Kojo's Kitchen

Daniel E. Cordovés Borroto C411

Email: daniel.cordovesb@estudiantes.matcom.uh.cu

Github: https://github.com/dcordb/kojos-kitchen/

28 de marzo de 2020

Problema a resolver

La cocina de Kojo es uno de los puestos de comida rápida en un centro comercial. El centro comercial está abierto entre las 10:00 am y las 9:00 pm cada día.

En este lugar se sirven dos tipos de productos: sándwiches y sushi. Para los objetivos de este proyecto se asumirá que existen solo dos tipos de consumidores: unos consumen solo sándwiches y los otros consumen solo productos de la gama del sushi.

En Kojo hay dos períodos de hora pico durante un día de trabajo; uno entre las 11:30 am y la 1:30 pm, y el otro entre las 5:00 pm y las 7:00 pm. El intervalo de tiempo entre el arribo de un consumidor y el de otro no es homogéneo pero, por conveniencia, se asumirá que es homogéneo. El intervalo de tiempo de los segmentos homogéneos, distribuye de forma exponencial.

Actualmente dos empleados trabajan todo el día preparando sándwiches y sushi para los consumidores. El tiempo de preparación depende del producto en cuestión. Estos distribuyen de forma uniforme, en un rango de 3 a 5 minutos para la preparación de sándwiches y entre 5 y 8 minutos para la preparación de sushi.

El administrador de Kojo está muy feliz con el negocio, pero ha estado recibiendo quejas de los consumidores por la demora de sus peticiones. Él está interesado en explorar algunas opciones de distribución del personal para reducir el número de quejas. Su interés está centrado en comparar la situación actual con una opción alternativa donde se emplea un tercer empleado durante los períodos más ocupados. La medida del desempeño de estas opciones estará dada por el porciento de consumidores que espera más de 5 minutos por un servicio durante el curso de un día de trabajo.

Se desea obtener el porciento de consumidores que esperan más de 5 minutos cuando solo dos empleados están trabajando y este mismo dato agregando un empleado en las horas pico.

Solución

Para darle solución a este problema se realizan dos simulaciones: una con solo dos empleados; y otra con dos empleados y un tercero en los horarios pico, luego se comparan los porcientos de clientes que esperan más de 5 minutos en cada simulación para sacar conclusiones sobre la eficiencia en Kojo's Kitchen.

Se tienen las siguientes variables:

De tiempo:

- `t` (tiempo actual en el sistema)

Estado del sistema:

- `n`: número de personas en el sistema
- `who_first`: número de cliente que está siendo atendido por el empleado 1 (0 indica que no hay cliente siendo atendido)
- `who_second`: número de cliente está siendo atendido por el empleado 2 (0 indica que no hay cliente siendo atendido)
- `who_third`: número de cliente está siendo atendido por el empleado 3 (0 indica que no hay cliente siendo atendido)
- `t_out_1`: tiempo después del cual el cliente `who_first` es atendido por el empleado 1 (infinito si no hay cliente siendo atendido)
- `t_out_2`: tiempo después del cual el cliente `who_second` es atendido por el empleado 2 (infinito si no hay cliente siendo atendido)
- `t_out_3`: tiempo después del cual el cliente `who_third` es atendido por el empleado 3 (infinito si no hay cliente siendo atendido)

Contadoras:

- `arrivals`: cantidad de llegadas
- `first_empl_req`: cantidad de pedidos completados por empleado 1
- `second_empl_req`: cantidad de pedidos completados por empleado 2
- `third_empl_req`: cantidad de pedidos completados por empleado 3

De salida:

- `t_in`: tiempo de llegada de cada cliente
- `t_out`: tiempo de salida de cada cliente
- `food_wants`: tipo de comida que quiere cada cliente

Los eventos serían:

- llegada de un nuevo cliente
- primer empleado terminó pedido de algún cliente
- segundo empleado terminó pedido de algún cliente
- tercer empleado terminó pedido de algún cliente (nota: este empleado solo trabaja en horarios pico)

En la solución la unidad de tiempo es minutos. Por consiguiente se reescriben los intervalos de tiempo de esta forma:

- tiempo en que abre la cocina: `t` = 0 (10:00 am)
- tiempo de inicio de primer intervalo de hora pico: `t` = 90 (11:30 am)
- tiempo de finalización de primer intervalo de hora pico: `t` = 90 + 120 (1:30 pm)
- tiempo de inicio de segundo intervalo de hora pico: `t` = 7 * 60 (5:00 pm)
- tiempo de finalización de segundo intervalo de hora pico: `t` = 7 * 60 + 120 (7:00 pm)
- tiempo de cierre de la cocina: `t` = 11 * 60 (9:00 pm)

Las llegadas de un nuevo cliente se realizan de la siguiente forma: se genera una variable con distribución exponencial que denota que a partir del tiempo actual (`t`) entra un nuevo cliente en G minutos (el tiempo de entrada sería `t + G` minutos), donde `G` es la variable generada.

Como hay intervalos de tiempo de hora pico e intervalos de tiempo regulares se tienen dos lambdas distintos para generar estas variables:

- `LAMBDA_REGULAR` (lambda a usar en intervalos regulares)
- `LAMDA_BUSY` (lambda a usar en intervalos de hora pico)

Por supuesto estos lambdas cumplen que `LAMBDA_BUSY` > `LAMBDA_REGULAR`.

Luego que llega un cliente se le asigna al primer empleado que lo pueda atender. Notar que el tercer empleado solo trabaja en la segunda simulación y en horarios picos. Cuando un empleado atiende a un cliente, el cliente sale del sistema y el empleado se le asigna un nuevo cliente, en particular, se le asigna el cliente máx(`who_first`, `who_second`, `who_third`) + 1.

El tiempo que un cliente i tuvo que esperar hasta recibir su comida es de `t_out[i] - t_in[i]`, queremos en cada simulación hallar el porciento de clientes cuyo tiempo de espera es mayor que 5 minutos.

Consideraciones Obtenidas

Luego de correr varias simulaciones variando los parámetros `LAMBDA_REGULAR` y `LAMBDA_BUSY` se obtuvieron los siguientes resultados:

Simulación	LAMBDA_REGULAR	LAMBDA_BUSY	Two Employees	Three Employees
1	0.2	0.3	78.34394	68.71165
2	0.2	0.3	77.84810	54.30463
3	1	0.7	99.66159	66.25222
4	0.1	0.05	51.51515	58.13953
5	0.1	0.05	55.81395	44.00000

Podemos notar que la cantidad de clientes que esperan más de 5 minutos es menor usando un tercer empleado. Sin embargo existen simulaciones en las que no es así (o están muy cercanos los porcientos), esto es entendible ya que la simulación con tres empleados pudo haber tenido más clientes entrando a la cocina en distintos momentos de tiempo. Claro esto es una situación pesimista, en general el modelo de tres empleados minimiza el tiempo de espera de los clientes.

Para más detalles sobre el modelo, realizar simulaciones con otros parámetros lambda y ver el código que lo implementa dirigirse al repo de GitHub:

https://github.com/dcordb/kojos-kitchen/

Pseudocódigo

```
function KojosKitchen(use_extra_employee)
   while True do
      cur_time = mín(next_arrival, t_out_1, t_out_2, t_out_3)
      if next_arrival ≠ inf and next_arrival = cur_time then
         t ← next_arrival
         gen food preference
         t_in ← t_in + t                                          ▷ esto es concatenación
         arrivals ← arrivals + 1
         if who_first = 0 then
            n ← n + 1
            who_first ← arrivals
            t_out_1 ← t + T_i                          ▷ T_i es el tiempo que el empleado tarda
         else if who_second = 0 then
            n ← n + 1
            who_second ← arrivals
            t_out_2 ← t + T_i                          ▷ T_i es el tiempo que el empleado tarda
         else if use_extra_employee and in_busy_hours(t) and who_third = 0 then
            n ← n + 1
            who_third ← arrivals
            t_out_3 ← t + T_i                          ▷ T_i es el tiempo que el empleado tarda
         else                                              ▷ Cliente debe esperar en la cola
            pass
         end if
         if in_busy_hours(t) then
            next_arrival ← t + G_busy       ▷ G_busy es una exponencial generada con LAMBDA_BUSY
         else
            next_arrival ← t + G_reg      ▷ G_reg es una exponencial generada con LAMBDA_REGULAR
         end if
         if next_arrival > T then          ▷ El nuevo cliente llega luego de cerrar la cocina
            next_arrival ← inf
         end if
      else if t_out_1 ≠ inf and t_out_1 = cur_time then
         first_empl_req ← first_empl_req + 1
         t ← t_out_1
         t_out[who_first] = t
         n ← n - 1
         next_guy ← máx(who_first, who_second, who_third) + 1
```

```
38:          who_first ← 0
39:          if next_guy ≤ arrivals then
40:             who_first ← next_guy
41:          end if
42:          if who_first = 0 then
43:             t_out_1 ← inf
44:          else
45:             t_out_1 ← t + T_i                          ▷ T_i es el tiempo que el empleado tarda
46:          end if
47:       else if t_out_2 ≠ inf and t_out_2 = cur_time then
48:          second_empl_req ← second_empl_req + 1
49:          t ← t_out_2
50:          t_out[who_second] = t
51:          n ← n - 1
52:          next_guy ← máx(who_first, who_second, who_third) + 1
53:          who_second ← 0
54:          if next_guy ≤ arrivals then
55:             who_second ← next_guy
56:          end if
57:          if who_second = 0 then
58:             t_out_2 ← inf
59:          else
60:             t_out_2 ← t + T_i                          ▷ T_i es el tiempo que el empleado tarda
61:          end if
62:       else if use_extra_employee and t_out_3 ≠ inf and t_out_3 = cur_time then
63:          third_empl_req ← third_empl_req + 1
64:          t ← t_out_3
65:          t_out[who_third] = t
66:          n ← n - 1
67:          next_guy ← máx(who_first, who_second, who_third) + 1
68:          who_third ← 0
69:          if next_guy ≤ arrivals then
70:             who_third ← next_guy
71:          end if
72:          if who_third = 0 then
73:             t_out_3 ← inf
74:          else
75:             t_out_3 ← t + T_i                          ▷ T_i es el tiempo que el empleado tarda
76:          end if
77:       else
78:          break
79:       end if
80:    end while
81:    cnt ← 0
82:    for all 1 ≤ i ≤ arrivals do
83:       if t_out[i] - t_in[i] > 5 then
84:          cnt ← cnt + 1
85:       end if
86:    end for
87:    return cnt * 100 / arrivals
88: end function
```