Informe Simulación Basada en agentes para Colas

Aaron Sarmiento
Robert Burgos
Nelson Rincon
Sergio Heredia
Laura García
Miguel Thómas

Materia:
Sistemas complejos

Universidad Sergio Arboleda

Escuela de Ciencias Exactas e Ingeniería
Ciencias de la Computación e Inteligencia Artificial
2024

Introducción

Los sistemas de colas son fundamentales para comprender y optimizar procesos en diversas industrias y sectores, incluidas las telecomunicaciones, el servicio al cliente y la fabricación. El análisis y la simulación de estos sistemas proporcionan información sobre la eficiencia operativa, la calidad del servicio y la planificación de la capacidad del sistema. Este documento describe la simulación de dos modelos de cola fundamentales: M/M/1/∞/∞ y M/M/1/K/∞. Estos modelos se caracterizan por un único servidor, tiempos de servicio exponenciales y una capacidad del sistema infinita o finita, respectivamente.

Simulación

Para abordar el problema resulta imperativo primero establecer las clases y sus roles, para el desarrollo de un problema de colas M/M/n donde n es el número de servidores necesitamos 3 clases fundamentales: Cliente, Servidor y Simulación, esta última dedicada a manejar la lógica única de cada una de las colas que se van a analizar.

Las clases Cliente y Servidor son comunes para ambos casos y se ´plantean de la siguiente manera:

- **Cliente:** La clase Cliente representa un cliente individual en el sistema de colas. Cada cliente tiene tres atributos principales:
    - **Arrival_time:** La hora a la que el cliente llega al sistema.
    - **start_service_time:** la hora a la que el cliente comienza a recibir el servicio. Inicialmente ninguno, se establece cuando el cliente comienza a ser atendido.
    - **end_service_time**: la hora a la que se completa el servicio del cliente. Al igual que start_service_time, inicialmente es Ninguno y se establece al finalizar el servicio.

      Además, la clase Cliente proporciona tres **métodos**:

    - **__init__(self, Arrival_time):** un constructor que inicializa un nuevo cliente con una hora de llegada específica.
    - **start_service(self, start_time):** Establece el start_service_time para el cliente, indicando cuándo comenzó a ser atendido.
    - **end_service(self, end_time):** establece el end_service_time para el cliente, marcando cuándo se completó su servicio.
    - **time_in_system(self):** Devuelve el tiempo total que el cliente pasó en el sistema, calculado como la diferencia entre end_service_time y Arrival_time. Si el servicio no ha finalizado (end_service_time es Ninguno), devuelve Ninguno.

```python
1 usage
class Customer:
    def __init__(self, arrival_time):
        self.arrival_time = arrival_time
        self.start_service_time = None
        self.end_service_time = None

    2 usages (2 dynamic)
    def start_service(self, start_time):
        self.start_service_time = start_time

    5 usages (4 dynamic)
    def end_service(self, end_time):
        self.end_service_time = end_time

    def time_in_system(self):
        if self.end_service_time is not None:
            return self.end_service_time - self.arrival_time
        return None
```

- **Servidor:** La clase Servidor modela un único servidor en el sistema de colas, que puede procesar o atender a los clientes. El servidor tiene dos atributos principales:
    - current_customer: rastrea al cliente actualmente atendido por el servidor.
    - busy: determina si esta atendiendo actualmente a un cliente

  Con respecto a los métodos resultan similares a la clase cliente sin embargo aqui se ahonda mas en la lógica que gestiona los tiempos haciendo uso de los métodos de Customer

```python
1 usage
class Server:
    def __init__(self):
        self.current_customer = None
        self.busy = False

    4 usages (2 dynamic)
    def start_service(self, customer, current_time):
        self.current_customer = customer
        self.busy = True
        self.current_customer.start_service(current_time)

    5 usages (4 dynamic)
    def end_service(self, current_time):
        if self.current_customer:
            self.current_customer.end_service(current_time)
            self.busy = False
            finished_customer = self.current_customer
            self.current_customer = None
            return finished_customer
        return None
```

M/M/1/∞/∞

El sistema de colas M/M/1/∞/∞ representa un escenario con un solo servidor, proceso de llegada de Poisson, tiempos de servicio exponenciales, capacidad de cola infinita y una fuente de población infinita. La simulación de este modelo está diseñada para analizar el número promedio de usuarios en el sistema (NS), el tiempo promedio de permanencia de los usuarios en el sistema (TS), así como el número promedio de usuarios en la cola (NQ) y el Tiempo promedio que los usuarios pasan esperando en la cola (TQ).

La simulación basada en Python utiliza programación orientada a objetos para modelar a los clientes y al servidor. Los clientes se generan con tiempos entre llegadas distribuidos exponencialmente, y los tiempos de servicio también se distribuyen exponencialmente. El servidor procesa a los clientes por orden de llegada. Las métricas de rendimiento se calculan rastreando el tiempo total que los clientes pasan en el sistema y en la cola, así como integrando la longitud de la cola durante la duración de la simulación.

La clase Simulación modela el comportamiento y el rendimiento de un sistema de colas durante un período específico, aprovechando las clases Cliente y Servidor previamente definidas. Esta clase está diseñada para simular escenarios típicos de sistemas de servicios, como el servicio de cajero de un banco, donde los clientes llegan aleatoriamente, hacen cola (si es necesario) y son atendidos por un servidor. Incluye parámetros y métodos para generar llegadas, atender a los clientes y calcular métricas clave de rendimiento.

Atributos

- **lambda_rate**: La tarifa a la que llegan los clientes al sistema (tasa de llegada).
- **mu_rate:** La velocidad a la que el servidor procesa o atiende a los clientes (tarifa de servicio).
- **time_end**: La duración durante la cual se ejecutará la simulación.
- servidor: una instancia de la clase Servidor para representar el único servidor en el sistema.
- **Clientes**: una lista para realizar un seguimiento de los clientes que actualmente esperan en la cola.
- **time:** Un contador para el tiempo actual dentro de la simulación.
- **total_time_in_system:** Acumula el tiempo total de permanencia de todos los clientes en el sistema, para luego calcular el tiempo promedio de permanencia en el sistema (TS).
- **total_customers_served**: Cuenta el número de clientes que han sido atendidos durante la simulación.
- **total_time_in_queue:** Acumula el tiempo total de espera de todos los clientes en la cola.
- **queue_length_time_product**: Integra el producto de la longitud y el tiempo de la cola, para calcular el número promedio de clientes en la cola (NQ).

Métodos

- **__init__**: Inicializa la simulación con las velocidades especificadas (lambda_rate, mu_rate), duración (time_end) y prepara el estado inicial.
- **generate_next_arrival:** utiliza la distribución exponencial para generar el tiempo hasta que llegue el siguiente cliente, en función de la tasa de llegada (lambda_rate).
- **generate_service_time:** Genera el tiempo de servicio para un cliente utilizando la distribución exponencial, en función de la tasa de atención (mu_rate).
- **run:** ejecuta la simulación, rastrea las llegadas de clientes y la finalización del servicio, actualiza el estado del sistema y calcula las métricas clave de rendimiento. Se repite hasta que la simulación alcanza el tiempo de finalización especificado, manejando las llegadas de clientes y los eventos de servicio, actualizando los tiempos de espera y ajustando la cola según sea necesario.

La simulación calcula dinámicamente NS (número promedio de usuarios en el sistema), TS (tiempo promedio que los usuarios pasan en el sistema), NQ (número promedio de usuarios en la cola) y TQ (tiempo promedio que los usuarios pasan en la cola) en función de los datos acumulados durante el período de simulación.

```python
class SimulationMM1K:
    def __init__(self, lambda_rate, mu_rate, time_end, K):
        self.lambda_rate = lambda_rate
        self.mu_rate = mu_rate
        self.time_end = time_end
        self.K = K
        self.server = Server()
        self.customers = []
        self.time = 0
        self.total_time_in_system = 0
        self.total_time_in_queue = 0  # Total time customers spend waiting in the queue
        self.total_customers_served = 0
        self.lost_customers = 0
        self.queue_length_time_product = 0  # To integrate queue length over time

    2 usages
    def generate_next_arrival(self):
        return np.random.exponential(1.0 / self.lambda_rate)

    2 usages
    def generate_service_time(self):
        return np.random.exponential(1.0 / self.mu_rate)

    1 usage
    def run(self):
        next_arrival_time = self.time + self.generate_next_arrival()
        last_event_time = self.time  # Track time of the last event for queue length integration
```

M/M/1/K/∞

El modelo M/M/1/K/∞ extiende el modelo M/M/1/∞/∞ introduciendo una capacidad finita (K) para el sistema, limitando el número de clientes que pueden estar en el sistema (esperando o siendo atendidos).) en cualquier momento dado. Este modelo es crucial para analizar sistemas donde los recursos son limitados o donde es importante limitar la congestión para mantener la calidad del servicio.

Similar a la simulación M/M/1/∞/∞, este modelo se implementa en Python y calcula las mismas métricas de rendimiento. Sin embargo, también realiza un seguimiento de los clientes perdidos: aquellos que llegan cuando el sistema está al límite de su capacidad y, por lo tanto, son rechazados sin servicio. Esta métrica es fundamental para comprender el impacto de la capacidad del sistema en la accesibilidad del servicio.

Aunque en el modelo teórico, como ya veremos más adelante, la restricción de capacidad finita aumenta grandemente la complejidad del sistema para correr la simulación basta con implementar condicionales en los puntos adecuados que impidan la inserción de nuevos clientes una vez excedido el umbral K.

Resultados

Para presentar resultados y establecer un punto de comparación objetivo primero debemos desarrollar numéricamente los valores esperados para cada una de las colas, sabemos que:

$$N_s = \sum_{n}^{k} nP_n$$

$$T_s = N_s/\boldsymbol{\lambda}$$

$$T_q = T_s - 1/\boldsymbol{\mu}$$

$$N_q = \boldsymbol{\lambda} * T_q$$

Donde **λ** y $\boldsymbol{\mu}$ son la tasa de llegada y tasa de atención respectivamente, n es el número de clientes y Pn es la probabilidad para un estado de n clientes (Esta probabilidad esta dada por una distribución de Poisson).

Ns, Ts, Tq y Nq son las métricas de rendimiento y representan el número promedio de clientes en el sistema, el tiempo que pasan los clientes en promedio en el sistema, el tiempo promedio que pasan los clientes el cola y el número promedio de clientes en cola respectivamente.

```
PS C:\Users\LENOVO\Documents\Python Scripts> & C:/Users/LENOVO/AppDa
hon3.11.exe c:/Users/LENOVO/Downloads/Queue2.py
Average number of users in the system (NS): 1.4294102168188108
Average time spent by users in the system (TS): 0.7494102993733869
Average number of users in the queue (NQ): 0.7938136401948964
Average time spent by users in the queue (TQ): 0.41610772804561985
Number of lost customers: 4892
```

Para resolver los valores numéricos debemos resolver primero la sumatoria de Ns y para ello hay que resolver primero Pn, que sabemos que es una distribución de Poisson por lo tanto Pn queda

$$P_n = \frac{1-\frac{\boldsymbol{\lambda}}{\boldsymbol{\mu}}}{1-(\frac{\boldsymbol{\lambda}}{\boldsymbol{\mu}})^{k+1}}$$

de modo que para el sistema **M/M/1**/∞/∞ solo nos quedaría el numerador pues K es infinito y al ser $\boldsymbol{\lambda < \mu}$ siempre el denominador tiende a ser uno y por consiguiente desaparecer. mientras que en el sistema **M/M/1/K**/∞ K se mantiene y mantenemos toda la formula.

Ahora que tenemos las sumatorias definidas podemos ver que se trata de una serie geométrica de la forma:

$$n * r^{n} * (1 - r)$$

Donde r es $\boldsymbol{\lambda/\mu}$**.** utilizando wolfram para resolver la serie geométrica en cada uno de esto casos obtenemos que para el sistema **M/M/1**/∞/∞**:**

$$N_s = \frac{\boldsymbol{\lambda}}{\boldsymbol{\mu}-\boldsymbol{\lambda}}$$

Mientras que para el sistema **M/M/1/K**/∞

$$N_s = \boldsymbol{\lambda\mu}\frac{1-(\frac{\boldsymbol{\lambda}}{\boldsymbol{\mu}})^{k}(k(1-\frac{\boldsymbol{\lambda}}{\boldsymbol{\mu}})+1)}{\boldsymbol{\mu}(\boldsymbol{\mu}-\boldsymbol{\lambda})(1-(\frac{\boldsymbol{\lambda}}{\boldsymbol{\mu}})^{k+1})}$$

Con esto ya es posible determinar parámetros para $\boldsymbol{\lambda, \mu}$ **y k ,** reemplazarlos tanto en código como en las fórmulas y verificar la validez de los resultados. para los calculos numericos ver el anexo utils.py un script que realiza todos los calculos segun los parametros ingresados para hacer mas rapida la comparación

M/M/1/∞/∞:

Parámetros/ Valor	Numérico	Simulación
λ = 0.1 **μ = 0.2**	Ns = 1 Ts = 10 Tq = 5 Nq = 0.5	Ns = 1.078 Ts = 10.635 Tq = 5.555 Nq = 0.564
λ = 2 **μ = 3**	Ns = 2 Ts = 1 Tq = 0.666 Nq = 1.333	Ns = 2.016 Ts = 1.008 Tq = 0.675 Nq = 1.350
λ = 10 **μ = 17**	Ns = 1.429 Ts = 0.143 Tq = 0.084 Nq = 0.84	Ns = 1.436 Ts = 0.144 Tq = 0.085 Nq = 0.847

M/M/1/k/∞

Parámetros/ Valor	Numérico	Simulación
λ = 0.1 **μ = 0.2** **K = 5**	Ns = 0.905 Ts = 9.048 Tq = 4.048 Nq = 0.405	Ns = 0.939 Ts = 9.373 Tq = 4.404 Nq = 0.441
λ = 2 **μ = 3** **K = 5**	Ns = 1.422 Ts = 0.711 Tq = 0.378 Nq = 0.756	Ns = 1.438 Ts = 0.756 Tq = 0.422 Nq = 0.803
λ = 10 **μ = 17** **K = 5**	Ns = 1.169 Ts = 0.117 Tq = 0.058 Nq = 0.581	Ns = 1.166 Ts = 0.121 Tq = 0.061 Nq = 0.596

Obteniendo asi un error promedio para el sistema sin restricciones de

- **Ns: 0.034**
- **Ts: 0.215**

- **Tq: 0.188**
- **Nq: 0.029**

Para el sistema con capacidad K limitada:

- **Ns: 0.018**
- **Ts: 0.125**
- **Tq: 0.134**
- **Nq: 0.033**

Conclusiones

La simulación de sistemas de colas M/M/1/∞/∞ y M/M/1/K/∞ proporciona información valiosa sobre la dinámica y el rendimiento de los sistemas de colas de un solo servidor con capacidades ilimitadas y limitadas. Los hallazgos clave incluyen:

- Utilización del sistema: Ambas simulaciones permiten analizar la eficiencia con la que se utiliza el servidor y la frecuencia con la que está inactivo, lo que proporciona implicaciones para la asignación de recursos y el tamaño del sistema.
- Impacto de los límites de capacidad: El modelo M/M/1/K/∞ destaca las consecuencias de la capacidad finita del sistema, incluido el potencial de pérdida de clientes y los efectos de limitar la longitud de las colas en la calidad general del servicio.
- Dinámica de colas: el cálculo de las métricas NQ y TQ ofrece una comprensión más profunda de los comportamientos de las colas, los tiempos de espera y sus implicaciones para la satisfacción del cliente y el rendimiento del sistema.

Con respecto a las simulaciones y su comparación con respecto a los valores numéricos esperados podemos decir que se obtuvieron resultados satisfactorios y un error relativamente bajo que nos indica que la simulación planteada es una aproximación correcta de un sistema real.