UNIVERSIDAD DE SALAMANCA

FACULTAD DE CIENCIAS

DEPARTAMENTO DE ESTADÍSTICA

TRABAJO DE FIN DE GRADO

SISTEMAS DE COLAS G|G: MODELIZACIÓN Y SIMULACIÓN

Tutor: Miguel Rodríguez Rosa

Autora: María Salgado Santamaría

Grado en Estadística

Curso académico 2023-24

UNIVERSIDAD DE SALAMANCA

FACULTAD DE CIENCIAS

DEPARTAMENTO DE ESTADÍSTICA

TRABAJO DE FIN DE GRADO

SISTEMAS DE COLAS G|G: MODELIZACIÓN Y SIMULACIÓN

Firmado por:

Autora: María Salgado Santamaría

Tutor: Miguel Rodríguez Rosa

Grado en Estadística

Curso académico 2023-24

Certificado del tutor TFG Grado en Estadística

D. Miguel Rodríguez Rosa, profesor del Departamento de Estadística de la Universidad de Salamanca,

HACE CONSTAR:

Que el trabajo titulado "*Sistemas de colas G|G: Modelización y simulación*", que se presenta, ha sido realizado por D.ª María Salgado Santamaría, con DNI 70926034Z y constituye la memoria del trabajo realizado para la superación de la asignatura Trabajo de Fin de Grado en Estadística en esta Universidad.

Salamanca, a 3 de julio de 2024.

RODRIGUEZ ROSA MIGUEL - 70821913Z

Firmado digitalmente por RODRIGUEZ ROSA MIGUEL - 70821913Z
Fecha: 2024.07.03 18:06:24 +02'00'

Fdo.: Miguel Rodríguez Rosa

Índice general

Resumen

La teoría de colas es un campo de estudio dentro de la teoría de la probabilidad que se enfoca en analizar y modelar sistemas de espera con el objetivo de optimizar su eficiencia y rendimiento. Estos sistemas involucran la llegada de entidades (clientes, paquetes, solicitudes, etc.) a un servicio o recurso limitado, donde deben esperar si el servicio está ocupado. Esta teoría encuentra aplicaciones que van desde las situaciones más cotidianas, como esperar en la cola del supermercado, hasta sectores avanzados como los servicios y la tecnología de la información.

En los primeros capítulos de este trabajo, se desarrolla el marco teórico que abarca los modelos G|G. En este tipo de sistemas, los tiempos entre llegadas y/o los tiempos de servicio no se distribuyen exponencialmente, a diferencia de lo que a menudo se presume al aplicar esta teoría en casos reales. Se estudia la conocida distribución de Erlang y su relación con estos procesos, además de ilustrar estos conceptos mediante ejemplos concretos presentes en la vida real.

Por último, se desarrolla el concepto de simulación y sus nociones básicas. Se explica el funcionamiento de los paquetes 'queuecomputer' y 'simmer' en el lenguaje R, detallando cómo realizar simulaciones de sistemas de colas G|G y estimar sus medidas de rendimiento. Finalmente, se implementa una aplicación web en R que permite a los usuarios emular sistemas con las características deseadas y calcular sus métricas de rendimiento.

Summary

Queuing theory is a field of study within probability theory that focuses on analysing and modelling waiting systems with the objective of optimising their efficiency and performance. These systems involve the arrival of entities (customers, packages, requests, etc.) to a limited service or resource, where they must wait if the service is busy. This theory finds applications ranging from the most everyday situations, such as waiting in the supermarket queue, to advanced sectors such as services and information technology.

In the first chapters of this paper, the theoretical framework covering G|G models is developed. In this type of systems, the inter-arrival and/or service times are not exponentially distributed, contrary to what is often assumed when applying this theory to real cases. The well-known Erlang distribution and its relation to these processes is studied, and these concepts are illustrated by means of concrete real-life examples.

Finally, the concept of simulation and its basic notions are developed. The operation of the 'queuecomputer' and 'simmer' packages in the R language is explained, detailing how to perform simulations of G|G queuing systems and estimate their performance measures. Finally, a web application

is implemented in R that allows users to emulate systems with the desired characteristics and calculate their performance metrics.

Capítulo 1

Introducción

Los primeros vestigios de la teoría de colas aparecieron en 1909, cuando el matemático danés Agner Krarup Erlang publicó un artículo sobre cómo dimensionar de forma más eficiente las líneas y centrales de conmutación telefónica (Erlang, 1909). Medio siglo después, el científico Leonard Kleinrock (1961) aplicó dicha teoría a la conmutación de paquetes, una de las tecnologías básicas de Internet.

Todos hemos tenido que hacer cola en el supermercado, en la oficina de correos o en el banco. Este tipo de esperas resultan muy molestas, pero no sólo quitan tiempo a los clientes, también afectan a la calidad y la economía de los establecimientos. Si le damos un par de vueltas, no sólo las personas esperamos a ser atendidas, los procesos de un ordenador pueden ponerse a la cola para acceder a recursos compartidos o los productos de una cadena de producción. ¿Será más efectivo contratar a 5 o a 10 cajeros? ¿Cuánto tiempo deberían esperar los clientes? La teoría de colas nos permite responder a este tipo de preguntas con criterio.

Por tanto, un sistema de colas es un modelo matemático que analiza las líneas de espera que forman los clientes u otras entidades para ser atendidos por un servidor. Este modelo incluye tanto a los clientes en espera como a aquellos que están siendo actualmente atendidos. El comportamiento de estos sistemas viene determinado por las características de los clientes, la línea de espera y el servicio.

1.1. Características de los sistemas de colas

Características de los clientes

- **Tamaño de la población**: se define como el número de posibles clientes que podrían entrar en el sistema. Puede ser una población finita o infinita.
- **Patrón de llegadas**: puede ser programado o, lo más común, aleatorio. Cuando se trata de un patrón estocástico, basándose en el proceso de nacimiento y muerte, el tiempo entre llegadas consecutivas seguirá una exponencial E($1/\lambda$), donde λ= llegadas/tiempo. Equivalentemente, la generación de llegadas hasta un tiempo t sigue una distribución de Poisson $\mathcal{P}(\lambda)$.
- **Comportamiento**: un factor importante a tener en cuenta es la actitud de los clientes al entrar en el sistema. Pueden ser pacientes y, por lo tanto esperar hasta ser atendidos, o, por el contrario, desistir y no ingresar en el sistema por no hacer cola o perder la paciencia tras un tiempo esperando e irse.

Características de la línea de espera

- **Capacidad**: se determina por el máximo número de clientes que pueden acceder a la cola. En general, se supone infinita para facilitar los análisis ya que se trata de un límite relativamente alto. En el caso en el que la capacidad máxima de la línea de espera es pequeña y por lo tanto es frecuente alcanzarla, se considera una cola finita.
- **Disciplina**: hace referencia al orden en el que se atiende a los clientes. Lo más habitual es atender por orden de llegada, lo que se denomina FIFO (first in first out). Otra opción es la disciplina LIFO (last in first out), que consiste en asistir al último que llega. También puede ser aleatoria, es decir, que no depende del instante de llegada al sistema.

  Cada una de estas disciplinas podría dividirse en dos categorías en función de si existe prioridad o no. Si hay clientes prioritarios, que son atendidos independientemente del instante de llegada, pueden darse dos casos: interrumpir el servicio en curso para concluirlo y empezarlo de nuevo después, o que el cliente prioritario se coloque al principio de la cola y sea el siguiente sin interrupciones.

Características del servicio

- **Patrón de servicios**: al igual que las llegadas, el tiempo de servicio puede ser programado o aleatorio, y estacionario o no estacionario respecto al tiempo o al estado (número de clientes en el sistema). En el caso de que sea aleatorio, basándose de nuevo en el proceso de nacimiento y muerte, los tiempos de servicio siguen una distribución exponencial E($1/\mu$), donde μ= servicios/tiempo o, lo que es lo mismo, una Poisson $\mathcal{P}(\mu)$.

  Supondremos que, en el caso de haber más de un servidor, todos siguen la misma distribución.
- **Número de canales de servicio (servidores)**: el número de servidores puede ser finito o infinito. En el caso de haber más de uno, serán servidores independientes entre sí y se pueden dar dos situaciones: múltiples sistemas con un único servidor (Figura 1.1) o un único sistema con múltiples servidores (Figura 1.2).

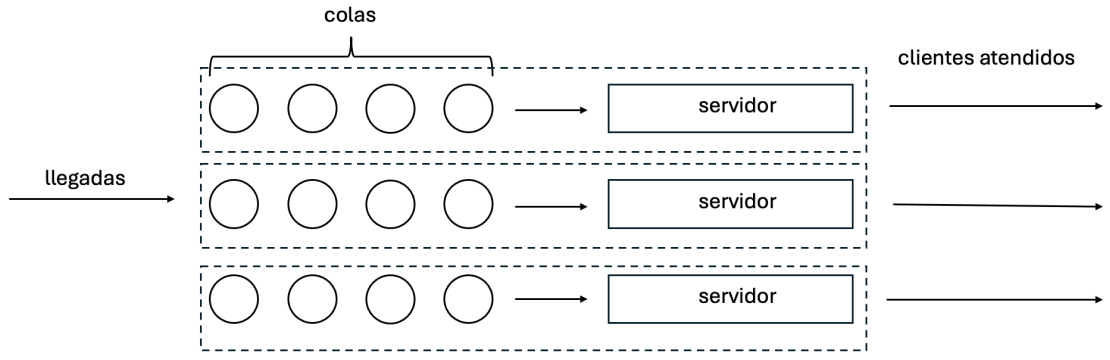

Figura 1.1: Diagrama del caso con múltiples sistemas con un único servidor

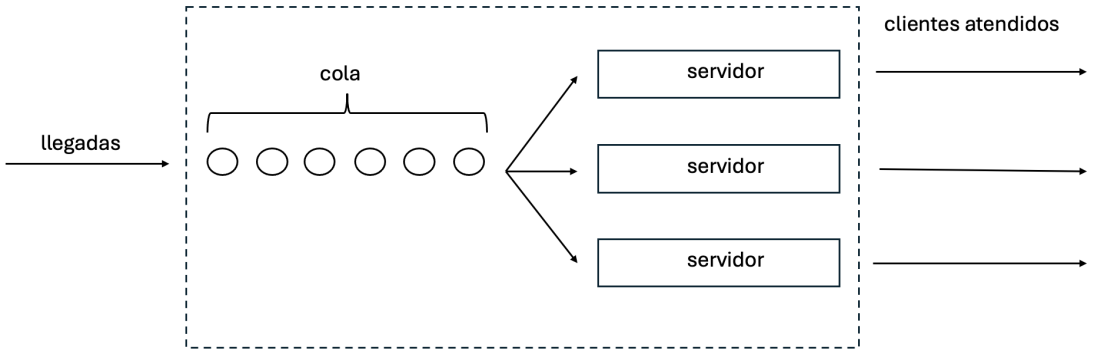

Figura 1.2: Diagrama del caso con un único sistema con múltiples servidores

Para definir correctamente el sistema, también es importante determinar el número de etapas, es decir, determinar si hay más de una estación de servicio y por lo tanto, la posibilidad de colas consecutivas.

1.2. Notación de Kendall

Una vez se conocen las características del sistema, la forma estándar para describirlo es la notación de Kendall (1953). Consiste en definir el proceso con una serie de símbolos de la forma A|B|C|X|Y|Z, donde A representa el patrón de llegadas y B el de servicios, C el número de servidores en paralelo, X la capacidad del sistema, Y la disciplina y Z el número de etapas. Para los símbolos A y B se usará la M de Markov cuando se trata de una distribución exponencial, la G para una genérica, E_k en el caso de Erlang y D para una distribución degenerada (tiempos constantes).

Generalmente, cuando se trata de un sistema de capacidad infinita, con disciplina FIFO y/o una única etapa, se omiten los símbolos correspondientes. Por ejemplo, un proceso M|M|1|∞|FIFO|1, que se escribirá como M|M|1, cuenta con un patrón de llegadas y servicios exponencial, un único canal de servicio, capacidad infinita, disciplina “first in first out” y una única etapa.

1.3. Nomenclatura

La nomenclatura estándar recoge estos símbolos:

λ = llegadas por unidad de tiempo

μ = servicios por unidad de tiempo

c = número de servidores en paralelo

ρ = tasa de ocupación

L = número medio de clientes en el sistema

L_q = número medio de clientes en la cola

W = tiempo medio en el sistema para cada cliente

W_q = tiempo medio en la cola para cada cliente

p_n = probabilidad de que haya n clientes en el sistema

q_n = probabilidad de que haya n clientes en el sistema en el momento en el que entra un nuevo cliente

Para los parámetros λ y μ, en el caso de que no sean constantes, será necesaria su estimación, de forma que:

$$\lambda = \frac{\text{número medio de llegadas}}{\text{unidad de tiempo}} \tag{1.1}$$

$$\mu = \frac{\text{número medio de servicios}}{\text{unidad de tiempo}} = \frac{\text{número medio de servicios}}{W - W_q} \tag{1.2}$$

Así, $1/\lambda$ representa el tiempo esperado entre llegadas y $1/\mu$ el tiempo esperado de servicio. La tasa de ocupación, definida como $\rho = \frac{\lambda}{c \cdot \mu}$, representa la fracción de tiempo que se espera que esté cada servidor ocupado y nos aporta información sobre si se ha alcanzado un estado estacionario o no. Si ρ<1 se trata de un sistema óptimo que ha alcanzado el estado estacionario, y por lo tanto, puede gestionar la llegada de clientes adicionales. Por el contrario, si $\rho \geq 1$, se dice que el sistema está saturado por lo que es incapaz de manejar la demanda.

1.4. Fórmulas de Little

Los parámetros L, L_q, W y W_q son medidas de rendimiento que aportan información sobre la eficacia del sistema. Jhon D. C. Little (1961) demostró la relación entre ellas en el estado estacionario, de forma que conociendo una, es posible calcular las demás. Relacionando el tamaño medio del sistema y los tiempos medios

$$L = \lambda \cdot W = L_q + \frac{\lambda}{\mu} \tag{1.3}$$

$$L_q = \lambda \cdot W_q \tag{1.4}$$

$$W = W_q + \frac{1}{\mu} \tag{1.5}$$

A continuación, se comentarán únicamente ejemplos de modelos basados en el proceso de nacimiento y muerte, de disciplina FIFO y con una única etapa.

1.5. Sistema simple M|M|1

Se trata de un modelo en el que las tiempos de llegada y de servicio siguen una distribución exponencial E($1/\lambda$) y E($1/\mu$) (las tasas de llegadas y servicios siguen una Poisson), un único servidor y capacidad infinita. Una vez determinados los parámetros λ y μ, si ρ=λ/μ <1, las medidas de rendimiento se definen como

$$\begin{aligned} L_q &= \frac{\rho^2}{1-\rho} = \frac{\lambda^2}{\mu \cdot (\mu - \lambda)} & L &= \frac{\rho}{1-\rho} = \frac{\lambda}{\mu - \lambda} \\ W_q &= \frac{L_q}{\lambda} = \frac{\lambda}{\mu \cdot (\mu - \lambda)} & W &= \frac{L}{\lambda} = \frac{1}{\mu - \lambda} \end{aligned} \tag{1.6}$$

1.6. Sistema con servidores en paralelo M|M|c

Se trata de un caso similar al anterior pero con c servidores en paralelo. En el caso de que no se trate de un sistema saturado, es decir, que $\rho = \lambda/c \cdot \mu < 1$

$$L_q = \frac{(\frac{\lambda}{\mu})^c \cdot \lambda \cdot \mu}{(c-1)! \cdot (c \cdot \mu - \lambda)^2} \cdot p_0 \qquad L = L_q + \frac{\lambda}{\mu}$$
$$W_q = \frac{L_q}{\lambda} = \frac{(\frac{\lambda}{\mu})^c \cdot \mu}{(c-1)! \cdot (c \cdot \mu - \lambda)^2} \cdot p_0 \qquad W = \frac{L}{\lambda} = W_q + \frac{1}{\mu} \tag{1.7}$$

Donde

$$p_0 = \left[\left(\sum_{i=0}^{c-1} \frac{\lambda^i}{\mu^i \cdot i!} \right) + \frac{1}{c!} \cdot \left(\frac{\lambda}{\mu} \right)^c \cdot \frac{c \cdot \mu}{c \cdot \mu - \lambda} \right]^{-1} \tag{1.8}$$

1.7. Sistemas con capadidad finita M|M|c|k

En este caso, el sistema tiene una capacidad limitada y sólo permite la entrada de k clientes. Cuando $\rho \neq 1$ es necesario el cálculo de la tasa efectiva de llegadas λ', que tiene en cuenta cómo afecta una capacidad limitada a las llegadas y al rendimiento general del sistema. Este nuevo parámetro se define como

$$\lambda' = \lambda \cdot (1 - p_k) \tag{1.9}$$

En el caso de un sistema simple con un sólo servidor

$$p_k = \frac{(1-\rho) \cdot \rho^k}{1 - \rho^{k+1}} \tag{1.10}$$

$$L_q = \frac{\rho \cdot \left[1 - (k+1) \cdot \rho^k + k \cdot \rho^{k+1}\right] - \rho \cdot (1-\rho) \cdot \left(1 - \rho^k\right)}{\left(1 - \rho^{k+1}\right) \cdot (1-\rho)}$$
$$L = \frac{\rho \cdot \left[1 - (k+1) \cdot \rho^k + k \cdot \rho^{k+1}\right]}{(1 - \rho^{k+1} \cdot (1-\rho)} \tag{1.11}$$

Cuando se trata de un modelo con c servidores en paralelo

$$p_0 = \left[\left(\sum_{i=0}^{c-1} \frac{\lambda^i}{\mu^i \cdot i!} \right) + \frac{1}{c!} \cdot \left(\frac{\lambda}{\mu} \right)^c \cdot \frac{1 - \rho^{k-c+1}}{1-\rho} \right]^{-1} \tag{1.12}$$

$$L_q = \frac{(c \cdot \rho)^c \cdot \rho}{c! \cdot (1-\rho)^2} \cdot \left[1 - \rho^{k-c+1} - (1-\rho) \cdot (k-c+1) \cdot \rho^{k-c}\right] \cdot p_0 \tag{1.13}$$

$$L = L_q + c - p_0 \cdot \sum_{i=0}^{c-1} \frac{(c-i) \cdot (c \cdot \rho)^i}{i!} \tag{1.14}$$

En ambas situaciones, el tiempo medio en la cola W_q y el tiempo medio en el sistema W se definen

como

$$W_q = \frac{L_q}{\lambda'} \qquad W = \frac{L}{\lambda'} \tag{1.15}$$

Como se ha comentado antes, en estos cuatro ejemplos el tiempo entre llegadas consecutivas y de servicios siguen una exponencial ya que se basan en el proceso de nacimiento y muerte. Sin embargo, al trasladar esta teoría a un caso real, cabe la posibilidad de que esto no sea así, lo que se denomina un sistema de colas G|G. Cuando no trabajamos con una distribución Exponencial o una Poisson, los análisis pueden complicarse, pero no es imposible. Los análisis y aproximaciones en los que profundizaremos en este trabajo son los que nos permitirán estudiar y trabajar con algunos de estos casos.

Una vez definidos los términos principales de este trabajo, veremos su estructura:

Tras un primer apartado con los objetivos del trabajo, nuestro primer capítulo aborda el marco teórico que abarca los modelos G|G, analizando cada uno de los tipos y exponiendo ejemplos de los mismos en el mundo real.

Nuestro segundo capítulo hablará del concepto de simulación, específicamente la simulación de modelos basados en eventos distretos (DES), como es el caso de un sistema de colas. Tras definir los conceptos básicos necesarios para entender el algoritmo, se explorarán los paquetes de R 'simmer' y 'queuecomputer', diseñados para la simulación de DES. Finalmente, se explicará la implementación y funcionamiento de una aplicación web creada con RShiny, la cual permitirá al usuario simular y analizar el rendimiento de sistemas con las características que prefiera.

En nuestro capítulo final se encuentran detalladas las conclusiones obtenidas a partir del desarrollo y análisis realizados.

Por último, se ofrecerá la bibliografía utilizada para la realización de este trabajo, en esta encontraremos diversos libros, artículos, y páginas web.

Capítulo 2

Objetivos

En este trabajo tendremos los siguientes objetivos:

- Realizar un estudio exhaustivo de las situaciones en las que se presentan distribuciones no exponenciales (sistemas G|G) e identificar las distribuciones alternativas relevantes.
- Desarrollar una teoría sólida, derivando las condiciones de no saturación, las ecuaciones de tráfico y las medidas de rendimiento pertinentes.
- Ilustrar estas nociones mediante ejemplos concretos que reflejen situaciones del mundo real.
- Implementar un programa en R para generar bases de datos simuladas que reproduzcan escenarios típicos de sistemas G|G, y que realice análisis detallados de los mismos.

Capítulo 3

Modelos de colas con distribuciones no exponenciales

3.1. Sistema M|G|1

En este modelo de un único servidor, se considera que el patrón de entradas al sistema es aleatorio y sigue una distribución de Poisson con una tasa media de λ llegadas/unidad de tiempo (los tiempos entre llegadas son exponenciales). Sin embargo, pese a que se siguen considerando independientes y con una misma distribución de probabilidad, los tiempos de servicio no se definen como exponenciales, cuentan con una distribución genérica.

Definiendo el tiempo de servicio como la variable aleatoria S y $\mu = \frac{1}{E[S]}$ la tasa media de servicios, asumimos que el sistema no está saturado, por lo que $\rho = \frac{\lambda}{\mu} < 1$.

El proceso consiste en derivar las llamadas fórmulas de Pollaczek-Khintchine (Khintchine, 1932; Pollaczek, 1930) que nos permitan calcular el valor esperado para alguna de las medidas de rendimiento y, posteriormente, aplicar las fórmulas de Little para obtener las demás.

Si pensamos en el tiempo que esperará un cliente que acaba de llegar al sistema, tenemos que tener en cuenta a aquellos que están en la cola (L_q) y a los que están siendo atendidos en ese momento.

Los L_q clientes que están en la cola, esperarán una media de E[S] cada uno, por lo que el tiempo de espera promedio del recién llegado será de $L_q \cdot E[S]$.

En el caso de que el servidor esté ocupado en ese momento, hay que añadir el tiempo de servicio restante que le queda al cliente que está siendo atendido, pero estos tiempos no pueden considerarse siempre iguales. De forma que el tiempo de espera promedio W_q será

$$W_q = L_q \cdot E[S] + P(\text{servidor ocupado}) \cdot E[\text{tiempo de servicio restante}|\text{servidor ocupado}]$$

Aplicando $L = \lambda \cdot W = L_q + \dfrac{\lambda}{\mu}$

$$W_q = \frac{P(\text{servidor ocupado}) \cdot E[\text{tiempo de servicio residual}|\text{servidor ocupado}]}{1 - \rho}$$

La probabilidad de que el servidor esté ocupado coincide con la tasa de ocupación, luego $P(\text{servidor ocupado}) = \rho$.

Por otro lado, el tiempo residual medio condicionado a que el servidor esté ocupado puede definirse como

$$E[\text{tiempo de servicio residual}|\text{servidor ocupado}] = \frac{E[S^2]}{2 \cdot E[S]} = \frac{1 + C_B^2}{2} \cdot E[S] \tag{3.1}$$

Donde C_B^2 es el cuadrado del coeficiente de variación: $\frac{Var[S]}{E^2[S]}$.

Considerando que $E[S] = \frac{1}{\mu}$, el tiempo de espera promedio de un cliente que acaba de llegar al sistema será

$$W_q = \frac{\rho}{\mu - \lambda} \cdot \frac{1 + C_B^2}{2} \tag{3.2}$$

Aplicando las fórmulas de Little

$$L_q = \frac{\rho^2}{1 - \rho} \cdot \frac{1 + C_B^2}{2} \tag{3.3}$$

$$W = \frac{\rho}{\mu - \lambda} \cdot \frac{1 + C_B^2}{2} + \frac{1}{\mu} \tag{3.4}$$

$$L = \frac{\rho^2}{1 - \rho} \cdot \frac{1 + C_B^2}{2} + \rho \tag{3.5}$$

En la siguiente tabla (Figura 3.1) vienen distintas expresiones de las medidas de rendimiento, todas equivalentes entre sí. En la segunda columna se utiliza el segundo momento de la distribución de servicio $E[S^2]$ y en la tercera, su varianza σ_B^2.

L_q	$= \frac{1 + C_B^2}{2} \cdot \frac{\rho^2}{1 - \rho}$	$= \frac{\lambda^2 \mathrm{E}[S^2]}{2(1 - \rho)}$	$= \frac{\rho^2 + \lambda^2 \sigma_B^2}{2(1 - \rho)}$
W_q	$= \frac{1 + C_B^2}{2} \cdot \frac{\rho}{\mu - \lambda}$	$= \frac{\lambda \mathrm{E}[S^2]}{2(1 - \rho)}$	$= \frac{\rho^2/\lambda + \lambda \sigma_B^2}{2(1 - \rho)}$
W	$= \frac{1 + C_B^2}{2} \cdot \frac{\rho}{\mu - \lambda} + \frac{1}{\mu}$	$= \frac{\lambda \mathrm{E}[S^2]}{2(1 - \rho)} + \frac{1}{\mu}$	$= \frac{\rho^2/\lambda + \lambda \sigma_B^2}{2(1 - \rho)} + \frac{1}{\mu}$
L	$= \frac{1 + C_B^2}{2} \cdot \frac{\rho^2}{1 - \rho} + \rho$	$= \frac{\lambda^2 \mathrm{E}[S^2]}{2(1 - \rho)} + \rho$	$= \frac{\rho^2 + \lambda^2 \sigma_B^2}{2(1 - \rho)} + \rho$

Figura 3.1: Fórmulas de Pollaczek-Khintchine para el modelo M|G|1

Ejemplo 1. En una sucursal del Banco Central, opera un cajero automático esencial para los clientes diarios. Los tiempos que cada cliente pasa en el cajero hasta completar una transacción siguen una distribución Normal, con una media de 8 minutos y una desviación estándar de 2 minutos. Los clientes llegan al cajero con una tasa promedio de 6 clientes por hora, siguiendo un proceso de Poisson.

El banco, ha decidido evaluar si merece la pena mantener este cajero. Para este propósito, han contratado a un analista estadístico para calcular las medidas de rendimiento necesarias.

$$\lambda = \frac{6}{60} = 0{,}1 \text{ clientes por minuto}$$

$$\mathrm{E[S]} = \frac{1}{\mu} = 8 \text{ minutos}$$

$$\rho = \frac{\lambda}{\mu} = \frac{0{,}1}{1/8} = 0{,}8 < 1 \text{ luego el sistema no satura}$$

$$C_B^2 = \frac{2^2}{8^2} = 0{,}0625$$

$$W_q = \frac{0{,}8}{1/8 - 0{,}1} \cdot \frac{1 + 0{,}0625}{2} = 17 \text{ minutos de media en la cola}$$

de forma que aplicando las fórmulas de Little

$$W = W_q + \frac{1}{\mu} = 17 + 8 = 25 \text{ minutos de media en el sistema}$$

$$L_q = \lambda \cdot W_q = 0{,}1 \cdot 17 = 1{,}7 \text{ clientes de media en la cola}$$

$$L = \lambda \cdot W = 0{,}1 \cdot 25 = 2{,}5 \text{ clientes de media en el sistema}$$

3.2. Sistema M|G|c

En los modelos con múltiples servidores (c servidores), el desarrollo matemático se complica, ya que el número de llegadas entre salidas ya no es exclusivamente dependiente del tamaño del sistema en el momento de la salida anterior. Aún así, se han podido obtener algunos resultados generales para los casos en los que se conoce la distribución G que siguen los tiempos de servicio.

Para llegar a las medidas de rendimiento en el caso de un modelo M|G|c hay que basarse en la relación lineal existente entre los k-ésimos momentos factoriales del tamaño del sistema y los k-ésimos momentos regulares del tiempo de espera en el sistema. Dicha relación se traduce de la forma

$$L_{(k)} = \lambda^k \cdot W_k \tag{3.6}$$

La demostración de dichos resultados se encuentra desarrollada en el quinto capítulo del libro "Fundamentals of Queueing Theory"(Gross et al., 2008).

Ejemplo 2. TechSolutions es una empresa líder en tecnología que ofrece soporte técnico a sus clientes a través de un centro de llamadas.

En el último año, el centro de llamadas ha registrado un promedio de 20 llamadas por hora, las cuales llegan de acuerdo a un proceso de Poisson. Los tiempos de atención de las llamadas, que varían dependiendo de la complejidad de los problemas, siguen una distribución gamma con un promedio de 5 minutos y una desviación estándar de 2 minutos. Para mantener un servicio eficiente, TechSolutions cuenta con 4 operadores altamente capacitados que atienden las llamadas de soporte.

Para mejorar la experiencia del cliente y optimizar los tiempos de respuesta, la empresa desea conocer el tiempo promedio que un cliente espera hasta que es atendido y cuánto dura el servicio.

Al no existir fórmulas cerradas para las medidas de rendimiento de un sistema M|G|c, como es este caso, este escenario complejo requeriría el uso de resultados tabulados obtenidos mediante métodos numéricos precisos. Es por ello, que más adelante se abordará la simulación de estos sistemas para ilustrar su comportamiento y análisis de manera detallada. Mediante la aplicación web diseñada en este trabajo, este tipo de problemas serán fácilmente resueltos.

3.3. Sistema M|D|c

En este modelo con c servidores y un proceso de llegadas de Poisson, se considera que los tiempos de servicio siguen una distribución degenerada. Esto significa que no cuentan con un patrón de servicios aleatorio, sino que está programado y todos los tiempos vienen determinados por la misma constante.

El sistema M|D|1 no es más que un caso especial de M|G|1, comentado anteriormente, en el que la varianza de la distribución que sigue la variable aleatoria S es cero. Por lo que si $\sigma_B^2 = 0$, λ es la tasa media de llegadas y $\rho = \dfrac{\lambda}{\mu \cdot c} < 1$, el tiempo promedio en la cola será

$$W_q = \frac{\rho^2/\lambda}{2 \cdot (1-\rho)} \tag{3.7}$$

Una vez calculado el tiempo medio que pasa un cliente en la cola, se pueden obtener el resto de medidas de rendimiento aplicando las fórmulas de Little (1.3, 1.4 y 1.5).

En el caso de que haya más de un servidor (c servidores), la derivación del número promedio de clientes en la cola consiste en un método complicado y no se han encontrado expresiones generales para el estado estable $\left(\rho = \frac{\lambda}{c\mu} < 1\right)$. Aún así, existen tabulaciones para casos numéricos concretos. Algunos de estos resultados, podemos encontrarlos en el libro "Introduction to Operations Research" de los autores Hillier y Lieberman (2009).

Ejemplo 3. En la autopista A-10, una estación de peaje cuenta con un único carril dirigido por un empleado encargado de cobrar el peaje. La administración de la autopista está interesada en evaluar el desempeño de esta estación, considerando el flujo de vehículos que la atraviesan diariamente.

Los vehículos llegan a la estación de peaje de manera aleatoria, siguiendo un proceso de Poisson con una tasa promedio de llegada de 20 vehículos por hora. Por otro lado, el tiempo que el empleado tarda en atender a cada vehículo es constante y se distribuye de manera determinística (degenerada), con un tiempo de servicio de 30 segundos por vehículo.

$$\lambda = \frac{20}{60} = \frac{1}{3} \text{ coches por minuto}$$

$$\mu = \frac{60}{30} = 2 \text{ coches por minuto}$$

$$\rho = \frac{1/3}{2} = \frac{1}{6} \text{ <1 luego el sistema no satura}$$

$$W_q = \frac{\left(\frac{1}{6}\right)^2 / \frac{1}{3}}{2 \cdot \left(1 - \frac{1}{6}\right)} = \frac{1}{20} \text{ minutos = 3 segundos de media en la cola}$$

De forma que aplicando las fórmulas de Little

$$L_q = \lambda \cdot W_q = \frac{1}{3} \cdot \frac{1}{20} = \frac{1}{60} \text{ coches de media en la cola}$$

$$W = W_q + \frac{1}{\mu} = \frac{1}{20} + \frac{1}{2} = \frac{11}{20} \text{ minutos = 33 segundos de media en el sistema}$$

$$L = \lambda \cdot W = \frac{1}{3} \cdot \frac{11}{20} = \frac{11}{60} \text{ coches de media en el sistema}$$

3.4. Sistema G|M|1

En este modelo no se imponen restricciones en cuanto a la distribución de los tiempos de llegadas, es decir, en este caso estudiaremos un sistema de colas en el que las llegadas al sistema no siguen un proceso de Poisson. Asumiendo que los tiempos de servicio son exponenciales y tienen una media de $\frac{1}{\mu}$ y que el sistema de un único servidor no está saturado (tasa de ocupación menor que 1), denotaremos X_n como el número de clientes que hay en el sistema en el momento en el que entra el n-ésimo cliente. B_n será el número de clientes que son atendidos y salen del sistema en un tiempo $T^{(n)}$ transcurrido entre la llegada n y la $n+1$ (puede denotarse por T ya que todos los tiempos entre llegadas son independientes y están igualmente distribuidos), de forma que

$$X_{n+1} = X_n + 1 - B_n \ \ (B_n \leq X_n + 1, X_n \geq 0)$$

Teniendo en cuenta que B_n es una variable aleatoria que no depende del pasado de la cola, dado el número de clientes en el sistema en el momento de la llegada del n-ésimo cliente, $\{X_0, X_1, X_2, \ldots\}$ es una cadena de Markov.

Siendo $A(t)$ la función de distribución acumulativa de T, podemos definir la probabilidad de que se concluyan k servicios entre dos llegadas consecutivas de la forma

$$b_k = P(B_n = k | X_n \leq k) = \int_0^\infty \frac{e^{-\mu t} \cdot (\mu t)}{k!} \, dA(t) \tag{3.8}$$

Si q es el vector de probabilidad de que un recién llegado se encuentre con n clientes ya en el sistema, y M es la matriz de transición de un único paso

$$M = \left\{ p_{ij} \right\} = \begin{pmatrix} 1 - b_0 & b_0 & 0 & 0 & \cdots \\ 1 - \sum_{k=0}^{1} b_k & b_1 & b_0 & 0 & \cdots \\ 1 - \sum_{k=0}^{2} b_k & b_2 & b_1 & b_0 & \cdots \\ \vdots & \vdots & \vdots & \vdots & \ddots \end{pmatrix}$$

$$q = \{q_n\}, n = 0, 1, 2, 3, \ldots$$

Considerando el estado estacionario

$$qM = q, \ q\vec{1} = 1 \tag{3.9}$$

por lo que

$$\begin{cases} q_i = \sum_{k=0}^{\infty} q_{i+k-1} \cdot b_k & (i \geq 1) \\ q_0 = \sum_{j=0}^{\infty} q_j \left(1 - \sum_{k=0}^{j} b_k\right) & (i = 0) \end{cases} \tag{3.10}$$

La resolución del sistema se encuenta en el capítulo cinco de "Fundamentals of Queueing Theory" (Gross et al., 2008), en el que se concluye que la distribución del punto de llegada en estado estacionario será

$$q_n = (1 - r_0) \cdot r_0^n \ \ (\text{con } n \geq 0, \rho < 1) \tag{3.11}$$

Siendo r_0 una raíz obtenida en la resolución del sitema (3.9) que se enuentra en el intervalo $(0, 1)$. Si trasladamos esta conclusión al caso M|M|1, obtenemos que en dicho modelo $r_0 = \rho$, por lo que la probabilidad de que haya n clientes en el sistema en estado estacionario sería $p_n = (1 - \rho) \cdot \rho^n$.

Sustituyendo ρ por r_0 en las fórmulas planteadas para el modelo M|M|1 (1.6), obtenemos las si-

guientes medidas de rendimiento en cuanto a los tiempos del sistema

$$W_q = \frac{r_0}{\mu \cdot (1 - r_0)} \tag{3.12}$$

$$W = \frac{1}{\mu \cdot (1 - r_0)} \tag{3.13}$$

Es importante tener en cuenta que hemos definido q_n como la probabilidad de que haya n clientes en el sistema justo antes de que entre otra llegada, no en general, por lo que $q_n \neq p_n$ (no como en el modelo M|G|1, que dicha igualdad sí se cumple). Esto implica que aplicar las fórmulas de Little para relacionar L y L_q con W y W_q sería un error. Para ello, será necesario agregar el superíndice (A) para especificar que se están calculando en relación a puntos de llegada y definir dichas medidas como

$$L^{(A)} = \frac{r_0}{1 - r_0} \tag{3.14}$$

$$L_q^{(A)} = \frac{r_0^2}{1 - r_0} \tag{3.15}$$

Ejemplo 4: En la fábrica Antolín de Burgos, hay una enfermería administrada por la aseguradora de la empresa, a la cual los trabajadores pueden acudir durante su jornada laboral. Un especialista fue designado para determinar el tiempo promedio de espera de los empleados para ser atendidos. Este análisis reveló que el tiempo de atención a los obreros sigue una distribución exponencial con una media de 14 minutos por consulta. Sin embargo, la distribución del tiempo entre las llegadas de los obreros no fue especificada.

Además, la empresa quiere aprovechar los datos para estimar el número de obreros que visitan la enfermería, con el fin de evaluar si es viable mantener este servicio considerando el gasto asociado.

Al no existir fórmulas cerradas para las medidas de rendimiento de un sistema G|M|1, como es este caso, este escenario complejo requeriría el uso de resultados tabulados obtenidos mediante métodos numéricos precisos. Es por ello, que más adelante se abordará la simulación de estos sistemas para ilustrar su comportamiento y análisis de manera detallada. Mediante la aplicación web diseñada en este trabajo, este tipo de problemas serán fácilmente resueltos.

3.5. Sistema G|M|c

Cuando contamos con un modelo similar al anterior, pero con c servidores en vez de uno, gran parte del desarrollo matemático se mantiene igual, excepto el valor de b_k. La tasa media de servicio ahora ya no es μ si no que pasa a ser $c\mu$ o $n\mu$ en función del estado del sistema, luego b_n depende de i y de j, lo que implica una matriz de transición de la cadena de Markov bastante diferente.

Todo $p_{ij} = 0$ cuando $j > i + 1$, pero cuando $c \leq j \leq i + 1$ implica que todos los servidores están ocupados. Este es el caso en el que tenemos una tasa de servicio de la forma $c\mu$, de forma que

$$p_{ij} = b_{i+1-j} \text{ (donde } c \leq j \leq i + 1) \tag{3.16}$$

$$b_n = \int_0^\infty \frac{e^{-c\mu t} \cdot (c\mu t)^n}{n!} \, dA(t) \tag{3.17}$$

Realmente, la resolución de los cálculos es la misma que en el caso de G|M|1, a diferencia de que hay que incluir el término c. El principal objetivo sigue siendo el mismo, obtener las raíces del sistema resultante.

De nuevo, la resolución del sistema se encuentra en el capítulo cinco del libro "Fundamentals of Queueing Theory" (Gross et al., 2008), concluyendo con

$$W_q = \frac{q_c}{c\mu\,(1 - r_0)^2} \tag{3.18}$$

Resulta interesante comentar que otros autores como Cohen (1973) y Ross (1978) son también importantes referencias en lo que a resolución de problemas de modelos G|M|c se refiere, ya que ofrecen otra perspectiva para resolver este modelo.

Ejemplo 5. El Mercadona de la Gran Vía de Madrid cuenta con tres cajeros operativos. Cada año, realizan un estudio de mercado en el que uno de los apartados principales consiste en analizar el rendimiento del supermercado. Tras varias semanas recopilando datos, el encargado se ha dado cuenta de que las llegadas de los clientes a la cola siguen una distribución Normal, con una media de 32 clientes por hora y desviación típica de 4. Por el contrario, el tiempo que tarda un cajero en cobrar a un cliente sigue una distribución exponencial de media 7 minutos.

Al no existir fórmulas cerradas para las medidas de rendimiento de un sistema G|M|c, como es este caso, este escenario complejo requeriría el uso de resultados tabulados obtenidos mediante métodos numéricos precisos. Es por ello, que más adelante se abordará la simulación de estos sistemas para ilustrar su comportamiento y análisis de manera detallada. Mediante la aplicación web diseñada en este trabajo, este tipo de problemas serán fácilmente resueltos.

3.6. La distribución de Erlang

En los casos que se han desarrollado previamente, se han estudiado modelos en los que los tiempos entre llegadas o de servicio no eran exponenciales, si no que seguían una distribución genérica. Para estos casos, es común trabajar con la conocida distribución de Erlang.

La distribución de Erlang consiste en una variable continua T cuya función de densidad se expresa en función de los parámetros R y k.

$$f(t) = \frac{R(Rt)^{k-1}e^{-Rt}}{(k-1)!} \tag{3.19}$$

El parámetro R será una tasa, y k, también denominado parámetro de forma, controla cuánto varían los tiempos de servicio alrededor de la media. Ambos son estrictamente positivos, y el parámetro k está restringido a números enteros, una de las pocas diferencias que distingue esta distribución de la Gamma. Si integramos por partes para obtener la media y la varianza, entonces

$$E[T] = \frac{k}{R} \qquad y \qquad Var[T] = \frac{k}{R^2} \tag{3.20}$$

Considerando estos resultados, podemos comprobar cómo el parámetro k cambia la forma de la distribución. Supongamos una tasa de llegadas λ dada y un conjunto de variables que siguen una Erlang con parámetro k y $R = k\lambda$. A medida que el parámetro de forma varía, también lo hace la forma de la distribución de Erlang (Figura 3.2).

Si nos fijamos, si $k = 1$ la función de densidad se asemeja mucho a una Exponencial y a medida que k crece, la forma de la función se acerca a la de una Normal. De hecho, cuando $k = 1$, se trata de una distribución exponencial de parámetro $\frac{1}{\lambda}$ y en el caso de $k \to \infty$ estaremos hablando de

una distribución degenerada, donde el tiempo entre llegadas es una constante con una varianza de cero.

La distribución de Erlang resulta muy útil en cuanto a sistemas de colas se refiere, debido a que la gran mayoría de las distribuciones de tiempos entre llegadas y de servicio caen dentro del intervalo que forman el M|D|s y el M|M|s, es decir, $0 < \sigma < \frac{1}{\lambda}$ si hablamos de tiempos entre llegadas, y $0 < \sigma < \frac{1}{\mu}$ si hablamos de tiempos de servicio.

La distribución de Erlang de parámetro de forma k y de tasa $k\lambda$ también podría verse como la suma de A_i, variables independientes que siguen una idéntica distribución exponencial de parámetro $k\lambda$. Es decir, si modelamos el patrón de tiempos entre llegadas con una Erlang, es lo mismo que decir que el cliente pasa por k fases hasta entrar al sistema y, en el caso de ser los tiempos de servicio los que siguen una Erlang de parámetros k y $k\mu$, se considera que el servicio consta de k tareas iguales (o fases) hasta completarse.

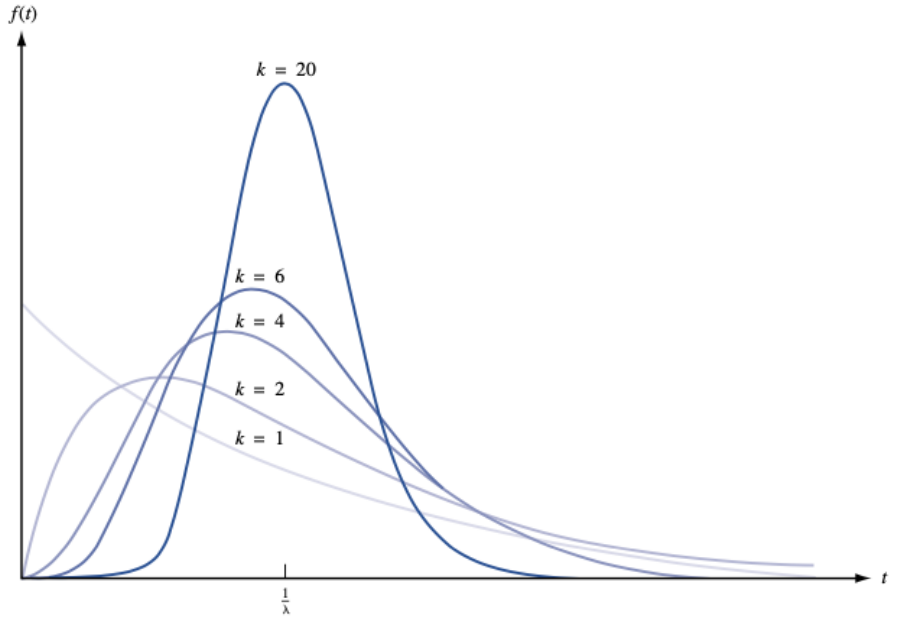

Figura 3.2: Distribución de Erlang para distintos valores de k

3.7. Sistema M|E_K|c

Consideremos un modelo con un proceso de llegadas de Poisson y c servidores, pero sin ninguna restricción sobre la distribución de los tiempos de servicio, es decir, un modelo M|G|c.

En este caso, el valor de k se encuentra dentro del intervalo $1 < k < \infty$, luego nos podemos estar refiriendo a una Erlang de media $\frac{1}{\mu}$ y varianza $\frac{1}{k\mu^2}$. Luego, si se estiman la media y varianza de la distribución empírica (G), podemos obtener un valor de k para ajustar una Erlang a nuestro patrón de servicios.

Pongamos el ejemplo de un sistema M|E_k|1 (que no deja de ser un modelo M|G|1). Conociendo los parámetros k, μ y λ y aplicando las fórmulas de Pollaczek-Khintchine para un modelo con patrón de servicios no exponencial en función de σ^2 (Figura 3.1)

$$W_q = \frac{\frac{\lambda^2}{k\mu^2} + \rho^2}{2(1-\rho)} = \frac{1+k}{2k} \cdot \frac{\lambda}{\mu(\mu-\lambda)} = \frac{1+1/k}{2} \cdot \frac{\rho}{\mu(1-\rho)} \tag{3.21}$$

$$L_q = \frac{\frac{\lambda}{k\mu^2} + \rho^2/\lambda}{2(1-\rho)} = \frac{1+k}{2k} \cdot \frac{\lambda^2}{\mu(\mu-\lambda)} = \frac{1+1/k}{2} \cdot \frac{\rho^2}{1-\rho} \tag{3.22}$$

$$W = W_q + \frac{1}{\mu} \tag{3.23}$$

$$L = \lambda \cdot W \tag{3.24}$$

En el caso haber más de un servidor, se puede intentar desarrollar un proceso en función de las k fases del servicio, en vez de en términos de los clientes. Aún así, no se han encontrado resultados generales para el estado estable ($\rho = \lambda/c\mu < 1$) que definan el número de clientes esperados en el sistema, por lo que es necesario una resolución numérica para cada caso particular. Algunos resultados se encuentran tabulados en el libro "Introduction to Operations Research" (Hillier & Lieberman, 2009).

Ejemplo 6. En el centro de control del satélite OrbitSat-1, se están planteando instalar una antena adicional para manejar mejor las solicitudes de conexión. Antes de tomar una decisión, desean conocer la media de solicitudes que suelen recibir y el tiempo medio que tardan en procesarlas. Para ello, han recogido datos sobre los intervalos entre llegadas de solicitudes y los tiempos de procesamiento. Las solicitudes llegan al satélite según una distribución de Poisson con una media de 16 solicitudes por hora. El tiempo medio que tarda en procesarse cada solicitud es de 2.5 minutos con una desviación de 5/4 minutos.

El modelo más adecuado para esta situación sería un M|G|1 que podemos ajustar a un M|E_k|1. Sabiendo que $\frac{1}{\mu} = 2{,}5$ y que la varianza de la distribución empírica de los tiempos de servicio es $\sigma^2 = 25/16$, podemos estimar el valor de k

$$\sigma^2 = \frac{25}{16} = \frac{1}{k\mu^2} = \frac{1}{k \cdot \left(\frac{2}{5}\right)^2} \text{ entonces } k = 4$$

Una vez sabemos que los tiempos de servicio siguen una distribución de Erlang con $k = 4$ y que el sistema no satura ($\rho = 2/3 < 1$), aplicamos las fórmulas de Pollaczek-Khintchine para obtener el numero medio de solicitudes que hay en la cola (3.22) y después las fórmulas de Little para calcular el resto de medidas de rendimiento

$$L_q = \frac{1+4}{2 \cdot 4} \cdot \frac{\left(\frac{16}{60}\right)^2}{\frac{2}{5} \cdot \left(\frac{2}{5} - \frac{16}{60}\right)} = \frac{5}{6} \text{ solicitudes de media en la cola}$$

$$W_q = \frac{1}{\lambda} \cdot L_q = \frac{60}{16} \cdot \frac{5}{6} = \frac{25}{8} \text{ minutos de media de espera}$$

$$W = W_q + \frac{1}{\mu} = \frac{25}{8} + \frac{5}{2} = \frac{45}{8} \text{ minutos de media en el sistema}$$

$$L = \lambda \cdot W = \frac{16}{60} \cdot \frac{45}{8} = \frac{3}{2} \text{ solicidudes de media en el sistema}$$

3.8. Sistema E_K|M|c

Consideremos ahora un proceso de colas de c servidores, cuyos tiempos de servicio se distribuyen de forma exponencial, al contrario que los tiempos entre llegadas, de los que no se especifica qué distribución sigue. Uno de los métodos más efectivos será considerar que dichos tiempos se distribuyen siguiendo una Erlang de parámetro de forma k y de tasa $R = k\lambda$.

De la misma manera que con el modelo anterior, los tiempos entre llegadas tienen una media de $\frac{1}{\lambda}$ y una varianza de $\frac{1}{k\lambda^2}$, por lo que si no se conoce el número de etapas que el cliente pasa hasta entrar en el sistema, k, se pueden estimar dichos descriptivos de la distribución empírica (G) y determinar un valor de k que nos permita ajustar una Erlang a nuestro patrón de llegadas.

De nuevo, pongamos el ejemplo de un modelo E_k|M|1, que al igual que se comentó en el modelo M|E_k|1, no deja de ser un caso específico de G|M|1, por lo que el desarrollo matemático para obtener las medidas de rendimiento será muy similar.

En primer lugar, denotaremos por $p_n^{(P)}$ al número de fases de llegada en estado estacionario, lo que incluye las k fases por las que cada cliente ha pasado para "entrar oficialmente" en el sistema y las fases completas de la siguiente llegada, que todavía "está en camino". De esta forma, las probabilidades de que haya n clientes en el sistema cuando se encuentra en estado estacionario vienen dadas por la relación

$$p_n = \sum_{i=nk}^{nk+k-1} p_i^{(P)} \tag{3.25}$$

El desarrollo matemático completo de donde se derivan los siguientes resultados se encuentra en el tercer capítulo del libro "Fundamentals of Queueing Theory" (Gross et al., 2008), que deriva el valor de $p_j^{(P)}$ de la forma

$$p_j^{(P)} = \rho\,(1 - r_0) \cdot r_0^{j-k} \quad \left(\text{donde } j \geq k-1, \rho = \frac{\lambda}{\mu}\right) \tag{3.26}$$

Siento r_0 la única raíz de la ecuación característica obtenida de

$$\mu r^{k+1} - (k\lambda + \mu)r + k\lambda = 0 \tag{3.27}$$

Definiendo p_n (3.26) en función de los operadores ρ y r_0^k (3.27) para cuando el sistema no está vacío tenemos

$$p_n = \rho\left(1 - r_0^k\right)\left(r_0^k\right)^{n-1} \text{ con } n \geq 1 \tag{3.28}$$

Luego entonces

$$L = \rho\left(1 - r_0^k\right)\sum_{n=1}^{\infty} n\left(r_0^k\right)^{n-1} = \frac{\rho}{1 - r_0^k} \tag{3.29}$$

Una vez obtenemos el número medio de clientes en el sistema, al contrario que en el caso G|M|1, sí podemos aplicar las fórmulas de Little y despejar las medidas $L_q = L - \rho$, $W = L/\lambda$ y $W_q = W - 1/\mu$.

De la misma forma que en los sistemas M|E_k|c, cuando contamos con más de un servidor, no se

han llegado a unos resultados generales aplicables a todos los casos, por lo que es necesaria la aplicación de una teoría avanzada para obtener los resultados de cada caso particular. Algunos de estos resultados se encuentran tabulados en el libro "Introduction to Operations Research" (Hillier & Lieberman, 2009).

Ejemplo 7. El puesto fronterizo entre dos países se separa en dos puestos de inspección aduanera. El tiempo que tardan en revisar la documentación y equipaje de un vehículo sigue una distribución exponencial con una media de 3 minutos. Por otro lado, los tiempos entre las llegadas a la frontera se distribuyen según una Normal de media 5 minutos y desviación estándar de $\frac{5\sqrt{2}}{2}$ minutos.

El modelo más adecuado para esta situación sería un G|M|2, que podemos ajustar a un E_k|M|2. Sabiendo que $\frac{1}{\lambda} = 5$ y que la varianza de la distribución empírica de los tiempos entre llegadas es $\sigma^2 = \frac{25}{2}$, podemos estimar el valor del parámetro de forma k:

$$\sigma^2 = \frac{25}{2} = \frac{1}{k\lambda^2} = \frac{1}{k \cdot \left(\frac{1}{5}\right)^2} \text{ entonces k = 2.}$$

Al no existir fórmulas cerradas para las medidas de rendimiento de un sistema E_K|M|c, como es este caso, este escenario complejo requeriría el uso de resultados tabulados obtenidos mediante métodos numéricos precisos. Es por ello, que más adelante se abordará la simulación de estos sistemas para ilustrar su comportamiento y análisis de manera detallada. Mediante la aplicación web diseñada en este trabajo, este tipo de problemas serán fácilmente resueltos.

Capítulo 4

Simulación

4.1. Modelo de simulación para un sistema de colas

La complejidad de muchos de los sistemas del mundo real conlleva la creación de modelos analíticos que resultan inasequibles. Por eso, en muchos casos es frecuente recurrir a la simulación para su estudio. Este método existe desde antes de la aparición de los ordenadores, pero el avance tecnológico ha hecho que tanto la herramienta como la técnica hayan evolucionado de la mano.

Shannon (1975) define el concepto de simulación como "el proceso de diseñar un modelo de un sistema real y realizar experimentos con este modelo con el propósito ya sea de entender el comportamiento del sistema o de evaluar diversas estrategias (dentro de los límites impuestos por un criterio o un conjunto de criterios) para el funcionamiento del sistema".

Sin embargo, antes de poner en práctica esta técnica, es necesario conocer y entender algunos de los conceptos básicos.

- **Sistema**: Es una colección de entidades que actúan e interactúan hacia el logro de algún objetivo lógico (Schmidt & Taylor, 1970).
- **Estado**: En general, los sistemas no se mantienen iguales en el tiempo, por lo que es necesario definir su estado. El estado de un sistema se define como el conjunto de variables que definen las condiciones del mismo en un momento dado. En un proceso de colas, un ejemplo de variable de estado sería el número de servidores ocupados o el tamaño en la cola en el momento en el que entra un nuevo cliente.
- **Entidad**: cualquier elemento de interés en el sistema se denominará entidad y sus características o propiedades serán atributos. Siguiendo con el ejemplo de un sistema de colas, los clientes son las entidades.

En función de si las variables de estado del sistema cambian en momentos concretos (definidos en el tiempo), o si lo hace de manera continua, la simulación del modelo puede clasificarse como estática o dinámica respectivamente. A su vez, ambos casos pueden ser deterministas, si no se tienen en cuenta variables aleatorias, o estocásticos, si estas son necesarias para la representación de la variabilidad inherente al sistema.

En este caso, nos centraremos en la simulación estática, en concreto en la estocástica, que es la clasificación correcta para un sistema de colas, también conocida como la simulación de modelos basados en eventos discretos (DES). Un evento se define como un suceso instantáneo capaz de modificar el estado del sistema. Entre eventos, todas las variables de estado permanecen constantes. La mejor forma de explicar el algoritmo es a través un ejemplo.

Consideremos el caso de un sistema de un único servidor, en el que tanto los tiempos entre llegadas como los de servicio se distribuyen exponencialmente. La capacidad del sistema será infinita y seguirá una disciplina FIFO, por lo que se trata de un modelo M|M|1. Algunos ejemplos reales de este tipo de procesos pueden ser un supermercado con un único cajero, un taller con un sólo técnico o un satélite con una única antena para manejar solicitudes de conexión.

Antes de comenzar con el desarrollo del algoritmo, es necesario definir algunas características y concretar ciertas suposiciones para asegurar una comprensión clara del proceso. En primer lugar, asumiremos que las llegadas sólo pueden entrar de una en una, aleatoriamente y los tiempos entre ellas seguirán la distribución especificada en la primera tabla (Figura 4.1 (fuente: Winston (2003), al igual que el resto de figuras citadas en la sección)). Por otro lado, consideraremos que los tiempos de servicio, también aleatorios, seguirán la distribución representada en la segunda tabla (Figura 4.2). Tras ser atendidos, los clientes saldrán del sistema.

Interarrival Time (minutes)	Probability
1	.20
2	.30
3	.35
4	.15

Figura 4.1: Distribución de los tiempos entre llegadas

Service Time (minutes)	Probability
1	.35
2	.40
3	.25

Figura 4.2: Distribución de los tiempos de servicio

Otro aspecto importante que hay que especificar antes de empezar, son las variables que definen el estado del sistema. En este tipo de procesos contamos con tres: el número de clientes en el sistema, el estado del servidor (libre u ocupado) y el tiempo de la próxima llegada. La aparición de un nuevo evento también cambiará el estado del sistema. En todo proceso de colas con capacidad infinita existen dos posibles eventos: la llegada de un nuevo cliente, y su salida tras completar el servicio. Dichos eventos se programarán para que ocurran en un momento concreto y toda la información sobre ellos se registrará y almacenará en una lista de eventos.

Ahora bien, ¿cómo se trabaja con un contador de tiempo en una simulación? La respuesta es sencilla, definiremos una variable llamada “tiempo de reloj” que irá contabilizando el tiempo de los sucesos. Esta idea ahora parece algo abstracta, pero a medida que vayamos explicando el algoritmo se irá aclarando.

Comenzaremos la simulación con un sistema vacío y en tiempo cero, que es cuando tiene lugar nuestro primer evento, una llegada. Al entrar en el sistema, este primer cliente se encuentra con un servidor ocioso, por lo que se le atiende de forma inmediata. Las llegadas posteriores pueden encontrarse el servidor libre, luego son atendidas en el momento, u ocupado, que será cuando se unan a la cola.

El siguiente paso será programar el tiempo de salida del primer cliente, generándolo de forma aleatoria a partir de la distribución de los tiempos de servicio (Figura 4.2) y calculándolo de la siguiente manera

Hora de salida = tiempo de reloj actual + tiempo de servicio generado

De la misma forma, programaremos el tiempo de la siguiente llegada, generando un tiempo entre llegadas a partir de su distribución (Figura 4.1) y estableciendo el tiempo de esta llegada como

Hora de llegada = tiempo de reloj actual + tiempo entre llegadas generado

Imaginemos que hemos programado que el tiempo que tardan en completar el servicio del primer cliente es de 3 minutos. Dicho cliente saldrá del sistema en el tiempo de reloj 3. De igual manera, hemos generado un tiempo entre llegadas de 1, luego la segunda llegada ingresará en el sistema cuando el reloj marque 1. Como se comentó antes, todos estos datos se registrarán en la lista de eventos. Una vez hemos acabado de programar la primera llegada, recurriremos a dicha lista para ver cuándo y cuál será el siguiente evento.

En el caso de ser una llegada, nuestra variable "tiempo de reloj" se mueve hasta el tiempo de llegada que hemos programado y repetimos el proceso explicado previamente. Por el contrario, si es una salida, nuestro reloj se moverá hasta la hora de salida programada y gestionaremos dicha salida. En este caso, tenemos dos opciones, si hay clientes esperando y, por lo tanto, el servidor atiende al primero de la fila, se resta un cliente del tamaño de la cola y se determina su tiempo de salida; o si no hay línea de espera y el servidor queda libre, lo que implica que el sistema está innactivo. Este proceso se irá repitiendo hasta que se cumpla una condición de parada.

Esta idea de simulación se denomina mecanismo de avance de tiempo del evento siguiente, puesto que el reloj avanza al tiempo del evento más inminente (Winston, 2003). Considerando que las variables de estado sólo cambian cuando se da un evento, evitamos los períodos de inactividad entre ellos, avanzando directamente de un evento a otro y realizando las acciones correspondientes. Es importante enfatizar que, cuando se procesa una llegada, es necesario programar la siguiente. Sin embargo, sólo podemos generar un tiempo de salida cuando un cliente es atendido, por lo cual, que el sistema quede innactivo puede suponer un problema. Para poder solucionarlo, se programa una "salida ficticia" con un tiempo de salida muy alto, como por ejemplo 9999, que el tiempo de reloj no pueda superar.

Existen otros métodos, como el de avance de tiempo de incremento fijo, en el que se determinan los saltos de tiempo que va dando el reloj y, tras cada actualización, se observa si hay algún evento programado para ese momento. En el caso de que lo haya, se realizan las acciones pertinentes y se actualiza el reloj. En el caso de no darse ningún evento, se sigue avanzando directamente. Sin embargo, este método no es tan eficiente como el mecanismo de avance de tiempo del evento siguiente (Winston, 2003).

Para demostrar cómo avanzaría una simulación de este modelo M|M|1 en el tiempo, explicaremos paso a paso un ejemplo numérico. Dicho algoritmo está representado en el siguiente diagrama de flujo (Figura 4.5) en el que todos los bloques han sido numerados para facilitar su comprensión.

En primer lugar, asumiremos que los primeros tiempos entre llegadas sucesivas (IT) y de servicio (ST) se han generado siguiendo sus distribuciones correspondientes (Figuras 4.1 y 4.2) y se muestran en la tercera tabla (Figura 4.3). Antes de comenzar, es necesario definir las variables que intervienen en el algoritmo:

- **TM** = Tiempo de reloj
- **AT** = Tiempo programado de la próxima llegada
- **DT** = Tiempo programado de la próxima salida
- **SS** = Estado del servidor (0 = libre, 1 = ocupado)
- **WL** = Longitud de la cola
- **MX** = Duración de la simulación

Customer Number	Interarrival Time (*IT*)	Service Time (*ST*)
1	—	3
2	2	3
3	2	2
4	3	1
5	4	1
6	2	2
7	1	1
8	3	2
9	3	—

Figura 4.3: Tiempos entre llegadas y de servicio generados

La variable MX representará nuestra condición de parada, es decir, será el tiempo de reloj máximo que puede alcanzar nuestra simulación antes de detenerse.

El primer paso será inicializar nuestras variables (bloque 1). Como comentamos previamente, comenzamos en tiempo cero (AT=0), con nuestro sistema vacío (SS=0 y WL=0) y una salida ficticia (DT=9999). Es importante destacar que esta salida ficticia debe ser mayor que la variable MX. Como podemos ver en la lista de eventos (Figura 4.4), contamos con un primer evento, una llegada en tiempo cero y con salida fiticia en tiempo 9999. Dicha lista será una de las salidas que nos devolverá el ordenador tras ejecutar el algoritmo.

La primera acción de nuestra simulación será buscar en la lista de eventos cuál será el siguiente evento (bloque 2), una llegada o una salida. En el caso de que AT<DT, estaremos hablando de una llegada, pero si AT>DT, gestionaremos una salida. Básicamente, necesitamos conocer qué tipo de evento llega antes y por lo tanto, qué acciones debemos llevar a cabo. En este caso, tenemos que AT=0 que es menor que DT=9999, luego estamos hablando de una llegada. Estableceremos el tiempo de reloj como el tiempo en el que entra en el sistema nuestro primer cliente, TM=0 (bloque 3).

Al entrar en el sistema, este primer cliente se lo encuentra vacío, es decir, con el servidor libre ya que SS=0 (bloque 4), luego es atendido al momento. Una vez entra en el servicio, realizamos los cambios correspondientes en nuestras variables. El servidor está ocupado, luego SS=1 (bloque 6), generamos un tiempo de servicio ST (bloque 7) y establecemos su tiempo de salida DT=TM+ST (bloque 8). En la tabla de tiempos de servicio generados (Figura 4.3), vemos que ST para este primer cliente es de 3, luego DT=0+3=3. El último paso cuando se gestiona una llegada, es programar la siguiente, luego se genera un tiempo entre llegadas IT (bloque 9) y se determina AT como AT=TM+IT (bloque 10). Como podemos ver en la Figura 4.3, la siguiente llegada será dentro de 2 unidades de tiempo, luego AT= 0+2=2. El segundo cliente entrará en el tiempo de llegada 2 y toda la información de la primera llegada, queda recogida en la Figura 4.4.

Como último paso de todo evento, es necesario comprobar si se puede seguir con la simulación o por el contrario, el tiempo de reloj ha alcanzado el tiempo máximo (bloque 18) y por lo tanto, el proceso se detiene y se imprimen los resultados (bloque 19). Para esta explicación asumiremos que siempre se verifica TM<MX, puesto que MX se ha definido como un valor muy alto. Por lo tanto, volvemos al bloque 2 y determinamos qué tipo de evento será el siguiente.

End of Event	Type of Event	Customer Number	System Variables			Event List	
			TM	SS	WL	AT	DT
0	Initialization	—	0	0	0	0	9,999
1	Arrival	1	0	1	0	2	3
2	Arrival	2	2	1	1	4	3
3	Departure	1	3	1	0	4	6
4	Arrival	3	4	1	1	7	6
5	Departure	2	6	1	0	7	8
6	Arrival	4	7	1	1	11	8
7	Departure	3	8	1	0	11	9
8	Departure	4	9	0	0	11	9,999
9	Arrival	5	11	1	0	13	12
10	Departure	5	12	0	0	13	9,999
11	Arrival	6	13	1	0	14	15
12	Arrival	7	14	1	1	17	15
13	Departure	6	15	1	0	17	16
14	Departure	7	16	0	0	17	9,999
15	Arrival	8	17	1	0	20	19

Figura 4.4: Representación computacional de la simulación

En este punto, contamos con AT=2 y DT=3, luego AT<DT, lo que implica que el siguiente evento será otra llegada. Igualamos TM al tiempo AT en el que entra este nuevo cliente y comprobamos si el servidor está ocupado o no.

Como el servicio del primer cliente tiene una duración de 3, cuando esta segunda llegada entra en el sistema, se encuentra con un servidor ocupado, por lo que se unirá a la cola. Sumamos uno al tamaño de la cola, WL=WL+1, lo que hace una lista de espera de tamaño 1 (bloque 5). El próximo paso será generar el tiempo de llegada del siguiente cliente. Como podemos ver en la Figura 4.3, la tercera llegada entra al sistema 2 unidades de tiempo después de la segunda, es decir, AT=2+2=4. De nuevo, volvemos al bloque 2 y repetimos el proceso.

Fijándonos de nuevo en la simulación representada en la Figura 4.4, podemos ver que el tiempo programado para la tercera llegada es AT=4 y que el tiempo de la siguiente salida es DT=3, luego esta vez, el siguiente evento será nuestra primera salida.

El primer paso será situar el tiempo de reloj en 3 (bloque 11). Una salida implica que el servidor deja de estar ocupado, luego es necesario analizar el tamaño de la cola como se puede ver en el bloque 12. En caso de que no haya línea de espera, el sistema se quedaría inactivo hasta la próxima llegada, ya que SS=0 y WL=0, luego se determinará DT=9999 (bloque 14). Sin embargo, como se determinó en el evento anterior, contamos con WL=1, por lo que hay un cliente esperando a ser atendido. En este caso, se resta una unidad al tamaño de la cola (bloque 15) y dicho cliente entra en servicio, luego es necesario programar su IT (bloque 16). Se establece su tiempo de salida en el bloque 17 y se continúa. Como podemos ver en la Figura 4.3, el tiempo de servicio dura 3 unidades de tiempo, por lo que DT= 3+3=6. Una vez realizadas todas las acciones correspondientes a una salida, volvemos al bloque 2 y evaluamos el siguiente evento.

En la representación computacional de la simulación se recoge toda la información necesaria por si el lector desea seguir analizando cómo continúa la simulación. Una vez se ha alcanzado el tiempo máximo y se detiene la simulación, es posible estimar las medidas de rendimiento ampliando brevemente el diagrama para que calcule la media de los tiempos de espera, el número de clientes esperado en el sistema, etc.

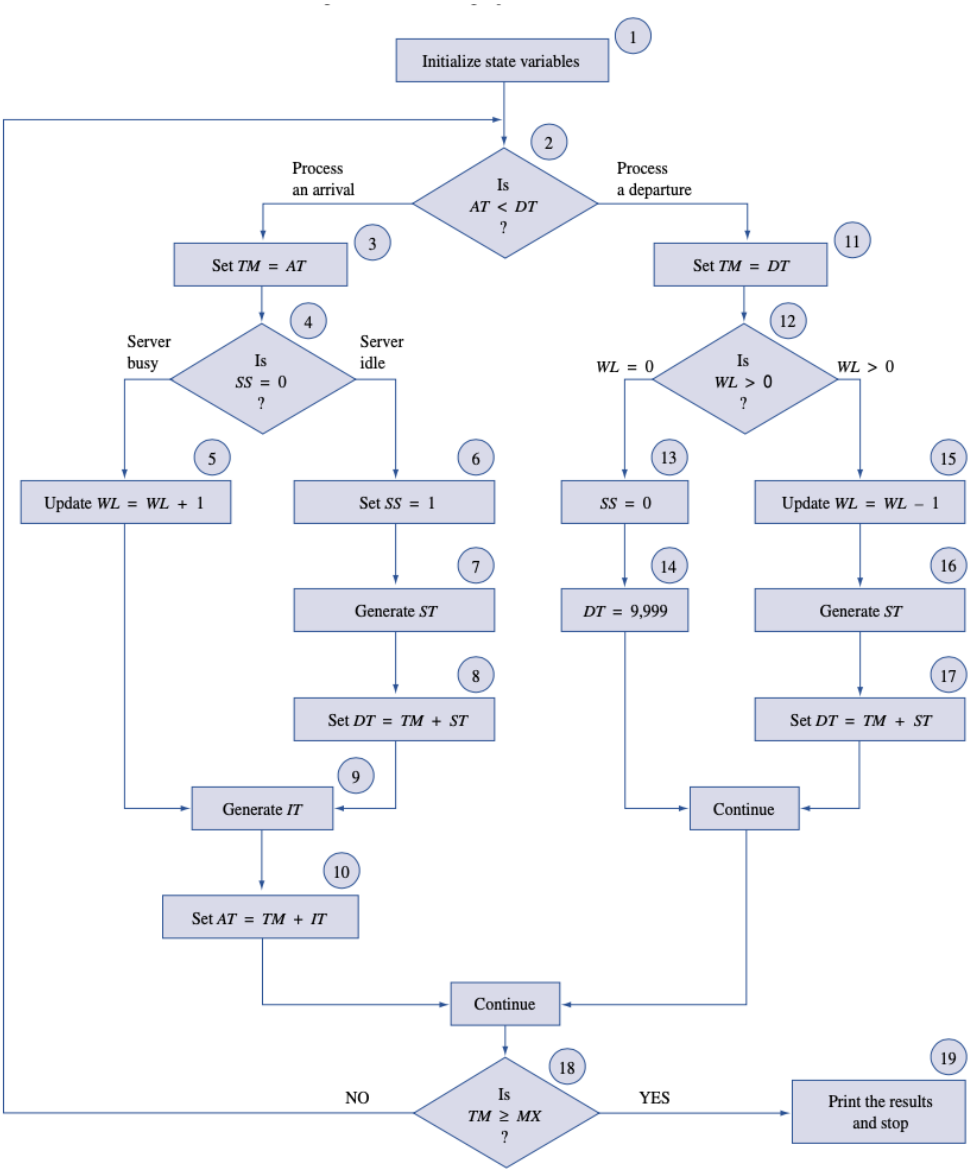

Figura 4.5: Diagrama de flujo para la simulación del modelo M|M|1

Aún así, los resultados obtenidos de una única simulación no son estadísticamente significativos, por lo que es necesario replicar múltiples veces dicha simulación. Hoad et al. (2010) definen el concepto de replicación como “ejecuciones múltiples de un mismo escenario con diferentes flujos de números aleatorios”. Las réplicas serán independientes entre sí, por lo que es posible calcular intervalos de confianza para las medidas de rendimiento del sistema simulado.

La primera pregunta que surge al trabajar con replicaciones es: ¿cuál es el número óptimo de réplicas que se deben hacer? En la literatura no hay un consenso al respecto, sin embargo, múltiples autores han propuesto distintos enfoques para intentar establecer un criterio objetivo:

Por un lado, Robinson (1994) ofrece un método gráfico basado en la representación de la media acumulada de la variable de salida tras realizar una serie de réplicas, frente al número de replicaciones. La idea es que el usuario elija visualmente el punto en el que dicha media se “aplana” para escoger el número de veces que será necesario repetir la simulación para obtener resultados correctos.

Por otro lado, está el método del intervalo de confianza (Law, 2024). El usuario especifica la precisión del intervalo de confianza para la estimación de la media de la variable de interés. Este método presenta dos ventajas frente al anterior: utiliza datos de salida del sistema simulado y ofrece una medida estadística de precisión, por lo que resulta más objetivo.

Por último, Banks et al. (2010) ofrecen un enfoque en el que se escoge el número de replicaciones basándose en un conjunto inicial de réplicas. Se estima la varianza de las variables de salida con dicho conjunto inicial y a partir de los resultados, con el objetivo de alcanzar la precisión absoluta establecida para la anchura media del intervalo de confianza, se estima el número de veces que hay que ejecutar la simulación.

Actualmente, existen múltiples paquetes para los distintos lenguages de programación que nos permiten simular un sistema de colas con las características que queramos. Con Python podemos recurrir a 'simpy' (Matloff, 2008), si preferimos trabajar con Java, tenemos la opción de 'JMT' (Bertoli et al., 2009) y, por supuesto, en el caso de R también contamos con herramientas robustas y eficientes para la simulación de un proceso de colas.

En este trabajo nos centraremos únicamente en la simulación de sistemas de colas G|G en el entorno de desarrollo integrado de RStudio, utilizando el lenguaje R. Desarrollaremos el algoritmo con los paquetes 'queuecomputer' (Ebert et al., 2020) y 'simmer' (Ucar et al., 2019) y estudiaremos sus distintas funciones valiéndonos de un ejemplo. Posteriormente, crearemos una aplicación web que permita a los usuarios estimar las medidas de rendimiento de un sistema con las características que ellos mismos hayan especificado.

4.2. Simulación de sistemas de colas G|G con R

4.2.1. Paquete queuecomputer

El paquete *queuecomputer* es una librería modular implantada en R que ofrece al usuario un método eficiente para simular un conjunto general de redes de colas. Además, integra perfectamente con el paquete *dplyr* si lo que se busca es generar un proceso de colas en paralelo.

De aquí en adelante, trabajaremos con los datos del ejemplo 6 resuelto en el apartado 3.7 de "Sistemas M|E_K|c" para explicar las distintas funciones que nos ofrece esta librería y sus correspondientes salidas. En dicho ejemplo, se definía un modelo de un único servidor con un proceso de llegadas de Poisson y tiempos de servicio no exponenciales, cuya distribución se aproximaba a una Erlang. Los datos y resultados obtenidos teóricamente fueron los siguientes:

Tabla 4.1: Resultados teóricos ejemplo práctico modelo M|E_4|1

c	1 satélite
λ	16/60 = 0.2666 solicitudes/min
μ	2/5 = 0.4 solicitudes/min
k	4
W	5.625 minutos de media en el sistema
W_q	3.125 minutos de media en la cola
L	1.5 solicitudes de media en el sistema
L_q	0.8333 solicitudes de media en la cola

Tras instalar y cargar las librerías *queuecomputer*, *dplyr* y *magrittr*, es necesario fijar una semilla si queremos que los resultados sean reproducibles. Al contrario que en el diagrama de flujo que explicamos al principio del capítulo, este paquete define como condición de parada un máximo de

llegadas al sistema, luego es necesario definir una variable 'n' que represente el número de clientes que se van a simular (n=50000 en nuestro caso). En este primer paso, también definiremos las variables 'lambda', 'mu', 'c' y 'k' con los valores correspondientes que se encuentran en la Tabla 4.1.

Diseñaremos la simulación con la función `queue_step`, cuyos parámetros son los tiempos de llegada y de servicio de los n clientes, así como el número de servidores del sistema. Para calcular los tiempos de reloj en los que llega cada cliente, es necesario generar un vector numérico con los tiempos entre llegadas de los n clientes, que seguirán una distribución Exponencial de parámetro λ y calcular su suma acumulativa.

```
R>t_entre_llegadas = rexp(n, lambda)
R>t_llegadas = cumsum(t_entre_llegadas)
```

En el caso de los tiempos de servicio, buscamos que sigan una Erlang. Como se comentó en los capítulos anteriores, la única diferencia entre esta distribución y la Gamma, es que el parámetro k es un número entero, luego podemos generar la variable 'tiempos de servicio' como una Gamma de parámetros k y kμ.

```
R>t_servicio = rgamma(n, shape=k, rate = k*mu)
```

Una vez generados los tiempos de llegada y de servicio, simulamos nuestro sistema

```
R>MEC = queue_step(arrivals = t_llegadas,service = t_servicio,
                   servers = c)
```

La lista MEC contiene seis objetos que recogen toda la información relevante sobre nuestra simulación:

- **departures:** vector con los tiempos de reloj en los que han salido los 50000 clientes.
- **server:** vector con el identificador del servidor que ha atendido a cada uno de los clientes. En nuestro ejemplo será un vector formado por n unos, puesto que sólo contamos con un único servidor.
- **departures_df:** data frame con los tiempos de llegada, de servicio, de salida, de espera y total en el sistema de cada cliente e identificador del servidor que les ha atendido.
- **queuelength_df:** data frame que nos indica el tamaño de la cola para cada instante de tiempo.
- **systemlength_df:** data frame que nos indica el tamaño del sistema para cada instante de tiempo.
- **servers_input:** número de servidores en el sistema, es decir, el valor de la variable 'c'.

Si visualizamos los primeros 15 clientes del data frame `departures_df`:

Tabla 4.2: Datos de la simulación para los primeros 15 clientes

arrivals	*service*	*departures*	*waiting*	*system_time*	*server*
1.72	2.45	4.17	0	2.45	1
9.39	1.69	11.1	0	1.69	1
10.8	3.21	14.3	2.69e- 1	3.48	1
13.4	3.22	17.5	8.95e- 1	4.11	1
20.7	2.60	23.3	0	2.60	1
22.5	2.45	25.8	7.95e- 1	3.25	1
24.9	1.54	27.3	8.62e- 1	2.40	1
40.3	1.89	42.2	0	1.89	1
46.8	2.71	49.5	0	2.71	1
48.8	2.07	51.6	7.23e- 1	2.79	1
53.8	5.30	59.1	0	5.30	1
62.6	2.79	65.4	0	2.79	1
65.1	1.93	67.3	2.54e- 1	2.18	1
70.8	3.76	74.5	0	3.76	1
74.5	1.78	76.3	1.46e- 2	1.80	1

Si nos fijamos en el primer cliente, vemos que llega al sistema en un tiempo de reloj de 1.72, el servidor completa el servicio en 2.45 unidades de tiempo y al tratarse del primer cliente, su tiempo de espera es de 0. Finalmente, sale del sistema en un tiempo de reloj de 4.17. En total, el tiempo que ha estado dicho cliente en el sistema es 2.45, la suma del tiempo de espera y de servicio. Si el lector continúa analizando la tabla, se dará cuenta de que los resultados obtenidos sería idénticos a los que se conseguirían al aplicar el algoritmo que se desarrolló anteriormente.

Aplicando la función `summary` sobre la lista MEC, también obtenemos información muy útil sobre la simulación, incluyendo las cuatro medidas de rendimiento.

```
Total customers:
 50000
Missed customers:
 0
Mean waiting time:
 3.19
Mean response time:
 5.69
Utilization factor:
 0.666785041031979
Mean queue length:
 0.853
Mean number of customers in system:
 1.52
```

Figura 4.6: Ejemplo de salida de la función `summary` aplicada a un modelo M|E_k|c.

Otro de los aspectos de la simulación que puede resultar interesante analizar, son los distintos tamaños que ha alcanzado la cola y la proporción de tiempo de dichos estados. Dentro de la lista resumen generada al pasarle la lista MEC a la función `summary`, encontramos un tibble llamado `slength_sum` que recoge dicha información (Tabla 4.3).

Tabla 4.3: Tamaños de la cola y su proporción en el tiempo

queuelength	*proportion*
0	0.333
1	0.282
2	0.175
3	0.0952
4	0.0519
5	0.0283
6	0.015
7	0.00802
8	0.00457
9	0.00286
10	0.00164
11	0.00115
12	0.000702
13	0.000314
14	0.000158
15	0.0000654
16	0.00000641

Para estimar las medidas de rendimiento, trabajaremos con los datos del data frame `departures_df`.

En primer lugar, calculamos la media de la columna `system_time`, que nos dará una aproximación del tiempo medio en el sistema para cada cliente (W). De la misma forma, la media de la columna `waiting` será la estimación para el tiempo medio en la cola para cada cliente (W_q). Una vez obtenidas estas medidas, podemos calcular L y L_q aplicando las fórmulas de Little (1.3 y 1.4). Los resultados obtenidos para nuestro ejemplo son los siguientes:

Tabla 4.4: Medidas de rendimiento estimadas mediante simulación

Parámetro	*Resultado teórico*	*Resultado simulación*	*Diferencia*
W	5.625	5.685926	0.06092570
W_q	3.125	3.191535	0.06653469
L	1.5	1.516247	0.01624685
L_q	0.8333	0.8510759	0.01777592

Si comparamos los resultados que obtuvimos teóricamente con las estimaciones de la simulación, podemos ver que los errores son relativamente pequeños. Sin embargo, estos resultados se basan en una única simulación. Como se mencionó anteriormente, al replicar la simulación varias veces y calcular cada medida de rendimiento como el promedio de las estimaciones obtenidas, podemos mejorar la precisión de los resultados.

A la hora de replicar una simulación en R, contamos con dos métodos: realizar un bucle for o aplicar la función `lapply`.

Si repetimos la simulación 1000 veces mediante un bucle, los resultados obtenidos son los siguientes:

Tabla 4.5: Medidas de rendimiento estimadas realizando 1000 replicaciones con un bucle

Parámetro	*Resultado teórico*	*Media*	*Varianza*	*Diferencia*
W	5.625	5.621345	0.006217	0.003655
W_q	3.125	3.121433	0.005860	0.003567
L	1.5	1.499025	0.000442	0.000975
L_q	0.8333	0.8323823	0.000417	0.000918

La ejecución del bucle tardó 36 segundos en completarse. Al contrastar las diferencias de la Tabla 4.4 con estos resultados, se observa que la precisión de las estimaciones mejora notablemente, pues los errores son claramente menores.

Por otro lado, tenemos la opción de replicar la simulación mediante `lapply`, que aplica una función a cada elemento de una secuencia definida en su primer parámetro. En este caso, utilizaremos una función anónima que ejecuta la simulación hasta un número máximo de clientes, definido por la variable 'n'.

```
R>simulacion_replicada <-lapply(1:replicaciones, function(i) {
      t_entre_llegadas = rexp(n, lambda)
      t_llegadas = cumsum(t_entre_llegadas)
      t_servicio = rgamma(n, shape=k, rate = k*mu)
      EMCrep <- queue_step(arrivals = t_llegadas,
                           service = t_servicio, servers = c)
      W = mean(EMCrep$departures_df$system_time)
      Wq = mean(EMCrep$departures_df$waiting)
      L = lambda * W
      Lq = lambda * Wq
      resultados_replicaciones <-c(W = W, Wq = Wq, L = L, Lq = Lq)})
```

La función `lapply` nos devolverá una lista con las estimaciones de cada una de las simulaciones realizadas. Para obtener una aproximación de cada una de las medidas, solo será necesario calcular los promedios de cada columna. Los resultados obtenidos han sido los siguientes:

Tabla 4.6: Medidas de rendimiento estimadas realizando 1000 replicaciones con lapply

Parámetro	*Resultado teórico*	*Media*	*Varianza*	*Diferencia*
W	5.625	5.6222182	0.0059718	0.0027818
W_q	3.125	3.1224287	0.0056256	0.0025713
L	1.5	1.4992582	0.0004247	0.0007414
L_q	0.8333	0.8326476	0.000400	0.0006523

La ejecución de las mil replicaciones ha tardado 53 segundos. Al igual que con el bucle, si comparamos las diferencias con el valor teórico de la Tabla 4.4 y de estos resultados, observamos que los errores de las estimaciones disminuyen.

Por último, el paquete *queuecomputer* también ofrece un método para graficar los resultados mediante la función `plot`, que requiere un argumento del tipo `queue_list`, como es el caso de la

lista MEC. Esta función nos devolverá seis gráficos distintos. Para facilitar su visualización en los siguientes ejemplos, sólo se graficará la simulación de los primeros 20 clientes que llegan al sistema.

En primer lugar, podemos ver un gráfico de densidad de los tiempos de llegada y salida, donde se observa la distribución de probabilidad de ambos eventos a lo largo del tiempo (Figura 4.7). En el eje X se representa el tiempo, mientras que en el eje Y se muestra la densidad de probabilidad. Este gráfico corresponde al parámetro `which = 1` de la función `plot`.

En segundo lugar, obtenemos dos histogramas lado a lado (Figura 4.8), uno para los tiempos de llegada y otro para los tiempos de salida. El eje horizontal representa el tiempo y el eje vertical representa la cantidad de llegadas o salidas que ocurrieron en esos intervalos de tiempo. Este gráfico corresponde al parámetro `which = 2` de la función `plot`.

La tercera salida (Figura 4.9) es un gráfico de densidad que muestra las distribuciones de los tiempos de espera y los tiempos totales en el sistema. En el eje X se representa el tiempo, mientras que el eje Y se observa la densidad de probabilidad. Este gráfico corresponde al parámetro `which = 3` de la función `plot`.

En el gráfico correspondiente al parámetro `which = 4` de la función `plot`, se puede observar el número de clientes en la cola y en el sistema a lo largo del tiempo (Figura 4.10). El eje de abscisas representa el tiempo, mientras que el de ordenadas representa el número de clientes.

La Figura 4.11 presenta un gráfico de rango de líneas que representa el estado de los clientes y servidores a lo largo del tiempo. Los estados se dividen en “servicio” (rojo) y “espera” (azul). Este gráfico corresponde al parámetro `which = 5` de la función `plot`.

Por último, si el parámetro de la función `plot` es `which = 6`, obtenemos un gráfico de la distribución empírica que muestra la función de distribución acumulativa de los tiempos de llegada y salida. Como podemos ver en la Figura 4.12, el eje X representa el tiempo y el eje Y, la proporción de observaciones que son menores o iguales a un valor específico en el eje X.

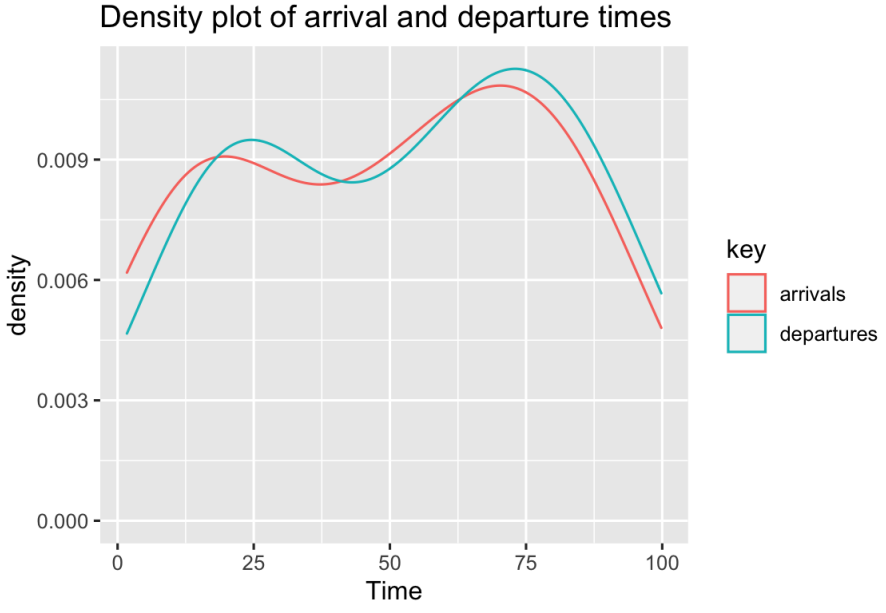

Figura 4.7: Gráfico de densidad de los tiempos de llegada y de salida

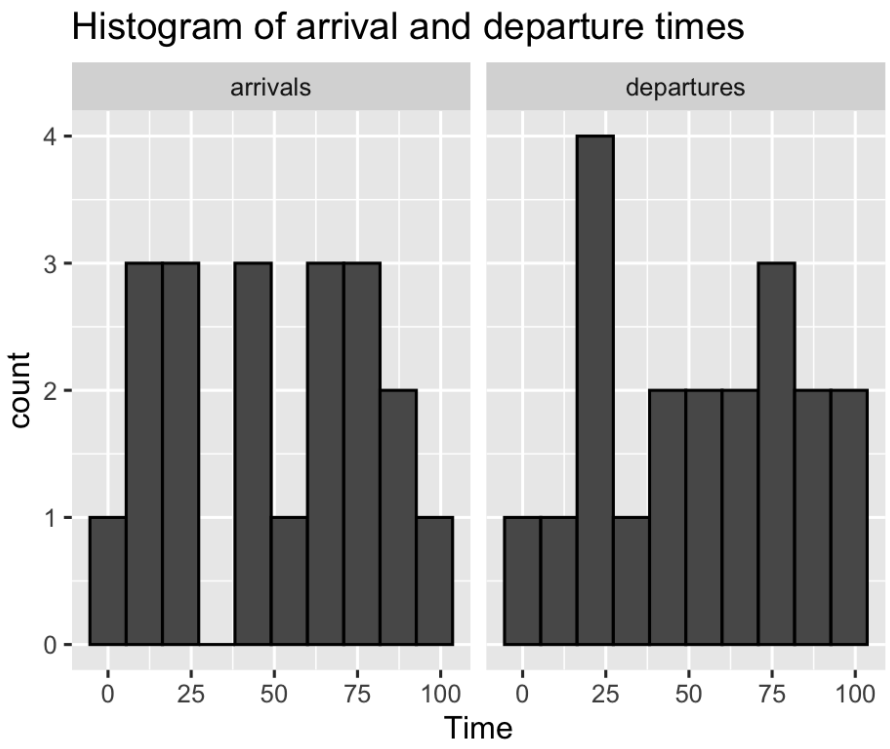

Figura 4.8: Histograma de los tiempos de las llegadas y las salidas

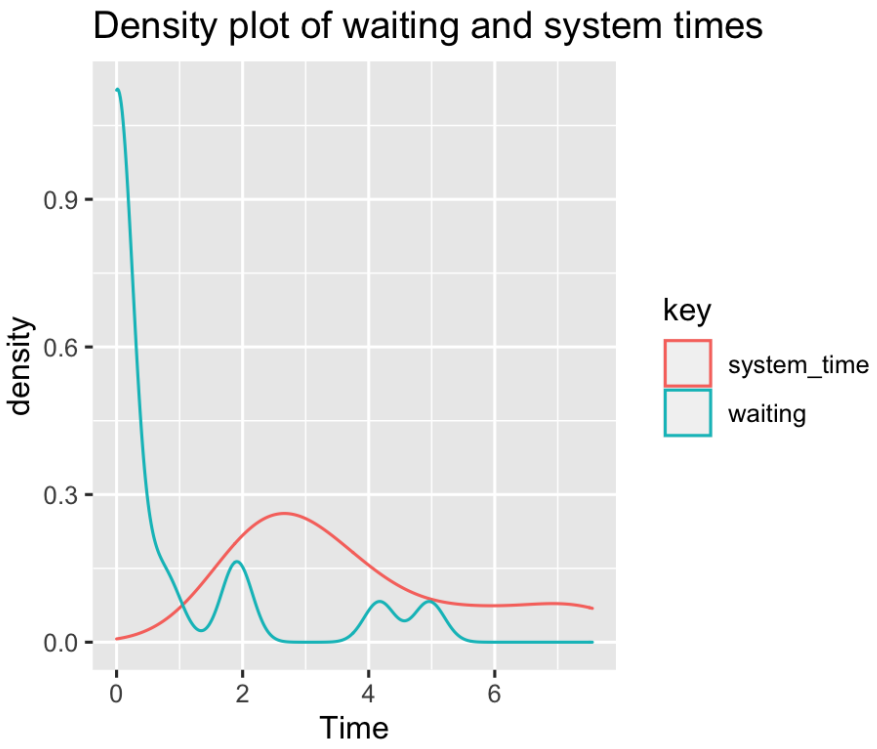

Figura 4.9: Gráfico de densidad de los tiempos de espera y totales en el sistema

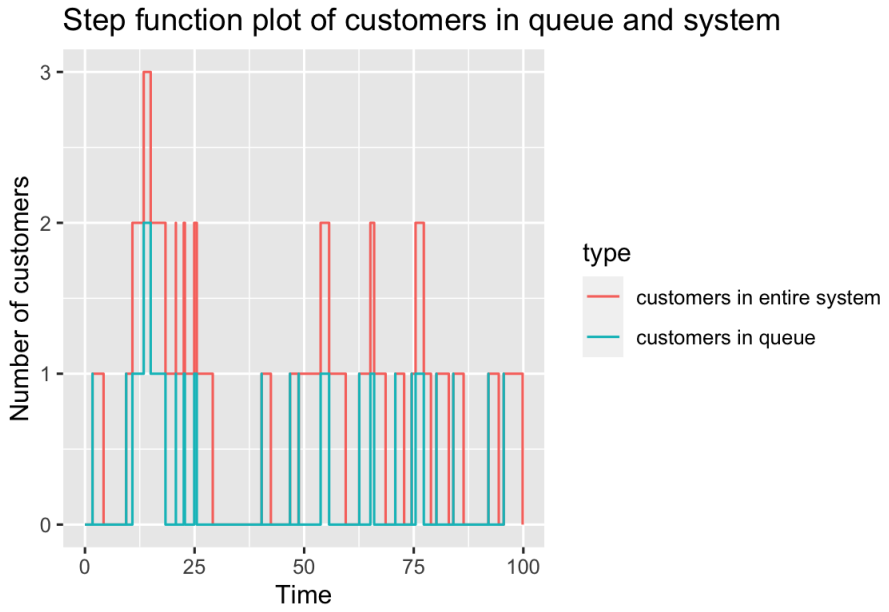

Figura 4.10: Gráfico de función escalonada de los clientes en la cola y en el sistema

Figura 4.11: Gráfico de rango de líneas del estado de los clientes y el sistema

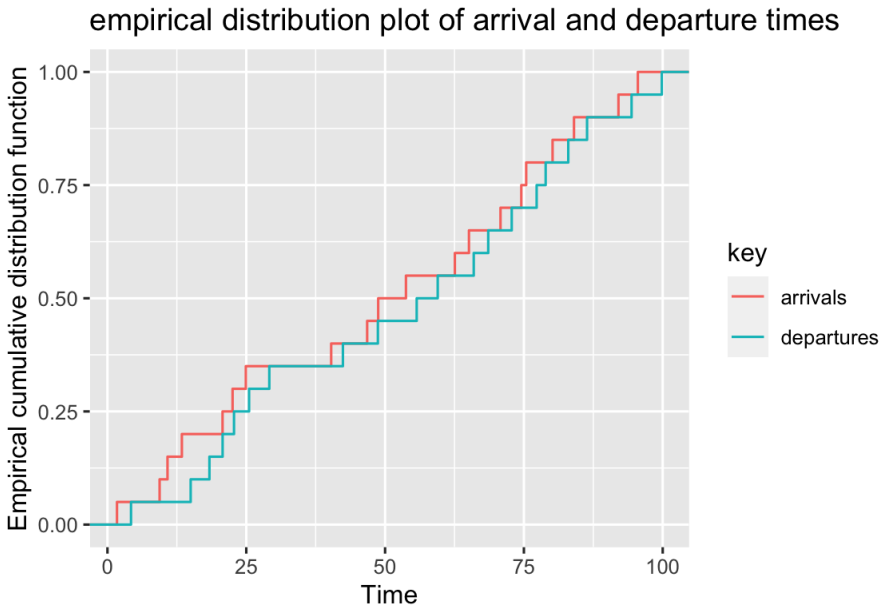

Figura 4.12: Gráfico de la distribución empírica de los tiempos de llegada y salida

La librería *queuecomputer* también es útil si se quiere simular un sistema E_K|M|c, en el que son los tiempos entre llegadas los que siguen una Erlang, miestras que los de servicio se distribuyen exponencialmente. El único cambio necesario será cambiar las distribuciones con las que generamos estas dos variables, de forma que

```
R>t_entre_llegadas = rgamma(n, shape = k, rate = k*lambda)
R>t_llegadas = cumsum(t_entre_llegadas)
R>t_servicio = rexp(n, mu)
```

El proceso para estimar las medidas de rendimiento será el mismo. Supongamos el ejemplo 7, planteado en el apartado 3.8 “Sistemas E_K|M|c”, cuyos datos son:

Tabla 4.7: Datos del ejemplo práctico de un modelo E_3|M|2

c	2 puestos
λ	1/5 = 0.2 vehículos/min
μ	1/3 = 0.3333 vahículos/min
k	3

Realizando una única simulación, los resultados de las estimaciones de las cuatro métricas son las siguientes:

Tabla 4.8: Medidas de rendimiento de un modelo E_3|M|2 estimadas mediante simulación

W	W_q	L	L_q
0.8177752	0.4818741	4.088876	2.40937

4.2.2. Paquete simmer

El paquete *simmer* es una librería para la simulación de eventos discretos (DES) que permite al usuario realizar modelos basados en procesos. Además, trabaja con el novedoso concepto de "trayectoria" que, según la definen los autores Ucar et al. (2019), consiste en una "lista de acciones estandarizadas que define el ciclo de vida de procesos equivalentes".

Al igual que en el capítulo anterior, nos apoyaremos en el ejemplo 6 resuelto en el apartado 3.7 "Sistemas M|E_K|c" para explicar las distintas funciones que nos ofrece esta librería y sus correspondientes salidas. Los datos del sistema y los resultados teóricos se encuentran en la Tabla 4.1. En este caso, también será necesario fijar una semilla para que los resultados sean reproducibles, y definir las variables 'lambda', 'mu', 'c' y 'k' con los valores correspondientes que se encuentran en la Tabla 4.1.

En primer lugar, es necesario definir la trayectoria que seguirá un cliente al entrar en el sistema con la función `trajectory`. Un cliente llega al sistema y es atendido por un servidor, tomando 1 unidad del recurso llamado 'servidor'. Su tiempo de servicio está determinado por una distribución de Erlang de parámetros k y kμ y, una vez concluido, el cliente libera el servidor.

```
R>trajectory("Recorrido del cliente")%>%
      seize("servidor", amount = 1)%>%
      timeout(function() rgamma(1, shape=k, rate = k*mu))%>%
      release("servidor", amount = 1) ->trayectoria
```

El siguiente paso es crear el entorno de la simulación con la función `simmer`. Añadimos un recurso llamado 'servidor', cuya capacidad está determinada por el número de servidores de los que dispone el sistema. Los clientes entrarán en el sistema a través un generador de entidades llamado 'llegadas' siguiendo una distribución Exponencial de parámetro λ y seguirán la trayectoria definida anteriormente.

```
R>simmer("Sistema M|E|c")%>%
      add_resource("servidor", capacity = c)%>%
      add_generator("llegadas", trayectoria,
                    function() rexp(1, lambda)) ->sistema
```

Al contrario que *queuecomputer*, este paquete define como condición de parada un tiempo de reloj máximo, por lo que es necesario definir la variable 'tmax'. Para este ejemplo, ejecutaremos la simulación hasta un tiempo máximo de tmax=8000. Una vez determinada la trayectoria y el entorno, ejecutamos la simulación.

```
R>sistema%>%
      run(until = tmax) ->sim.8000
```

La función `run` devuelve un objeto de tipo simmer, al que hemos llamado 'sim.8000', que contiene toda la información sobre nuestra simulación.

Como se muestra en la Tabla 4.9, al pasar este objeto a la función `get_mon_arrivals`, obtenemos una lista, a la que llamaremos 'llegadas.8000', con los siguientes datos de cada cliente:

- **`name:`** identificador del cliente.
- **`start_time:`** tiempo de reloj en el que entra en el sistema cada cliente.

- **end_time:** tiempo de reloj en el que sale del sistema cada cliente.
- **activity_time:** tiempo de servicio de cada cliente.
- **finished:** valor lógico que indica si el cliente pudo o no entrar en el sistema. En nuestro caso, la capacidad de la cola es infinita, luego todos podrán entrar.
- **replication:** identificador de la replicación si repetimos la simulación más de una vez.

Si visualizamos los primeros 15 clientes simulados de esta lista:

Tabla 4.9: Datos de los primeros 15 clientes simulados

name	*start_time*	*end_time*	*activity_time*	*finished*	*replication*
llegadas0	1.716167	4.519931	2.803764	TRUE	1
llegadas1	9.394150	12.401509	3.007359	TRUE	1
llegadas2	16.719503	17.925833	1.206330	TRUE	1
llegadas3	32.090757	35.583708	3.492951	TRUE	1
llegadas4	32.297097	41.027587	5.443879	TRUE	1
llegadas5	36.042049	42.742083	1.714496	TRUE	1
llegadas6	36.395890	44.484077	1.741994	TRUE	1
llegadas7	40.322633	46.467019	1.982942	TRUE	1
llegadas8	45.074893	51.152180	4.685161	TRUE	1
llegadas9	53.867007	57.679282	3.812275	TRUE	1
llegadas10	55.281382	60.053184	2.373902	TRUE	1
llegadas11	57.938269	62.767795	2.714611	TRUE	1
llegadas12	61.803245	68.335948	5.568153	TRUE	1
llegadas13	61.998286	69.771908	1.435961	TRUE	1
llegadas14	62.880551	74.121232	4.349324	TRUE	1

Para estudiar la evolución del estado del servidor y de la cola a lo largo del tiempo, utilizamos la función `get_mon_resources` con el objeto sim.8000. Esta función proporciona una lista, a la que llamaremos 'recursos.8000', como la que podemos observar en la Tabla 4.10 que contiene los siguientes elementos:

- **resource:** nombre del recurso. En nuestro caso, hemos añadido un único recurso llamado "servidor".
- **time:** tiempo de reloj en el que se da un cambio en la ocupacion de un servidor o de la cola.
- **server:** número de clientes en el servidor.
- **queue:** número de clientes en la cola.
- **capacity:** número de recursos en paralelo. En nuestro caso, será el número de servidores de nuestro sistema, es decir, nuestra variable 'c'.
- **queue_size:** capacidad de la cola. Esta característica se define al añadir el recurso a nuestro entorno de simulación mediante el parámetro `queue_size` de la función `add_resource`. Si se omite este parámetro, su valor por defecto será infinito.
- **system:** número de clientes dentro del sistema.
- **limit:** capacidad del sistema. Será la suma del número de recursos en paralelo y la capacidad de la cola. En nuestro caso, será infinito para todos los tiempos de reloj.
- **replication:** identificador de la replicación si repetimos la simulación más de una vez.

Si visualizamos las 15 primeras filas de la lista resultante:

Tabla 4.10: Ocupación del servidor y de la cola a lo largo del tiempo

resource	*time*	*server*	*queue*	*capacity*	*queue_size*	*system*	*limit*	*replication*
servidor	1.716167	1	0	1	Inf	1	Inf	1
servidor	4.519931	0	0	1	Inf	0	Inf	1
servidor	9.394150	1	0	1	Inf	1	Inf	1
servidor	12.401509	0	0	1	Inf	0	Inf	1
servidor	16.719503	1	0	1	Inf	1	Inf	1
servidor	17.925833	0	0	1	Inf	0	Inf	1
servidor	32.090757	1	0	1	Inf	1	Inf	1
servidor	32.297097	1	1	1	Inf	2	Inf	1
servidor	35.583708	1	0	1	Inf	1	Inf	1
servidor	36.042049	1	1	1	Inf	2	Inf	1
servidor	36.395890	1	2	1	Inf	3	Inf	1
servidor	40.322633	1	3	1	Inf	4	Inf	1
servidor	41.027587	1	2	1	Inf	3	Inf	1
servidor	42.742083	1	1	1	Inf	2	Inf	1
servidor	44.484077	1	0	1	Inf	1	Inf	1

Para estimar las medidas de rendimiento, trabajamos con los datos de los clientes de la lista llegadas.8000. En primer lugar, calculamos el tiempo medio que pasan los clientes en el sistema (W) y en la cola (W_q). Después, con las fórmulas de Little (1.3 y 1.4), calculamos L y L_q.

```
R>llegadas.8000%>%
      with(mean(end_time - start_time)) ->W
R>llegadas.8000%>%
      with(mean(end_time - start_time - activity_time)) ->Wq
R>Lq = lambda*Wq
R>L = lambda*W
```

Los resultados obtenidos para nuestro ejemplo son los siguientes:

Tabla 4.11: Medidas de rendimiento estimadas mediante simulación

Parámetro	*Resultado teórico*	*Resultado simulación*	*Diferencia*
W	5.625	5.671455	0.046455494
W_q	3.125	3.145076	0.020075962
cx	1.5	1.512388	0.012388132
L_q	0.8333	0.8386869	0.005386923

Al comparar los resultados obtenidos mediante simulación con los teóricos, podemos ver que los errores son relativamente pequeños. Aún así, podemos obtener unas mejores predicciones replicando la simulación.

Cuando trabajamos con el paquete *simmer*, al igual que con *queuecomputer*, contamos con dos métodos de replicación distintos: realizar un bucle for o aplicar la función `lapply`.

Si repetimos la simulación 1000 veces con un bucle, los resultados obtenidos son los siguientes:

Tabla 4.12: Medidas de rendimiento estimadas realizando 1000 replicaciones con un bucle

Parámetro	*Resultado teórico*	*Media*	*Varianza*	*Diferencia*
W	5.625	5.606496	0.1403364	0.018503959
W_q	3.125	3.106704	0.1326759	0.018296244
L	1.5	1.495066	0.009979477	0.004934389
L_q	0.8333	0.8284543	0.00943473	0.004845665

La ejecución del bucle tardó 45 segundos en completarse. Si contrastamos las diferencias de la Tabla 4.11 con los resultados obtenidos tras repetir la simulación 1000 veces, se observa que la precisión de las estimaciones mejora, pues los errores son menores.

Con el segundo método de replicación, repetimos mil veces una función anónima que ejecuta la simulación hasta un tiempo máximo, definido por la variable 'tmax' y la reinicia con `reset()`.

```
R>lapply(1:1000, function(i) {
    sistema%>%
       reset()%>%
       run(until = tmax) ->simulacion_replicada
```

La simulación de las 1000 replicaciones tardó en ejecutarse aproximadamente 34 segundos. Utilizando de nuevo la función `lapply`, estimaremos las medidas de rendimiento de cada replicación. Específicamente, para cada una de ellas, aplicamos la función `get_mon_arrivals` para obtener los tiempos de llegada y salida de los clientes, datos con los que se calculan las medidas de rendimiento de la misma forma que hicimos anteriormente. Las estimaciones finales serán las medias de dichas medidas obtenidas en las 1000 simulaciones. Los resultados obtenidos han sido los siguientes:

Tabla 4.13: Medidas de rendimiento estimadas realizando 1000 replicaciones con lapply

Parámetro	*Resultado teórico*	*Media*	*Varianza*	*Diferencia*
W	5.625	5.6093309	0.140259657	0.015669084
W_q	3.125	3.1101852	0.132530933	0.014814819
L	1.5	1.4958216	0.009974020	0.004178422
L_q	0.8333	0.8293827	0.009424422	0.003917285

De nuevo, si comparamos las diferencias con el valor teórico de la Tabla 4.11 y de los resultados obtenidos tras replicar 1000 veces la simulación, observamos que la precisión mejora notablemente ya que los errores son claramente menores.

Por otro lado, podríamos replicar la simulación descargando el paquete *parallel* (R Core Team, 2024) y utilizando la función `mclapply`, que aplica una función a cada elemento de una secuencia definida en su primer parámetro. A diferencia del enfoque anterior, esta técnica nos permitiría ejecutar varias réplicas de la simulación en paralelo, es decir, que múltiples instancias del proceso de simulación se realizaran simultáneamente en lugar de una tras otra, lo que reduciría significativamente el tiempo de ejecución.

Por último, el paquete *simmer.plot* (Ucar & Smeets, 2023) ofrece un método gráfico sencillo para representar la simulación de modelos basados en eventos discretos generados con *simmer*. La función que nos permite obtener estas salidas es

```
plot(x, metric, names, items)
```

En la Tabla 4.14 aparecen las distintas combinaciones de los parámetros de esta función y la referencia de la figura resultante. En total obtenemos siete gráficos distintos que representan distintas medidas de nuestro sistema de colas. Para estos ejemplos, trabajaremos con las listas 'llegadas.8000' y 'recursos.8000' definidas anteriormente, que obtienen información sobre los clientes, el sistema y la cola de la simulación sin replicaciones del modelo M|E_4|1.

Tabla 4.14: Combinaciones de los parámetros de la función plot

x	***metric***	***names***	***items***	***Figura***
llegadas.8000	flow_time	-	-	Figura 4.13
llegadas.8000	waiting_time	-	-	Figura 4.14
llegadas.8000	activity_time	-	-	Figura 4.15
recursos.8000	usage	servidor	system	Figura 4.16
recursos.8000	usage	servidor	queue	Figura 4.17
recursos.8000	usage	servidor	server	Figura 4.18
recursos.8000	utilization	servidor	-	Figura 4.19

Para los siguientes seis gráficos, el tiempo de simulación se representa en el eje X. Además se ha añadido una línea discontinua que representa la métrica calculada teóricamente.

En la Figura 4.13, podemos ver el tiempo que pasa en el sistema un cliente que entra en un instante concreto. La línea discontinua se sitúa en el punto flow time = 5.625, que corresponde al tiempo medio que un cliente pasa en el sistema (W) calculado teóricamente.

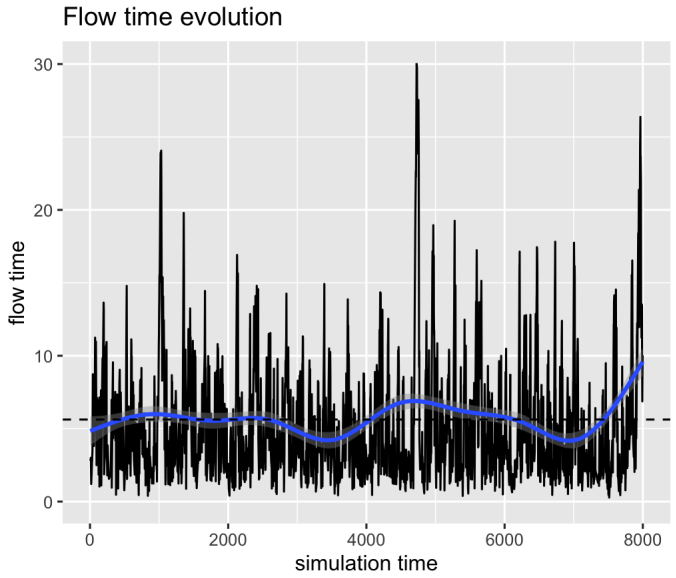

Figura 4.13: Tiempo que cada cliente pasa en el sistema.

El siguiente gráfico (Figura 4.14) es similar al anterior, pero representa el tiempo en la cola que pasa cada cliente. Las medias móviles, calculadas para cada cliente y representadas por la línea azul, están divididas prácticamente a la mitad por la línea discontinua situada a la altura de W_q = 3,125, calculado teóricamente.

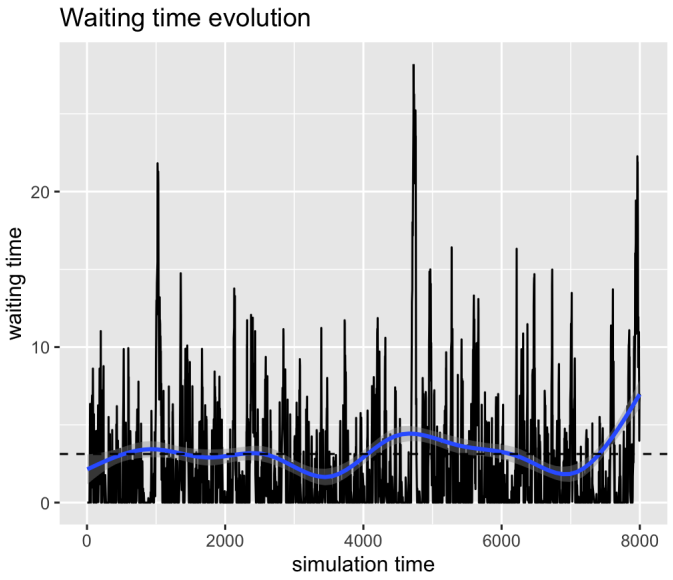

Figura 4.14: Tiempo que cada cliente pasa en la cola.

Con este tercer gráfico (Figura 4.15) podemos analizar el tiempo en el que el recurso está ocupado, es decir, el tiempo del servicio de cada cliente que entra en el sistema y es atendido por el servidor.

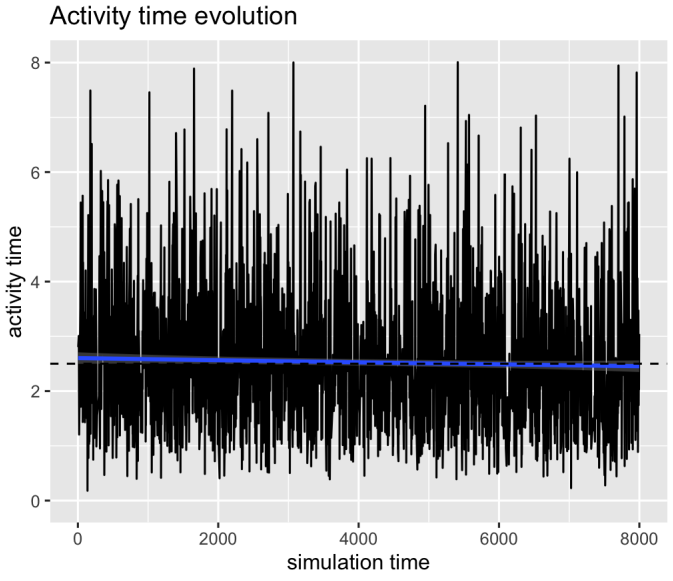

Figura 4.15: Tiempo de servicio de cada cliente.

En la Figura 4.16, podemos observar cómo el tamaño del sistema, representado por la línea roja, se acerca gradualmente a medida que pasa el tiempo a un estado estacionario con $L = 1{,}5$.

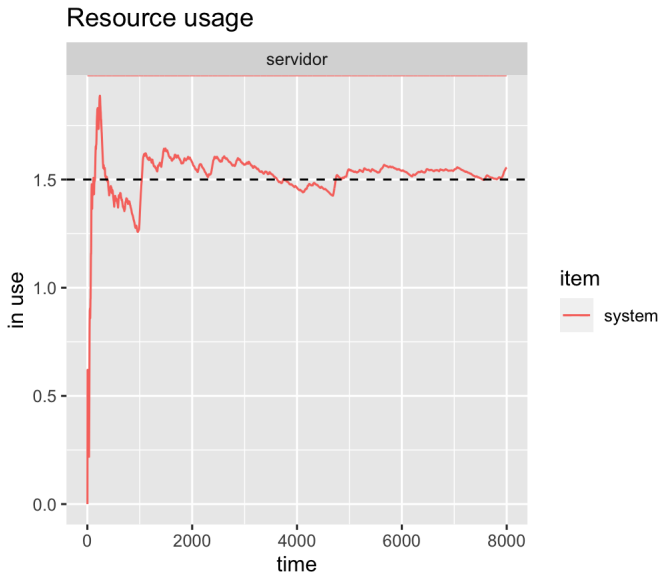

Figura 4.16: Convergencia del número de clientes en el sistema.

Este segundo gráfico correspondiente a los recursos (Figura 4.17) es muy similar que el anterior, pero en este caso se representa el tamaño de la cola. De igual forma, observamos cómo converge hacia un estado estacionario con un tamaño de cola de $L_q = 0{,}8333$.

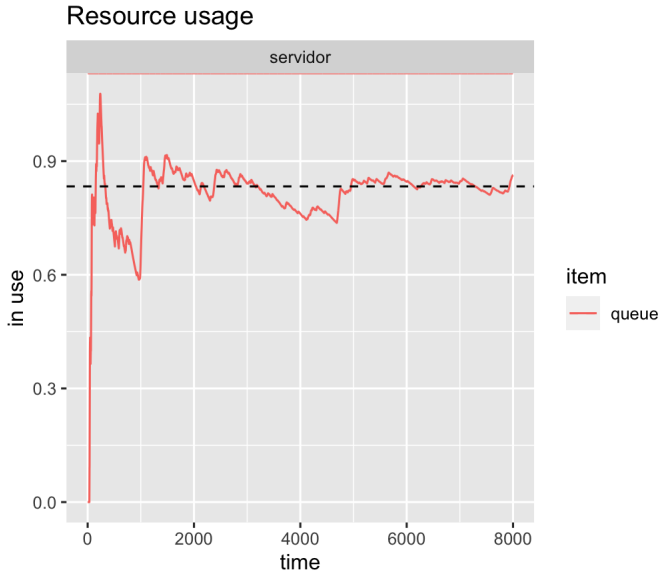

Figura 4.17: Convergencia del número medio de clientes en la cola.

En este penúltimo gráfico (Figura 4.18, se representa el número de clientes que están siendo atendidos. La línea discontinua representa la diferencia de los valores teóricamente calculados L y L_q.

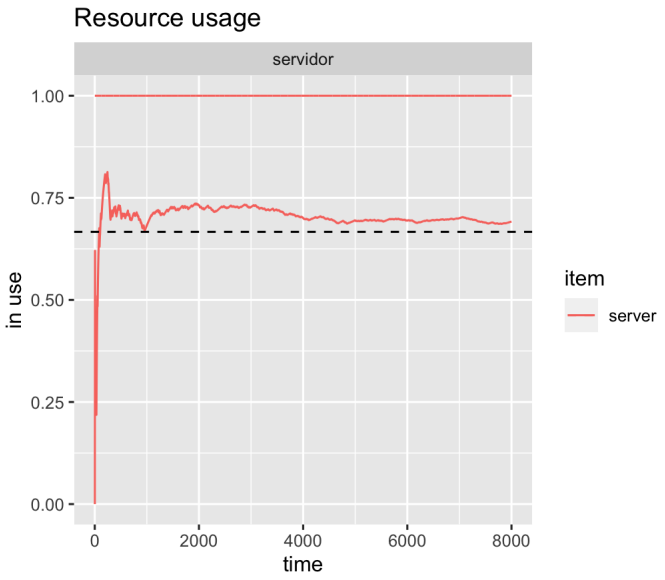

Figura 4.18: Convergencia de la diferencia del número medio de clientes en el sistema y la cola.

Por último, obtenemos un gráfico (Figura 4.19) en el que el eje X se representa el único recurso, mientras que en el eje Y, observamos el porcentaje de tiempo que está en uso, es decir, atendiendo a un cliente.

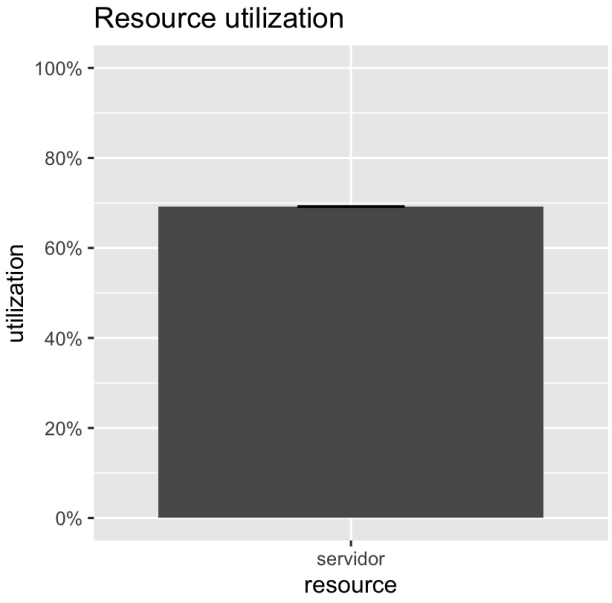

Figura 4.19: Porcentaje de tiempo que es utilizado cada servidor.

En el caso de querer simular un sistema $E_K|\mathrm{M}|\mathrm{c}$, en el que los tiempos de servicio se distribuyen exponencialmente, mientras que los tiempos entre llegadas siguen una Erlang de parámetro de forma k, también podemos recurrir al paquete *simmer*. Definiendo la trayectoria de un cliente que entra en el sistema de la siguiente forma:

```
R>trajectory("Recorrido del cliente")%>%
      seize("servidor", amount = 1)%>%
```

```
        timeout(function() rexp(1, mu))%>%
        release("servidor", amount = 1) ->trayectoria
```

Y el entorno de la simulación:

```
R>simmer("Sistema E|M|c")%>%
      add_resource("servidor", capacity = c)%>%
      add_generator("llegadas", trayectoria,
                    function() rgamma(1, shape=k, rate =k*lambda)) ->sistema
```

Podemos estimar las medidas de rendimiento de igual forma que con un sistema M|E_K||c. Supongamos el ejemplo 7 planteado en el apartado 3.8 "Sistemas E_K|M|c", cuyos datos se encuentran en la Tabla 4.7. De esta forma, los resultados de las estimaciones de las cuatro métricas, realizando una única simulación, son los siguientes:

Tabla 4.15: Medidas de rendimiento de un modelo E_3|M|2 estimadas mediante simulación

W	W_q	L	L_q
0.7768864	0.4463044	3.884432	2.231522

4.2.3. Diseño de una aplicación web con RShiny

El paquete *shiny* (Chang, 2020) es una herramienta que permite a los usuarios crear aplicaciones web accesibles para aquellos que no manejan este lenguaje de programación. Una de las ventajas de este tipo de aplicaciones es que no es necesario ser experto en el diseño web para crearlas. Toda app generada con *shiny* cuenta con dos elementos principales: la interfaz de usuario o UI, y el servidor.

La interfaz es el espacio donde el usuario definirá los 'inputs' a través de widgets, es decir, los datos y variables con los que se ejecutará posteriormente el código, y se mostrarán los 'outputs', que son las salidas que ofrece la aplicación. Por otro lado, el servidor recibe los inputs proporcionados en la UI y ejecuta la lógica que hay "detrás" de los outputs.

En nuestro caso, el objetivo de la aplicación será simular un sistema de colas con las características que el usuario introduzca y estimar las medidas de rendimiento.

En primer lugar, debemos diseñar la interfaz de usuario. Con la función `sidebarLayout` definimos la estructura de nuestra aplicación, que estará formada por una barra lateral, generada con la función `sidebarPanel`, y un panel central generado con la función `mainPanel`, donde se mostrarán los outputs.

En la barra lateral se encontrarán los widgets para que el usuario defina las variables correspondientes. Diseñaremos tres tipos de entradas diferentes. Los valores de las variables 'lambda' , 'mu', 'c', 'k' y 'replicaciones' serán inputs numéricos generados con la función `numericInput`. Por ejemplo, para la tasa de llegadas por unidad de tiempo:

```
R>numericInput("lambda", "Número medio de llegadas por unidad de tiempo",
                    value = 1, min = 0)
```

En primer lugar se define el id de la variable, seguido de su etiqueta. El parámetro 'value' será el valor por defecto de 'lambda' y 'min' el número mínimo que se le puede atribuir.

Para que el usuario pueda elegir qué metodo de simulación quiere utilizar (simmer o queuecomputer) y qué tipo de sistema es (E_K|M|c o M|E_K|c), utilizaremos dos widgets de botones para escoger

una de las opciones, como por ejemplo:

```
R>radioButtons("method", "Método de simulación:",
               choices = c("simmer", "queuecomputer"),
               selected = "simmer")
```

La variable 'method' define el método de simulación elegido, que por defecto será simmer, y 'choices' es la lista de opciones. Para seleccionar el tipo de sistema, la variable con la que trabajaremos se llamará 'distribution' y M|E|c será la opción por defecto.

Como se comentó en los apartados anteriores, cada método de simulación cuenta con un método de parada distinto. En el método simmer se trata de un tiempo de reloj máximo definido por 'tmax', miestras que con queuecomputer se simulan hasta 'nmax' clientes que entran en el sistema. Mediante paneles condicionales, el usuario podrá definir la variable correspondiente al método seleccionado.

```
R>conditionalPanel(
        condition = "input.method == 'queuecomputer'",
        numericInput("nmax", "Número máximo de clientes", value =1000,
                     min = 0)),
R>conditionalPanel(
        condition = "input.method == 'simmer'",
        numericInput("tmax", "Tiempo máximo de reloj", value = 100,
                     min = 0))
```

El último paso para diseñar la interfaz de usuario, será definir las salidas que nos devolverá la aplicación y que se mostrarán en el panel central. En este caso, obtendremos un único output, llamado 'medidas_rendimiento2', tipo texto, que contiene las estimaciones de las medidas de rendimiento del sistema simulado.

En segundo lugar, debemos desarrollar el servidor, donde se calculará nuestro único output. Antes de comenzar con la simulación, cargamos todas las librerías necesarias, fijamos una semilla y comprobamos, mediante un if, si el sistema se satura, es decir, calculamos la tasa de ocupación ρ y nos aseguramos de que sea menor que 1.

El resto de nuestro código estará dividido en cuatro bloques mediante instrucciones if, dependiendo del método de simulación y el tipo de sistema que el usuario haya seleccionado:

```
R>if (input$method == "queuecomputer" && input$distribution == "E|M|c")
R>if (input$method == "queuecomputer" && input$distribution == "M|E|c")
R>if (input$method == "simmer" && input$distribution == "E|M|c")
R>if (input$method == "simmer" && input$distribution == "M|E|c")
```

Para las cuatro combinaciones, se estimarán las medidas de rendimiento mediante replicaciones de la simulación para una mayor precisión, de la misma forma que se explicó en los apartados anteriores. En el caso de queuecomputer, las réplicas se ejecutan mediante un bucle for, en el que se simula el sistema, se calculan las medidas de rendimiento y se almacenan en vectores. Así, obtenemos cuatro vectores, uno por cada estimación. Finalmente, calculamos las medias de cada vector y almacenamos los resultados en una lista llamada 'medidas_rendimiento'.

```
R>medidas_rendimiento <- list(W = mean(W_vec), Wq = mean(Wq_vec),
```

```
                                    L = mean(L_vec), Lq = mean(Lq_vec))
```

En el caso del método simmer, utilizamos la función `lapply`.

Tras replicar la simulación, dicha función nos devuelve una lista de listas (‘medidas_rendimiento3) con las estimaciones de W, W_q, L y L_q de cada réplica. Convertimos este objeto en un data frame y calculamos las medias.

```
R>df_medidas <- do.call(rbind, lapply(medidas_rendimiento3, as.data.frame))
R>medidas_rendimiento <- apply(df_medidas, 2, mean, na.rm = TRUE)
```

En todos los casos, obtenemos un output de la forma:

```
R>return(cat(
      " W = ", round(medidas_rendimiento[[1]],6), " unidades de tiempo
      pasa un cliente de media en el sistema. \n",
      "Wq = ", round(medidas_rendimiento[[2]],6), " unidades de tiempo
      pasa un cliente de media en la cola. \n",
      "L = ", round(medidas_rendimiento[[3]],6), " clientes hay de media
      en el sistema. \n",
      "Lq = ", round(medidas_rendimiento[[4]],6), " clientes hay de media
      en la cola. "))
```

Por último, con la función `shinyApp(ui = ui, server = server)` establecemos la conexión entre la interfaz de usuario definida (ui) y el código que gestiona la lógica y las salidas de la aplicación (server).

4.2.4. Manual para la aplicación

Bienvenido a la aplicación de simulación de sistemas de colas G|G. Esta aplicación le permite simular sistemas de colas utilizando varios parámetros que puede ajustar según sus necesidades y obtener una estimación de sus medidas de rendimiento. A continuación le proporcionamos una guía paso a paso sobre cómo utilizar esta herramienta.

Al abrir la aplicación, se encontrará con un panel donde puede ingresar los parámetros necesarios para realizar la simulación, junto con algunas pautas para introducir los datos, como se puede ver en la Figura 4.20.

Figura 4.20: Menú principal de la aplicación.

A continuación, se describen los campos que debe completar:

- **Número medio de llegadas por unidad de tiempo (lambda):** Este parámetro representa la tasa promedio de llegadas al sistema por unidad de tiempo. Debe ser un número positivo.
- **Número medio de servicios por unidad de tiempo (mu):** Este parámetro indica el número promedio de servicios completos por unidad de tiempo. Debe ser un número positivo.
- **Número de servidores:** Cantidad de servidores disponibles en el sistema de colas. Debe ser un número positivo.
- **Parámetro de forma de la distribución de Erlang:** Este valor indica el número de fases por las que pasa un cliente antes de completar su servicio. Debe ser un número entero positivo.
- **Número de replicaciones de la simulación:** Este campo determina el número de veces que se repetirá la simulación para obtener resultados más precisos. No existe un único valor válido; a mayor número de replicaciones, mejor será la precisión de los resultados. Sin embargo, llega un punto en el que aumentar el número de replicaciones no es eficiente, ya que el tiempo de cálculo aumenta significativamente sin una mejora proporcional en la precisión.
- **Método de simulación:** Cuenta con dos métodos de simulación. ‘Simmer’ le permite determinar el tiempo de reloj total que quiere que simule el algoritmo, mientras que con ‘queuecomputer’ elegirá el número de clientes total que debe procesar.
- **Tiempo máximo de reloj:** Si escoge el método ‘simmer’, este widget tomará el nombre de “Tiempo máximo de reloj”, que deberá ser un número positivo. Si por el contrario escoge el método ‘queuecomputer’, cambiará de forma automática a “Número máximo de clientes” como puede ver en la Figura 4.21. En este caso, además de positivo, debe ser un número entero.

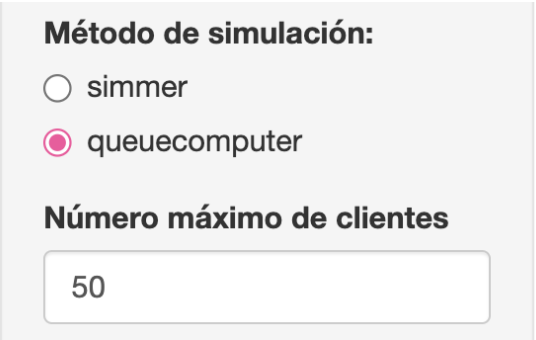

Figura 4.21: Cambio del menú con el método 'queuecomputer'.

- **Tipo de sistema:** Por último, debe especificar el tipo de sistema que desea simular. En la opción E|M|c, los tiempos de servicio se distribuyen exponencialmente, mientras que los de llegada siguen una distribución de Erlang. Por el contrario, en un sistema M|E|c, los tiempos entre llegadas siguen una distribución Exponencial y los de servicio una Erlang.

Finalmente, para proporcionar al lector una visión clara de las salidas de esta aplicación, en la imagen 4.22 se presentan los resultados del siguiente ejemplo:

En el aeropuerto de Madrid ofrecen un servicio de lavado de coches con una única cabina a la que llegan una media de 6 coches/hora. Cada cliente tarda en limpiar el coche 4 minutos, con una desviación típica de $\sigma = \sqrt{8}$ minutos.

Aproximamos los tiempos de servicio a una distribución Erlang de parámetro de forma $k = 2$. Tras introducir los datos en la aplicación y utilizando el método 'simmer' con 1000 réplicas y un tiempo máximo de relojk de 1000, obtenemos la siguiente salida:

```
W =  6.922841  unidades de tiempo pasa un cliente de media en el sistema.
Wq =  2.733646  unidades de tiempo pasa un cliente de media en la cola.
L =  0.692284  clientes hay de media en el sistema.
Lq =  0.273365  clientes hay de media en la cola.
```

Figura 4.22: Ejemplo de salida de la aplicación.

Capítulo 5

Conclusiones

Tras la investigación y los análisis realizados hemos podido concluir:

- Cuando se trata de modelos G|G, en muy pocos casos se ha llegado a determinar una expresión genérica para calcular las medidas de rendimiento. En consecuencia, es necesario recurrir a métodos numéricos complejos para cada caso concreto.
- Existe una distribución alternativa llamada Erlang, a la cual pueden aproximarse casi todos los modelos G|G. Esta distribución es muy útil para trabajar cuando no se conoce específicamente la distribución empírica de los tiempos entre llegadas y/o de servicio, algo bastante común en este tipo de estudios.
- Se ha demostrado que en situaciones cotidianas de la vida real, en las que a priori no queda patente que se trate de un modelo de colas, sí que resulta posible aplicar esta teoría.
- La simulación supone una herramienta muy útil para analizar y estimar medidas de rendimiento en sistemas de colas, ya sea cuando las distribuciones empíricas son conocidas, pero el cálculo de las métricas es analíticamente complejo; o cuando las distribuciones específicas de los tiempos entre llegadas y/o de servicio son desconocidas. Su capacidad para modelar la complejidad de estos procesos de forma flexible y validar resultados teóricos bajo diferentes escenarios hace de la simulación una opción eficaz y versátil en la investigación y optimización de sistemas de colas.
- Actualmente, se han desarrollado múltiples librerías y métodos para la simulación de eventos discretos en cada lenguaje de programación. Los paquetes 'queuecomputer' y 'simmer' de R, han resultado ser herramientas eficaces para el análisis de distintos tipos de sistemas G|G. Pese a no existir apenas diferencias entre ellos, la simulación con 'simmer' ofrece algo más de libertad a la hora de definir la trayectoria y el entorno de simulación.
- Realizar múltiples replicaciones de una simulación mejora significativamente la precisión de los resultados obtenidos. Esta práctica no solo aumenta la robustez de las conclusiones alcanzadas, sino que también proporciona una mayor confianza en la validez y fiabilidad de las estimaciones de rendimiento de los sistemas analizados.

Bibliografía

Banks, J., Carson II, J. S., Nelson, B. L., & Nicol, D. M. (2010). *Discrete-event system simulation*. Pearson Education. Fourth edition.

Bertoli, M., Casale, G., & Serazzi, G. (2009). JMT: Performance Engineering Tools for System Modeling. *ACM SIGMETRICS Performance Evaluation Review*, *36*(4), 10-15.

Chang, W. (2020). *shiny: Web Application Framework for R* [Accessed: 15-06-2023]. https://cran.r-project.org/web/packages/shiny/index.html

Cohen, J. W. (1973). Some results on regular variation for distributions in queueing and fluctuation theory. *Journal of applied probability*, *10*(2), 343-353.

Ebert, A., Wu, P., Mengersen, K., & Ruggeri, F. (2020). Computationally Efficient Simulation of Queues: The R Package queuecomputer. *Journal of Statistical Software*, *95*(5), 1-29.

Erlang, A. K. (1909). The theory of probabilities and telephone conversations. *Nyt Tidsskrift for Matematik*, *90*, 33.

Gross, D., Shortle, J. F., Thompson, J. M., & Harris, C. M. (2008). *Fundamentals of Queueing Theory*. John Wiley & sons, Inc. Fourth edition.

Hillier, F. S., & Lieberman, G. J. (2009). *Introduction to Operations Research*. McGraw-Hill. Ninth edition.

Hoad, K., Robinson, S., & Davis, R. (2010). Automated selection of the number of replications for a discrete-event simulation. *Journal of the Operational Research Society*, *61*(11), 1632-1644.

Khintchine, A. Y. (1932). Mathematical theory of a stationary queue. *Matematicheskii Sbornik*, *39*(4), 73-84.

Kleinrock, L. (1961). Information flow in large communication nets. *RLE Quarterly Progress Report*, *1*.

Law, A. M. (2024). *Simulation modeling and analysis*. McGraw-Hill. Sixth Edition.

Matloff, N. (2008). *Introduction to discrete-event simulation and the simpy language* (Vol. 2). Davis, CA. Dept of Computer Science. University of California at Davis.

Pollaczek, F. (1930). Über eine Aufgabe der Wahrscheinlichkeitstheorie. *Mathematische Zeitschrift*, *32*, 64-100.

R Core Team. (2024). *Package 'parallel'* [Accessed: 15-06-2023]. https://stat.ethz.ch/R-manual/R-devel/library/parallel/doc/parallel.pdf

Robinson, S. (1994). *Successful simulation: a practical approach to simulation projects*. McGraw-Hill Book Co Ltd.

Ross, S. M. (1978). Average delay in queues with non-stationary Poisson arrivals. *Journal of applied probability*, *115*(3), 602-609.

Schmidt, J. W., & Taylor, R. E. (1970). *Simulation and analysis of industrial systems*. R. D. Irwin.

Shannon, R. E. (1975). *Systems Simulation: The Art and Science*. Prentice Hall.

Ucar, I., & Smeets, B. (2023). *simmer.plot: Plotting Methods for 'simmer'* [https://r-simmer.org, https://github.com/r-simmer/simmer.plot].

Ucar, I., Smeets, B., & Azcorra, A. (2019). simmer: Discrete-Event Simulation for R. *Journal of Statistical Software*, *90*(2), 1-30.

Winston, W. L. (2003). *Operations research. Applications and algorithms*. Thomson Learning, Inc. Fourth Edition.