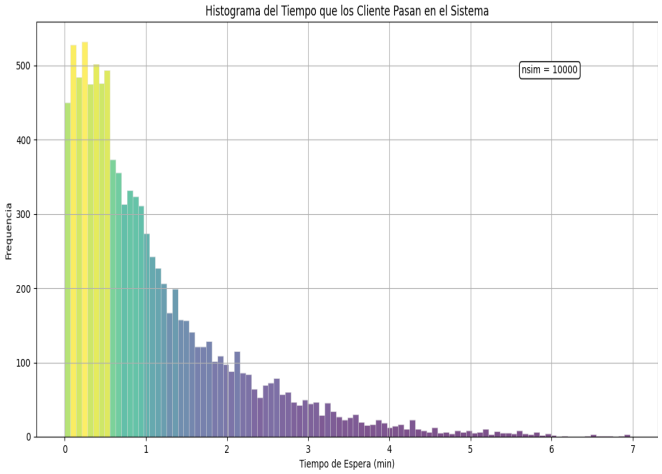

Figure 7. Múltiples Colas: Histograma de los primeros 10000 arrivos.

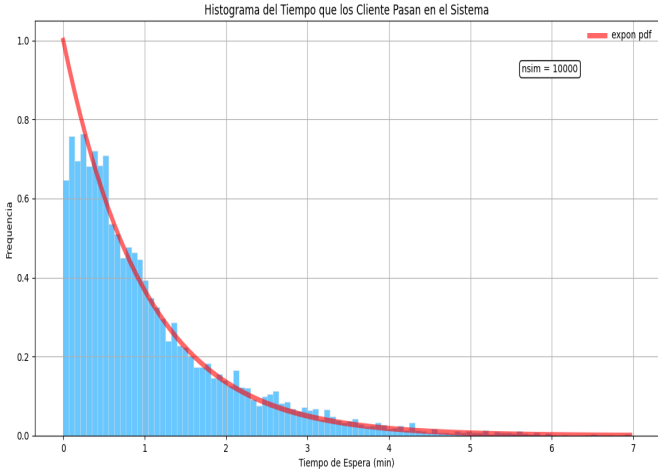

Figure 8. Múltiples Colas: Histograma normalizado y comparación con v.a. exponencial con $\lambda = 1$

salvedad de que en este caso siempre se obtuvo un $p-valor$ demasiado chico y un rechazo a la hipótesis nula a un nivel de confianza del 95%.

En el caso de la distribución de los datos de tiempo de espera, su histograma también se asemeja al de una exponencial de parámetro $\lambda = 1$. No obstante, se procedió a hacer las mismas pruebas que se realizaron para el caso de una única cola pero sin obtener resultados diferentes.

3.3. 1 vs 2 Colas

Por último se optó por realizar un ztest sobre el tiempo de espera de los clientes y verificar si alguno de los dos modelos es significativamente mejor en términos de minimización de estos tiempos. Suponiendo que se tiene que el

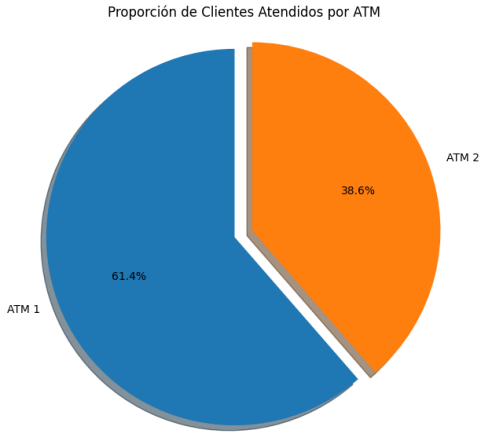

Figure 9. Múltiples Colas: Proporción de Clientes atendidos por cajero

actual modelo de la sucursal es un sistema con múltiples colas. El dueño va a considerar cambiarse a uno de única cola si su verdadera media del tiempo de espera de los clientes es menor a $0.9min$. Si consideramos que:

- $H_0 : \mu = 0.9$ donde μ es el verdadero tiempo promedio de espera de los clientes
- $H_a : \mu < 0.9$

Luego bajo hipótesis nula, tenemos que el estadístico de prueba es:

$$Z = \frac{\bar{X} - \mu_0}{s\sqrt{n}}$$

Donde X es la v.a. de tiempo de de espera de los clientes, μ_0 es la media bajo hipótesis nula, y n es el tamaño de la muestra. Notar también que como n es suficientemente grande, usando el TCL, no es difícil ver que el estadístico tiene distribución aproximadamente normal. Luego el valor observado obtenido al usar una única cola es el que mostramos en la sección 3.1

$$z = \frac{0.889 - 0.9}{0.755\sqrt{10000}} \approx -1.457$$

Si hacemos el test con un nivel de confianza del 95% H_0 es rechazada si $z < -1.64$. Dado que esto no se cumple, podemos decir que no hay evidencia suficiente que justifique el cambio a una única cola.

3.4. Otras Métricas

Se tomaron en cuenta otros posibles resultados tales como los percentiles, y otras posibles configuraciones en los parámetros. Además se probó agregar más cajeros con el fin de determinar si alguna estrategia de encolado resultaba mejor cuando está cantidad crecía. Algo a notar (dado en como se implementó el algoritmo) es que se tiene preferencia por aquellos cajeros con menor numeración, entonces ocurre que estos tienden a tener una proporción más alta de clientes aún cuando todos tienden el mismo o mayor tiempo de servicio. Si bien para el caso con dos cajeros esto no es de notarse demasiado, puede ser un problema si aquellos cajeros con alta numeración son los más rápidos. De todas formas, como mencionamos antes no es muy complicado alterar el algoritmo para agregar una elección aleatoria.

Para la configuración que estuvimos trabajando a lo largo del articulo, se obtuvieron los siguientes percentiles:

Modelo	20	50	80
Única Cola	0.2566	0.7393	1.5053
Múltiples Colas	0.2695	0.7572	1.6914

Con esto puede verse que el 80% de los clientes salieron del sistema antes de los 1.7min. Algo que puede ser un poco contra intuitivo, es que si bien hubo mayor proporción de clientes atendidos por el cajero 1 en el modelo de múltiples colas, el tiempo de espera de los clientes fue ligeramente mayor (A pesar de que la mayoría de ellos utilizaron el ATM "más rápido").

4. Conclusión

Recopilando los resultados podemos concluir lo siguiente:

- Los tiempos de espera promedio y proporción de clientes atendidos por el cajero 1 para las primeras 10000 llegadas, son aproximadamente 0.889 minutos y 0.589 respectivamente para el modelo de una cola.
- De 1000 test de validación sobre la proporción de clientes atendidos por el cajero 1, el 44.2% rechazaron con una confianza del 95% que los datos de la muestra provenían de una v.a. Bernoulli de parámetro $p = \frac{4}{7}$. Donde $p = P(TS_1 \leq TS_2)$, es decir, la probabilidad de que el cajero 1 atienda más rápido a un cliente que el cajero 2.
- Para el caso de múltiples colas la mayoría de los tests rechazaron la hipótesis anterior. Capaz puede deberse a que, como se vio en la introducción del problema, los clientes tienen preferencia por aquellos cajeros con menor numeración. En el caso del modelo 1, esto solo se veía cuando ambos cajeros estaban disponibles para un cliente, es decir, cuando la fila estaba vacía y ambos cajeros desocupados. En cambio en el modelo 2, esta preferencia ocurre con más frecuencia. Específicamente cuando hay más de una cola con la menor longitud.
- Al 95% de confianza, en ambos modelos el test de Kolmogorov Smirnov rechazó que la muestra del tiempo de espera obtenidos proviniesen de una v.a. exponencial o gamma a pesar de su similitud en la forma del histograma.
- En ambos modelos, 50% de los clientes realiza sus transacciones antes de los 50 segundos y el 80% antes de los 100 segundos.

- De acuerdo a un ztest efectuado al 95% de confianza, no hay evidencia suficiente para considerar un modelo por encima de otro, dado que ambos producen resultados similares.

Además, si bien para la configuración trabajada, las condiciones de preferencias por el cajero 1 en caso de igual tamaño de filas o cuando el cliente tiene la opción de ir a ambos cajeros, no son demasiado influyentes. Cuando el numero de cajeros crece, esto puede repercutir en el tiempo de espera de los clientes. Por ejemplo si se tuvieran los cajeros más rápidos con numeración más alta. Esta preferencia puede influenciar aún si todos los cajeros tienen la misma distribución de tiempo de servicio. Por lo tanto, como se mencionó durante el articulo, es recomendable usar alguna selección aleatoria para estos casos en los que el cliente tiene varias posibles elecciones.

References

[1] Ross, S. (2013), *Simulation*, (5th ed). Amsterdam, The Netherlands. Elsevier.

[2] Devore, J. L. (2010), *Probability and Statistics for Engineering and the Sciences*, (8th ed.). California Polytechnic State University, San Luis Obispo. Brooks Cole/Cengage Learning.

[3] Segata, M., Lo Cigno, *Practical Discrete Event Simulation and The Python Simulator*. Universidad de Trento, Recuperado el 17 de Julio de 2020, de http://disi.unitn.it/locigno/teaching-duties/spe/11_DES_PythonSimulator.pdf