UNIVERSIDAD DE GRANADA

Simulación de Sistemas

grado en ingeniería informática

Práctica 3

Modelos de Simulación Dinámicos y Discretos

Autor
José María Sánchez Guerrero

Rama
Computación y Sistemas Inteligentes

Escuela Técnica Superior de Ingenierías Informática y de Telecomunicación

Curso 2019-2020

Índice general

Capítulo 1

Mi segundo modelo de simulación Discreto

Nuestro modelo de simulación consistirá en un servidor que presta un determinado servicio a una serie de clientes, los cuales solicitarán dicho servicio periódicamente. Cuando llega un cliente y el servidor no está ocupado, será atendido inmediatamente; en caso contrario, el cliente tendrá que esperar en la cola. Cuando se completa un servicio, el servidor elegirá al siguiente en una forma FIFO.

Al empezar la simulación, no habrá clientes esperando y el servidor está libre. Utilizaremos el mismo generador exponencial tanto para el tiempo que tardarán en llegar los clientes, como el tiempo que tardará el servidor en atender a cada uno.

1.1. Simulación con incremento fijo de tiempo

En esta simulación, vamos a tratar al tiempo incrementándolo de unidad en unidad. Para evitar problemas con el manejo del tiempo, tendremos que modificar los generadores de datos para que nos devuelvan los valores redondeados al entero más próximo. Si obtenemos un valor igual a 0, devolveremos 1 en su lugar, ya que el suceso generado quedaría en un tiempo anterior al actual, que generamos al incrementar en una unidad.

Este será nuestro código resultante, que nos servirá tanto para generar el tiempo de llegada del cliente como para generar el timepo del servicio (sólo tendremos que modificar la variable *tlleg* por *tserv*):

```
float generallegada(float tlleg){
  float u = random();          // o tambien rand() en lugar de random()
  u = ( u / (RAND_MAX+1.0) ); //RAND_MAX es una constante del sistema
  u = round( -tlleg * log(1-u) );

  if (u != 0)
    return u;
  else
    return 1.0;
}
```

Para la simulación vamos a emplear diferentes unidades de medida de tiempo (horas, minutos, segundos...) y con un número de clientes a atender bastante alto, de unos 10.000, para que los resultados sean robustos. Este es el resultado que obtenemos:

tiempo	**tlleg**	**tserv**	**% tiempo ocioso**	**Media clientes en cola**
horas	0.15	0.1	0.019994	0.000000
medias horas	0.3	0.2	0.625621	0.078153
cuartos de hora	0.6	0.4	7.224885	0.356633
minutos	9	6	31.501728	1.647961
segundos	540	360	33.934223	1.282934

Como podemos ver, los porcentajes de tiempo ocioso del servidor tienen una variación bastante grande. Esto se debe a que los valores pequeños (unidades de tiempo más altas como las horas o la medias horas) producen unos valores muy próximos a cero en los generadores y, en consecuencia, son redondeados. Como ya hemos visto, estos valores no serán devueltos como 0, si no como 1.

Con esto estamos diciendo que un suceso dura más de lo que realmente es, y vamos acumulando este error en toda la simulación. Esta es la razón por la cual los modelos de incremento fijo no son los más apropiados, ya que sus valores son bastante diferentes a los que obtendríamos sobre el papel.

A continuación, vamos a mostrar unas gráficas con diversas ejecuciones para comprobar que los resultados obtenidos son coherentes y ver mejor las diferencias que obtenemos al utilizar distintas medidas tiempo:

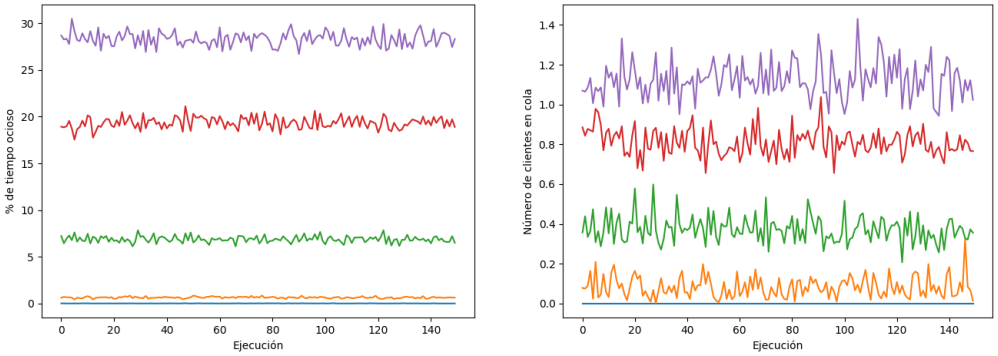

Figura 1.1: Porcentaje de tiempo ocioso del servidor y media de clientes en cola

Aqui podemos observa cómo las gráficas van teniendo un porcentaje de ocupación y un número de clientes en cola más alto a medida que disminuimos las valores *tlleg* y *tserv*. También podemos observar que, pese a que cada una de las líneas rondan un rango de valores, existe cierta variabilidad en los resultados; por lo cual sería un poco irresponsable fiarnos de un solo dato como el de la tabla.

1.2. Simulación con incremento variable de tiempo

Ahora vamos a tratar el mismo problema pero con un incremento de tiempo variable, para hacerlo más eficiente y preciso. En este caso, la variable no tendrá ni por qué ser un entero ni tendremos que parsear los generadores de datos, ya que se incrementará su valor hasta el suceso más cercano. Decimos que es más preciso porque porque no tiene que dar los saltos que daba el anterior modelo y, en consecuencia, gana en eficiencia ya que va suceso a suceso.

Esta es la nueva línea de código que tendremos que añadir (también quitaremos la que aumenta el tiempo de manera uniforme $reloj++$):

```
1 reloj = min(tiempo_llegada, tiempo_salida)
```

A continuación, vamos a ejecutar este modelo de la misma forma que hemos hecho con el modelo anterior y óbservar los cambios que se obtienen:

tiempo	tlleg	tserv	% tiempo ocioso	Media clientes en cola
horas	0.15	0.1	33.993752	1.260754
medias horas	0.3	0.2	33.555038	1.250663
cuartos de hora	0.6	0.4	34.353592	1.382941
minutos	9	6	34.451107	1.211860
segundos	540	360	33.695103	1.261086

Vemos que los resultados esta vez han sido más regulares, ya que para todas las medidas de tiempo se han obtenido unos valores entre el 33 y el 34 por ciento. A diferencia del modelo anterior, en este podríamos decir que los resultados son bastante más fiables independientemente de si utilizamos horas, minutos, segundos... Esto es debido a que ahora no tenemos la acumulación del error que teníamos anteriormente, y los sucesos se producen siempre en el momento declarado, sin ponderar.

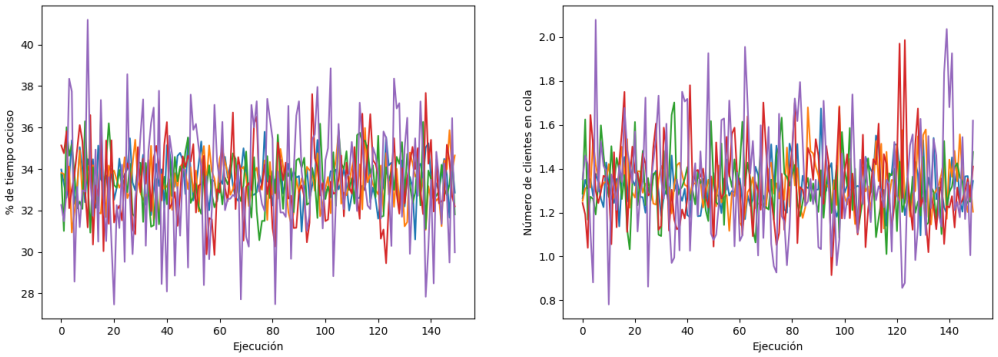

Figura 1.2: Porcentaje de tiempo ocioso del servidor y media de clientes en cola

Si observamos ahora las gráficas con las diversas ejecuciones, vemos que están prácticamente todas unas encima de otras. Como acabamos de decir, ya no tenemos este error en las distintas medidas de tiempo. No obstante, vemos que aún tenemos bastante variabilidad en los datos, llegando algunos a máximos como 42 o a mínimos como 27.

Por otra parte, esta mejora no sólo afecta a las medidas de tiempo, si no que también afecta a la eficiencia. En el modelo anterior, los incrementos en el reloj dependen de la medida del tiempo; mientras que en el modelo actual, el número de incrementos dependerá del número de sucesos. Vamos a comprobarlo extrayendo una tabla que compare los tiempos de ejecución para todas las medidas con incremento fijo, con las mismas medidas en incremento variable:

tiempo	tlleg	tserv	tiempo fijo	tiempo variable
horas	0.15	0.1	0.001671	0.002071
medias horas	0.3	0.2	0.002937	0.002562
cuartos de hora	0.6	0.4	0.003491	0.001119
minutos	9	6	0.003358	0.001464
segundos	540	360	0.020543	0.002939

Podemos ver que el tiempo de ejecución en el modelo con incremento fijo va aumentando lo que parece lineal o exponencialemente, mientras que en el modelo con incremento variable tenemos unos tiempos prácticamente iguales en unas ejecuciones y otras.

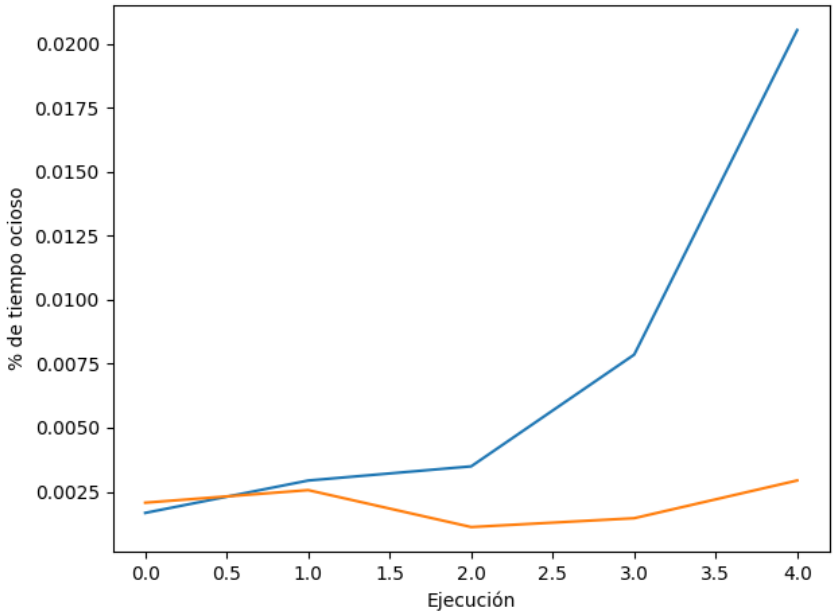

Figura 1.3: En azul los tiempos con incremento fijo y, en naranja, variable.

En la gráfica lo podemos corroborar. Tras estudiar los dos modelos, concluímos que utilizar uno de incremento variable sería lo acertado, no sólo por la mayor eficiencia en tiempo que nos da, si no por algo más importante como es la precisión y calidad al trabajar con distintos tipos de datos o unidades de medida.

1.3. Programa de simulación dinámico y discreto

En esta sección vamos a abordar un problema similar al anterior, pero con una cantidad de m servidores idénticos trabajando en paralelo. La cola de clientes seguirá siendo la misma, tratando a los clientes del mismo método FIFO. Si utilizamos más de un servidor en complicado, pero con $m = 1$ es posible obtener los valores teóricos de casi todas las medidas de rendimiento.

Vamos a ejecutar el programa para ver qué resultamos obtenemos. Lo haremos con unos parámetros como los anteriores ($parada = 10.000$, $tlleg = 9$ y $tserv = 6$):

```
jose@jose-Lenovo-G50-70:~/Universidad/SS (Simulacion de Sistemas)/SS_Practicas/P3$
Tiempo medio de espera en cola = 12.840
Tiempo medio de estancia en el sistema = 18.840
Numero medio de personas en cola = 1.425
Numero medio de personas en el sistema = 2.086
Longitud media de colas no vacias = 3.261
porcentaje medio de tiempo de ocio por servidor = 33.960
Longitud maxima de la cola = 29
```

Figura 1.4: Ejecución del programa colammk con $m = 1$.

Una vez obtenidos estos valores, vamos a contrastarlos con los teóricos. Lo primero es calcularlos:

- Tiempo medio de espera en cola $= \frac{tserv^2}{tlleg - tserv} = \frac{6^2}{9-6} = \frac{36}{3} = 12$
- Tiempo medio de estancia en el sistema $= \frac{tserv \cdot tlleg}{tlleg - tserv} = \frac{6 \cdot 9}{9-6} = \frac{54}{3} = 18$
- Número medio de clientes en cola $= \frac{tserv^2}{tlleg(tlleg - tserv)} = \frac{6^2}{9 \cdot (9-6)} = \frac{36}{27} = 1.3$
- Número medio de clientes en el sistema $= \frac{tserv}{tlleg - tserv} = \frac{6}{9-6} = \frac{6}{3} = 2$
- Longitud media de colas no vacías $= \frac{tlleg}{tlleg - tserv} = \frac{9}{9-6} = \frac{9}{3} = 3$
- Porcentaje de tiempo de ocio del servidor $= \left(1 - \frac{tserv}{tlleg}\right) \cdot 100 = \left(1 - \frac{6}{9}\right) \cdot 100 = \frac{100}{3} = 33.3$

Como podemos ver, los resultados se ajustan bastante bien a los que esperábamos teóricamente. Esto se debe, en parte, a que la simulación se ha realizado con un tiempo de parada muy alto, es decir, la simulación ha durado bastante tiempo y las medias que se obtienen son lo suficientemente representativas.

Si probamos con un tiempo de parada bajo, los resultados serán más impredecibles o variables, como se muestra a continuación:

```
jose@jose-Lenovo-G50-70:~/Universidad/SS (Simulacion de Sistemas)/SS_Practicas/
Tiempo medio de espera en cola = 4.842
Tiempo medio de estancia en el sistema = 10.842
Numero medio de personas en cola = 0.996
Numero medio de personas en el sistema = 1.740
Longitud media de colas no vacias = 1.947
porcentaje medio de tiempo de ocio por servidor = 25.629
Longitud maxima de la cola = 5
```

Figura 1.5: Ejecución del programa colammk con $m = 1$ y tiempo bajo.

Ahora vamos a aumentar el número de servidores m, pero manteniendo un equilibrio. El tiempo de servicio dividido por el número de servidores tiene que permanecer constante. Vamos a modificar el programa para que repita varias veces la simulación y calcule las medias y desviaciones típicas de las medidas de rendimiento.

Las líneas más importantes que se han modificadas son las siguientes:

```
void fin(){
  ....
  // Variables nuevas para medias y desviaciones
  retrasoMedio += retrasomedio;
  estanciaMedia += estanciamedia;
  enColaMedio += encolamedio;
  enSistemaMedia += ensistemamedio;
  colasMedia += colasnovaciasmedio;
  porcentajeOcioMedio += porcentajemedioocio;
  longMaxCola += maximacola;

  retrasoMedioDes += pow(retrasomedio, 2);
  estanciaMediaDes += pow(estanciamedia, 2);
  enColaMedioDes += pow(encolamedio, 2);
  enSistemaMedioDes += pow(ensistemamedio, 2);
  colasMediaDes += pow(colasnovaciasmedio, 2);
  porcentajeOcioMedioDes += pow(porcentajemedioocio, 2);
  longMaxColaDes += pow(maximacola, 2);
}
```

```
float calcularDesviacion(float suma, float media)
{
    float desviacion = suma - (n_simulaciones * pow(media, 2));
    desviacion = desviacion / (n_simulaciones - 1);

    return sqrt(desviacion);
}
```

Por último ya tenemos el bucle for, la división entre el número de simulaciones para hacer la media y desviaciones típicas, y posteriormente imprimimos los resultados.

La prueba de este programa la vamos a realizar con varios números de servidores, sin embargo, el resto de parámetros que usaremos serán los mismos que los de la ejecución con $m = 1$, ya que los resultados obtenidos eran muy similares a los valores teóricos. En este caso vamos a realizar la ejecución con 2, 3, 4 y 5 servidores, adaptando el *tserv* a cada uno de ellos para mantener el equilibrio. Este ha sido el resultado:

Número de servidores	**2**	**3**	**4**	**5**
Tiempo medio espera en cola	e	e	e	e
Tiempo medio estancia en el sistema	e	e	e	e
Numero medio personas en cola	e	e	e	e
Número medio personas en el sistema	e	e	e	e
Longitud media colas no vacías	e	e	e	e
Porcentaje medio tiempo de ocio	e	e	e	e
Longitud máxima cola	e	e	e	e

Podemos ver que a medida que aumentamos el número de servidores, los resultados van siendo mejores. Los tiempos medios de espera en cola se reducen, con menos clientes en cola y, a su vez, un número mayor de clientes en el sistema. Por otro lado, podemos ver que el porcentaje medio de tiempo de ocio del servidor, es prácticamente el mismo en todos los casos (teniendo en cuenta siempre un cierto error). Esto quiere decir que el sistema está bien equilibrado y a su vez, deducimos que estamos desperdiciando potencia del servidor; es decir, que pese a tener un número alto de servidores, no tenemos los suficientes clientes para que funcionen eficientemente o, simplemente, no los distribuímos de una forma eficaz.

Para finalizar con este modelo de simulación, vamos a ver que sucede si reemplzamos los generadores actuales por generadores determinísticos y uniformes. El nuevo código implementado para el determinístico y para el uniforme, respectivamente, es:

```
// Deterministico. Devuelve siempre el valor medio introducido
float generador_deterministico(float media)
{
  return media;
}

// Uniforme. Devuelve un valor uniformemente generado con media
// en el valor introducido
float generador_uniforme(float media)
{
  float u;
  u = (float) random();
  u = u/(float)(RAND_MAX+1.0);
  return(media*2*u);
}
```

Capítulo 2

Mi tercer modelo de simulación Discreto

2.1. Remolcador de un puerto

2.2. Mejoras propuestas para el remolcador

Capítulo 3

Análisis de Salidas y Experimentación

3.1. ¿Cuánto hay que simular?

3.2. Intervalos de confianza

3.3. Comparación de más de dos sistemas